The design and implementation of a Socca Editor

Table of Contents

1 Abstract

A prototype for a Socca editor was developed in Java. This master thesis describes the design of this prototype, some of the implementation trade-offs and a few of the more significant problems that were encountered. The reasons why Java was chosen as the programming language is also discussed. The thesis ends with a list of possible extensions to the editor and future work.

2 Introduction

The goal of the project was to create a prototype of a Socca editor. Socca (Specification Of Coordinated and Cooperative Activities) is a specification formalism for the description, analysis and specification of software processes that is being developed at the Computer Science department of the University of Leiden. Socca describes a software process from four different perspectives:

1. Data perspective: the class diagram, that is, the classes of the program, their attributes and operations and the relationships between the classes.

2. Behavior perspective: The behavior of the classes (external STD). This perspective describes the way a class may be used. There is a state transition diagram for each class.

3. Functionality perspective: the functionality of the operations (internal STD). There is a state transition diagram (possibly more than one) for each operation that describes which actions are performed (and their order) when the operations is called.

4. Communication perspective: the communication between objects (paradigm)

For a more detailed description of SOCCA, see [1] and [2].

3 Problem description

The prototype of the editor partially supports the syntax of Socca. The focus of implementation of the editor is currently on the data perspective; the other three perspectives may be implemented on a later date. Although the other perspectives aren't implemented currently, the design of the editor anticipates their inclusion.

Socca is currently being developed at the University of Leiden. Because of this constant development, some areas of Socca are not thoroughly researched yet. Whenever this is the case, the Unified Modeling Language (UML) has been used as a guide. UML is likely to become the de-facto standard of modeling languages. So it is a good idea to follow UML whenever possible, unless a better alternative for certain aspects of UML are found. The result is an editor that implements UML but replaces some of the weaknesses of UML with Socca.

Pieter Jan 't Hoen (hoen@wi.leidenuniv.nl) is currently working on integrating a module concept with Socca. Socca currently has no real support for modules or packages. Once modules have been added, a Socca model can be split up it smaller parts (modules) that are each somewhat independent of each other. If the modules are chosen in such a way that the number of interdependencies between the modules is kept low then the modules can be designed mostly independent of each other, possibly by different people. If, at a later date, a multi-user version is created of the editor, it can greatly benefit from this module concept.

Work in this area isn't finished but to allow some initial experimentation with these new concepts, the editor has some very limited support for these modules. Since research is still ongoing, basing the modules of the editor on the current research was deemed problematic. Instead UML's packages were used as basis for the module concept in the editor.

UML package concept is not much more than a way to group parts of the data model together. Such a group is placed in a rectangle and the name of the package is placed at the top of the rectangle. The packages of the editor work in the same way. One concept of Pieter Jan's work was added; each data model element in a package is given a visibility icon. This visibility (public, protected or private) determines whether a data model element outside the package can have a relation with an element inside the package. If an element is public, it is visible to elements outside the package and so it can have relations with elements outside the package. If it is private, it is not visible outside the package and such relations are not possible. Protected elements are only different from private elements when one package inherits from another. Package inheritance is still work in progress and is as such not supported in the editor.

The data perspective is implemented using a graphical user interface that allows adding, deleting and editing of classes (and their attributes and operations) and packages. It also allows adding relationships between classes, editing them and removing them. These relationships include inheritance relationships, uses relationships, aggregations, associations and containment relationships (of a class within a package). The user can focus on a part of the model by opening new views and/or suppressing parts of the view.

4 The implementation

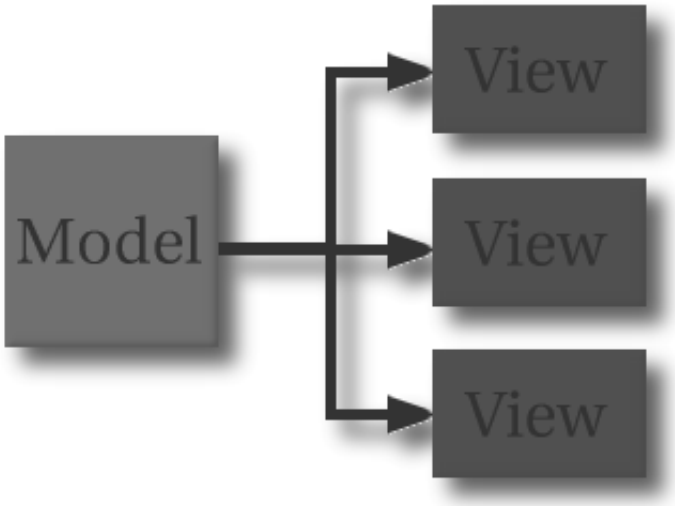
The editor is implemented in Java, version 1.1, using the Java Foundation Classes (JFC). Java was chosen because of its platform independent nature and its ability to be used within a WWW-browser.

One of the requirements of the editor was that the information in the model can be shown in multiple views. The editor had to support multiple windows that each show (a part of) the model. These views are kept consistent; changes in one view result in updates of the other views containing the same model elements. To support this type of behavior, the MVC (Module View Controller) design pattern was used to keep the views consistent.

4.1 The architecture

The editor uses the MVC (Module View Controller) concept that was first introduced by the language Smalltalk [3]. In Smalltalk the programs that used this design pattern were split up into a part that contained the data, the model, a part that presented the data to the user, the view, and a part that allowed the user to manipulate the data, the controller. The advantage of this approach is that, because there can be more than one view of a model, the data can be visualized in different ways at the same time. Another advantage is that the source code of a model or a view is easier to reuse in another program than a piece of source code that contains both the model and view as a monolithic whole.

Over time it was discovered that splitting in three parts is not always the best way. The controller has strong dependencies on the model and view parts. Because of these dependencies, the model and view It turned out to be better to split the program in a model and a view part and put the controller in the view and/or in the model. This is the approach that was taken while designing the editor. It is also the approach taken by the designers of the Java Foundation Classes (JFC, a.k.a. swing). As a result the JFC classes and the classes of the editor integrate quite naturally. This design pattern (using only model and view, no controller) is described in Design Patterns [4] under the name Observer.



The model (left) will send events (arrows in the middle) to the view (right) whenever a part of the model changes

The editor consists of three parts. These parts are all put in a separate package. There is the `model` package that contains all the models. The `view` package contains all the views that correspond to the models in the `model` package. This package also contains the dialogs that are used by the view classes to interact with the user. There are some classes that don't belong to the `model` or the `view` packages. These classes are put in the global (or root) package. All these classes in these packages are described below. There is a class diagram included in this paper of the model and the view packages, to clarify this description.

4.1.1 The model package

- At the base of the model hierarchy is the `ModelBase` class. This class contains all the functionality needed to send events to the views to notify them of a change in the model.

- There are several classes that inherit from the class `ModelBase`: `ElementModel`, `ContainmentModel`, `GeneralizationModel`, `UsesRelationModel`, `AssociationModel`, `AssociationRoleModel` and `AggregateModel`.

- `ElementModel` represents the basic elements of the data perspective, i.e. packages and classes (with the classes `PackageModel` and `ClassModel` to represent them).

- A package may contain other packages and classes. This is expressed with the `ContainmentModel` class. This class stores the visibility of the element outside of the package and possibly another name (an alias), for the element.

- A class can inherit some of its attributes and operations from another class. This relationship between classes is represented by the `GeneralizationModel` class. A generalization can optionally be given a name, for this reason the class has a name attribute.

- A class can use another class in the implementation of its behavior. To show this in the data perspective, an uses relation is drawn. Such a relation is represented by the class `UsesRelationModel`. This class will store the names of the operations that are used.

- When one class is in fact a composition of other classes, an aggregate relation is a relation between classes exists. This aggregate relation is represented by the class `AggregateModel`.

- All the relations that don't fall into one of the above categories can be represented by the class `AssociationModel`. It represents a general relation between two or more classes. A name and an optional stereotype can be given to an association. Each class that is part of an association plays a certain role in the association. This role is described with the class `AssociationRoleModel`. It stores the name of the role of that class, the number of objects of that class that can participate in the association (multiplicity), whether the association is navigable in that direction and finally if there is an inherent ordering among the objects of that class that participate in the association.

- Classes can have members, i.e. operations and attributes. These members are represented by the model `MemberModel`. This model stores the name and visibility of the member and whether the member is part of an object or of the class itself (`static` in C++) .

- The class `AttributeModel` is a subclass of `MemberModel`. It stores the type of the member, an optional initial value and whether the value of the member is derived from other members.

- `OperationModel` is also a subclass of `MemberModel`. The things it stores are an optional return type, whether the operation is implemented or abstract and the parameters that the operations needs. Of these parameters the type and optional default value are stored.

- Although `StdModel`, `StateModel` and `TransitionModel` are classes mentioned in the class diagram, these classes where not implemented due to time limitations. They were left in the design to aid the future addition of state transition diagrams to the editor.

  Every class has an external STD (State Transition Diagram). This STD is represented with the `StdModel` class. The StdModel has states and transitions. The states are represented with the `StateModel` class. The states are connected to one another by transitions, represented with the `TransitionModel` class.

4.1.2 The view package

In this section the classes in the `view` package will be described. Most of the classes in this package are the view of the corresponding model in the `model` class. All the names of the views end with "View", the corresponding model has the same name, but the "View" suffix is replaced with "Model". For example, the class `ElementView` is the view of the model `ElementModel`.

- The class `Movable` contains the functionality that allows the user to move a component on the screen to a new location. The user can use the mouse to drag a component (that is a subclass of `Movable`) and this class will allow it to move.

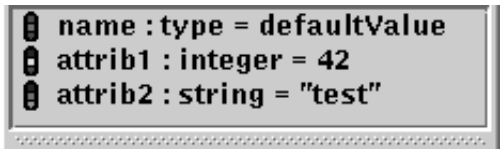
- `RelativeMovable` inherits from `Movable` and allows a component on the screen to be moved relative to another component. For example, A is a `Movable` and B is a `RelativeMovable` that moves relative to A. The user can move B around and the `RelativeMovable` will keep track of the relative offset to A. If the user moves A around, B will move as well to keep the offset the same.

- `BoundedMovable` inherits from `RelativeMovable`. It restricts the movement of B, relative to A, so that B is always on the contours of A.

- The class `ElementView` is used to show an `ElementModel` on the screen. It is the base class of the classes `ClassView` and `PackageView`.

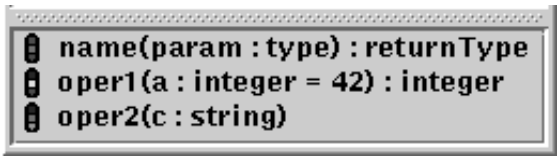
A ClassView that contains no attributes or operations.

- `ClassView` shows a `ClassModel` on the screen. At the top of the classview there are the stereotype and name labels and the visibility icon. The visibility icon uses the colors of a traffic light to show public visibility (green), protected visibility (yellow) and private visibility (red). Below the labels and icon there is a list of attributes (`AttributesView`). Below the attributes is a list of operations (`OperationsView`).



A part of a ClassView, the AttributesView, that shows three attributes.

- The `AttributesView` class is used to show a list of attributes. It presents the attributes in a vertical list and allows the user to scroll through them. Each attribute in the list of attributes is shown using the `AttributeView` class. It shows the visibility (public, protected or private) of the argument, the name, the type and the initial value of the attribute.



Another part of a ClassView, the OperationsView, that shows three operations.

- Just like there is an `AttributesView` class that shows the attributes of a class, there is an `OperationsView` class that shows the operations of a class. Each operation in the operation list is shown using the `OperationView` class. Besides the visibility and the name it also shows the parameters and the return type of the operation.

An empty PackageView.

- The objects of class `PackageView` are rectangles with the name and stereotype of the package at the left-upper corner. Inside the rectangle are the elements that are contained by the package.

- The class `PackageWindow` inherits from Java's `Window` class. Every instance of `PackageWindow` is a window on the screen. The window contains a menubar and a toolbar. Below the toolbar is an instance of `PackageView`

- At any given time the `PackageWindow` is in a certain mode. This can be the move mode that allows the user to move, resize or select elements or it can be the addClass, addPackage or addAssociation mode that allows the user to add a class, package or association to the `PackageView`. `Mode` is the class that stores the current mode the `PackageWindow` is in.

A Line that is connected to a ClassView.

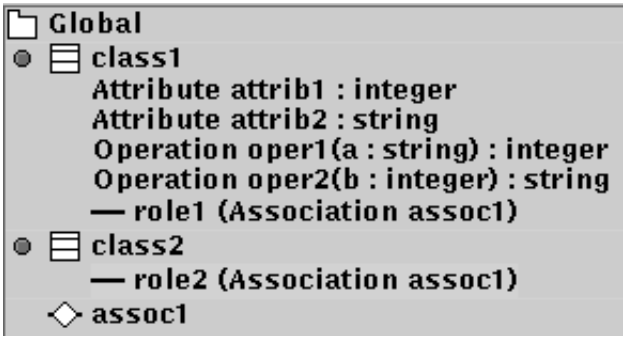
- Most of the relations are drawn using lines between `ClassView`s. The `Line` draws the lines. A line can consist of one or more straight lines. Each of these straight lines is drawn by a `LineSegment`.

AssociationView drawn with a diamond shape.

- The class `AssociationView` is used to show an association between classes. The class `AssociationView` has two ways to draw itself. One way is used when there is an association between exactly two classes. In that case the association is represented as a line between the classes. The other way is used when there are more than two classes. Then the `AssociationView` will draw a diamond with the name and the stereotype of the association inside the diamond. Each classes is connected to the diamond with a single line.

- Each class that is part of an association plays a certain role in that association. The name and multiplicity of that role is shown using the class `AssociationRoleView`.

- `GeneralizationView`, `UsesRelationView`, `AggregateView`: Although these classes are needed to create a full featured editor (and for that reason they are in the design of the editor) there was not enough time to implement these classes.

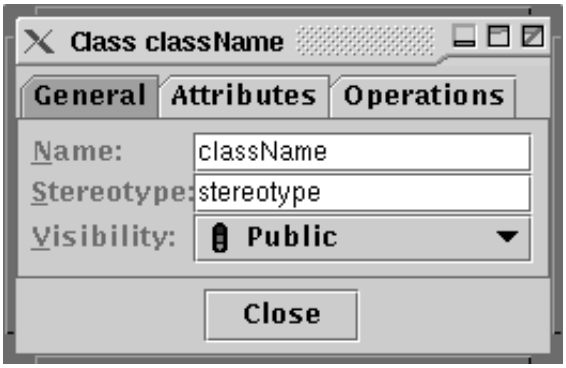


A ModelTree that shows two classes and a binary association between them. It also shows that the first class has two attributes and two operations.

- While writing the Socca Editor, it became clear that there are situations where one would like to have a `PackageView` that only partially shows the contents of a `PackageModel`. In other words, one might want to delete an element from the `PackageView` without deleting it from the `PackageModel`. In such a case, the user should be able to re-add the deleted elements.

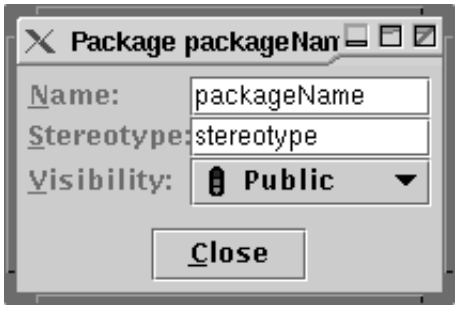
  To allow the user to add these deleted elements and to give the user a clear view of what is in the model, a tree was created that shows everything that is inside a `PackageModel`. This tree, `ModelTree`, shows all the elements in the `PackageModel` and everything that belongs to these elements.

  To the right is a screen-shot of a `ModelTree`. It is the tree of the package "Global". It contains two classes ("class1" and "class2") and an association between these classes ("assoc1"). Class "class1" contains two attributes and two operations.



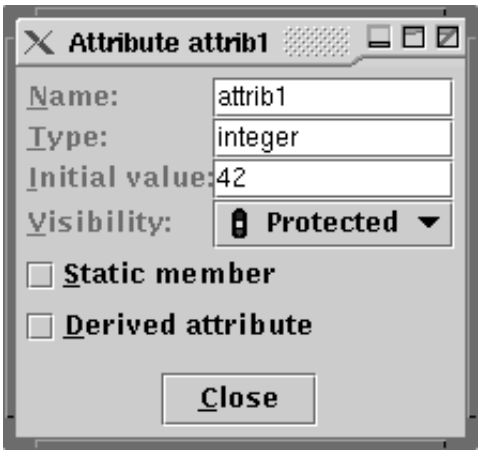
A dialog that allows class "className" to be edited.

- The class `ClassDialog` is a dialog that is used to edit the properties of a `ClassModel`. The name, stereotype and visibility of a classes can be edited and attributes and operations can be added, edited and removed.

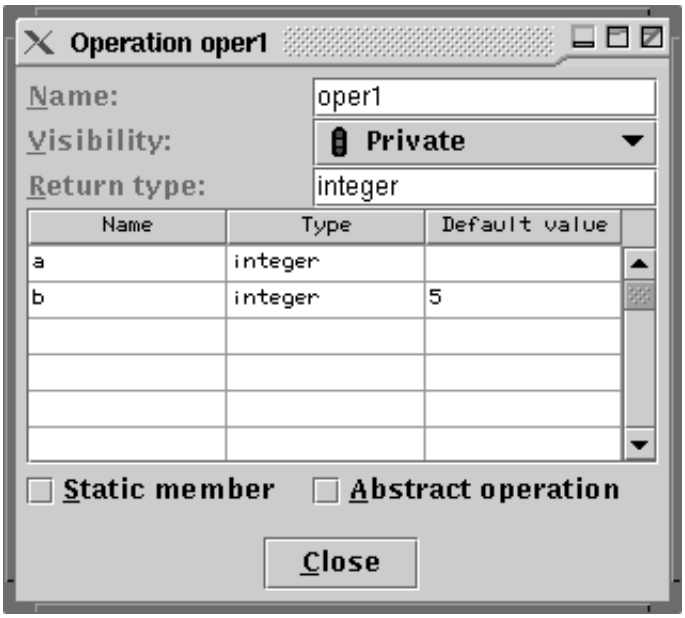


A Dialog allowing package "packageName" to be edited.

- Just like there is a `ClassDialog` to edit `ClassModel`s, there is a `PackageDialog` to edit `PackageModel`s. The name, stereotype and visibility of the package can be edited.



- The `AttributeDialog` class is used to edit the properties of an attribute. There is a text-field for the name, the data type and the initial value of the attribute. The drop-down box can be used to change the visibility of the attribute. Finally, there are two check boxes; if ‘Static member’ is checked it specifies that the attribute will part of the class and not of the object. ‘Derived attribute’ specifies that the value of the attribute will be derived from other attributes.



- Using the `OperationDialog` dialog one can change the name, visibility and return type of an operation. The table can be used to add, modify and remove the parameters of the operation. Each parameter has a name, data type and an optional default type. The 'Static member' check-box has the same function as the 'Static member' check-box of the `AttributeDialog` dialog. The 'Abstract operation' check-box specifies that the operation will not be implemented in the class it belongs to, but that it will be implemented in a derived class.

- The name and stereo type of an association can be changed with the `AssociationDialog` dialog.

5 Problems encountered

During the development of the editor numerous problems were encountered. Most of these were trivial and could easily be solved. Some of the problems however, required significantly more time and effort to overcome them. These problems are described below.

5.1 Event handling on Lines

The `LineSegment` class that is described above, is used to draw the lines that represent an association. There are two ways the lines can be drawn. One way is to draw directly on the `PackageView`. The disadvantage is that the `PackageView` is not aware of these lines and the lines will not be redrawn when an element inside the `PackageView` moves or resizes. Secondly, when the user wants to move a line to a different location, the line that was drawn is not automatically erased. The only way to remove the line on the old location is to tell the `PackageView` to redraw itself. This results in a lot of flickering.

The other way is to make every `LineSegment` a real `AWT Component`. The advantage of this approach is that AWT can then automatically handle the redraws and it can do this much more efficiently than the application itself can. Secondly lines can be moved around without flickering. There is, however, a disadvantage to using `AWT`'s `Component`. `AWT` keeps track of the component's location and size. The location is the x and y coordinate of the left-upper corner of the `Component` and the size is a width, height pair. These four values only allow for rectangular shapes. Indeed, `AWT` does not support `Component`s that are not rectangular.

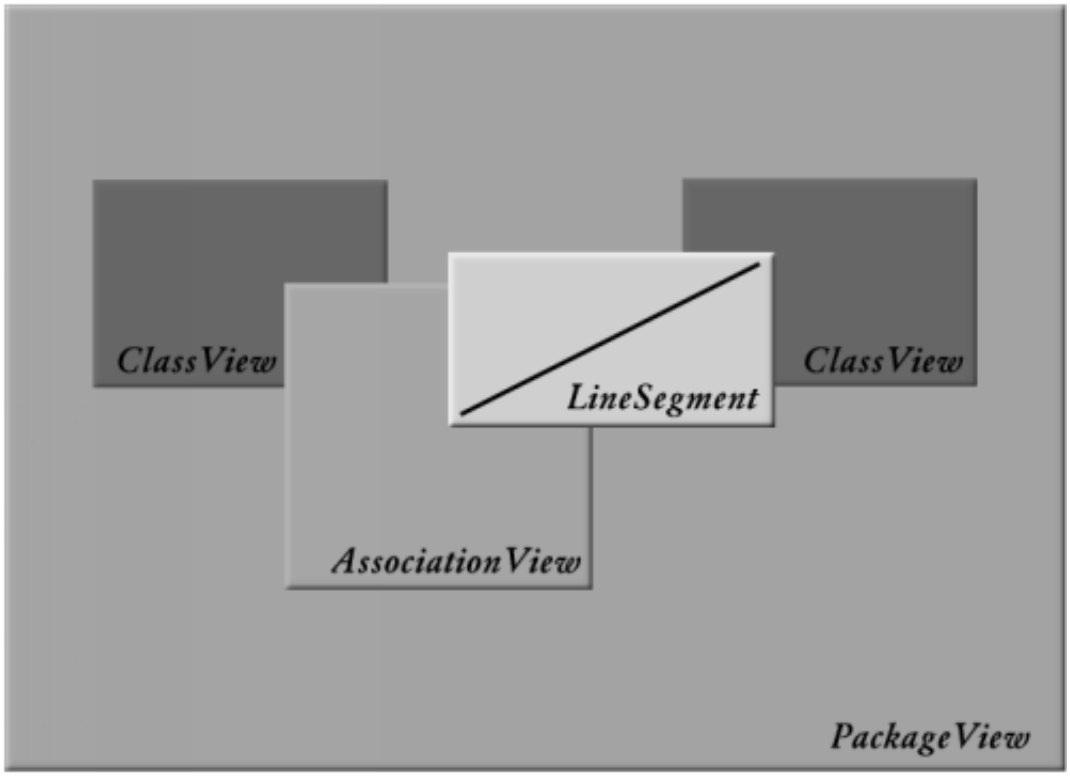
In version 1.1 of Java, transparency was introduced. This transparency allows the `Component` to leave a part of its rectangle undrawn. The `Component` behind it will then show through. This can be used to simulate non-rectangular shapes. `LineSegment` uses this transparency to only draw the line itself. The rest of its rectangle is left untouched and will show the components that lie below the `LineSegment`. This was fairly easy to get working correctly. The real problems was with event handling.

Java's event handling is currently quite complex. In Java 1.0 it was still pretty simple; when the user clicks a mouse button, an event is sent to the component the mouse is on. This event is sent to a method of the component. If the component decides to handle the event itself, it will do so and the method returns `true`. On the other hand, if the component doesn't handle the event itself, it will return `false`. At that point `AWT` will sent the event to the parent component which itself can decide whether to handle the event or not, etc.

This way of handling events has several disadvantages. The events are sent to a component regardless of whether it is interested in them or not. This is a real problem with mouse-move events, hundreds can be sent to the components per second when the user is moving the mouse quickly and maybe there isn't a single component interested in them. Performance noticeably suffers from the resulting number of method calls. A second disadvantage is that to react to an event, the programmer has to subclass the component. This leads to a explosion in the number of classes.

In Java 1.1 these problems were solved by introducing the concept of listeners. The programmer can register one or more listeners to a component. All the events are then sent to the listener(s). The advantage is that events are only sent to objects (the listeners) that are interested in them and listeners can be registered without deriving a new class from the component. This too is a concept that is easy enough to understand. What makes Java's event handling so complex is the desire to be compatible with the 1.0 way of event handling while at the same time maximize the performance gains of the 1.1 way.

Back to `LineSegment` and its rectangular shape. The problem is that `LineSegment` either receives events on its entire rectangle or not at all. What is required is that only events that are generated when the mouse pointer is close to the line are intercepted by the `LineSegment` and the events that generated on other parts of the rectangle should be passed on to the component that lies behind it.



In the above picture, a schematic view of a `PackageView` is shown. Inside the `PackageView` are two `ClassViews`, on top of one of them is an `AssociationView` and on top of the `AssociationView` a `LineSegment`. When the user clicks the mouse on the line inside the `LineSegment` the `LineSegment` class should handle the event. If the user clicks on the `LineSegment`, but not on the line, the event should be handled by the `AssociationView`, the right `ClassView` or the `PackageView`, depending on the location of the click.

The first thing that was tried to solve this problem was to use the 1.0 way of event handling. It seemed like a perfect fit; the `LineSegment` returns `true` when the mouse is on the line and it returns `false` when the mouse is on a different part of the rectangle. Unfortunately,

because the application was written using Swing and Swing's `JComponent` registers an event listener for its own use, the 1.0 compatibility code is disabled. The functions that were used for event handling in 1.0 are simply not called anymore once an event listener is registered (this is to maximize the performance gain of the new event handling code).

The only other option was to somehow adapt the 1.1 way of event handling to selectively pass the event to the `LineSegment` or to an underlying component. To achieve this, the AWT source code was studied extensively. It was discovered that AWT sends the event to the `LineSegment` if it has an event listener registered, `LineSegment` then sends the event to all the listeners. If `LineSegment` doesn't have any listeners registered it doesn't receive the event but the event is passed to an underlying component. The class that makes this decision is AWT's `Container`, it has the function `getMouseEventTarget(int x, int y)` that will find a component that has listeners registered and contains the given location. If this function could somehow be influenced to return the `LineSegment` when the mouse is on the line and the underlying component otherwise, all event handling would work as desired. Frustratingly enough, the `getMouseEventTarget` function is a private function and cannot be overridden in a subclass.

The next approach that was tried was to receive the event in `LineSegment` and then, depending on the location of the mouse, send the event to the listeners or explicitly to the parent of the component (`PackageView`). The problem with this approach is that the parent will, upon receiving the event, look if there is a child component that lies on the location of the mouse and has any listeners registered. If so it will forward the event to this component. In this case the `LineSegment` is the component that lies on top and has listeners registered, and thus the `LineSegment` will receive the event. `LineSegment` will try to sent it back to the parent which will forward it again to the `LineSegment`. This recursive calling of event handler functions will never stop.

Somehow the event should be tagged once the `LineSegment` has rejected it and sent it to the parent component. The parent component should then check the event for this tag and not forward it if it is indeed tagged. Since events are generated internally in `AWT` the event cannot be subclassed (`AWT` would still use the original class, not the subclass). So a way of tagging the event should be found in the original event class. The `AWT`'s `MouseEvent` class indeed has a way of tagging the event: `boolean consumed`. This flag is meant to be used by the listeners to tell the component that the event has already been processed and the component itself should not take any more action. Although this flag is present in the `MouseEvent` class, there aren't many components that use it for this purpose, they don't check this flag. This flag could thus be misused for the tagging purpose. The source code of this solution didn't look pretty and it isn't completely correct either. It is possible (like in the picture above) that there is a component between the `LineSegment` and the `PackageView`. Sending the event from the `LineSegment` to the `PackageView` is thus not always correct. Moreover, since the `consumed` flag of the `MouseEvent` is used in a way it was to meant to be used, it is not guaranteed to even work in future versions of Java. A better solution had to be found.

After studying the source code of AWT some more, it was discovered that the method getMouseEventTarget calls the method contains(int x, int y) of each child in the container until it finds a child that returns true. It will start at the top and continue calling the contains method until it finds a child that says it contains the point. The method contains is public and can thus be overridden in the MouseEvent class. The LineSegment class can return true if the x and y coordinates are on or near the line and false otherwise. If it returns false, the Container automatically goes on to the child that lies underneath the LineSegment and calls the contains method of that child. This is exactly what is needed. This approach turned out to work very elegantly and efficiently. After optimizing the code a bit, this is the solution that is used in the editor.

5.2 Text-fields in dialogs

Although the editor is currently not capable of supporting multiple users at the same time, it was designed with a multi-user environment in mind. When multiple users are working on the same project, it is important that each of them knows what the others are doing. For this reason, their views of the model have to be up to date and consistent at all times.
So when one user changes some aspect of the model or adds something new to it, all the users should see this change immediately. In the current implementation of the editor, the views are even updated while the user is typing in or changing a name or other string.

Most of the changes of the model are done using dialog boxes. These dialog boxes contain text fields that can be used to change names, stereotypes, multiplicity, etc. When the user types in one of these field, the field generates an event for each key press. The dialog box receives this event and changes the underlying model accordingly. The model, in turn, will generate an event to alert all the views of the change. Since the dialogs have registered themselves with the model, they too receive the events and will update the fields. The result is that all the views and dialogs are always up to date and consistent.

The internal implementation of JFC's text-fields make this process a little harder than it needs to be. These text-fields use the Document class to store their string. The Document class of JFC was meant to store and manipulate large text documents, but can also be (and is) used for the shorter strings of the text-fields.

The Document class has just two methods to change (part of) its contents: insertString and remove. insertString adds a string at a specified offset while remove can be used to remove a specified number of characters at a given offset.

The text-fields have only one method to change the string: setText. Internally the text-field calls Document's remove to remove the whole string and then insertString to insert the parameter of setText into the Document.

The problem with this is, that the Document emits an event on each change. So when the text-field calls the remove and then the insertString methods, two events are generated. One that clears the string and one that sets the string to its new value. The dialog box listens for document events and updates the model when one is received.

When two dialog boxes are visible on the screen and the user is typing in one of them, the `Document` of the active text-field generates events that are received by the dialog box. The dialog box updates the model and the model sends events to all the views, including the dialog boxes. The second dialog box also receives the event of the model and updates it's text-field. This text-field, unfortunately, generates two events; an event that makes the string empty and an event that sets the string to its new value. Each of these events is received by the second dialog box and the model is updated twice. The model sends out two events, etc. The editor will end up in a never ending cycle of events.

The first solution that was tried was to ignore the 'remove' events and only handle the 'insert' events. This indeed worked correctly when the user typed in a new name, every text field had the correct value. But when the user used the backspace key to erase a part of the name, the generated 'remove' events were ignored and the text fields were no longer consistent. A different solution was needed.

After some investigation it was discovered that the 'remove' events that were generated due to the `setText` method removed the entire string, while 'remove' events that were due to use of the backspace key removed only one character of the string. The current implementation of the editor ignores 'remove' events that will remove the entire string while processing the other events. This works correctly unless the user wants to delete the last character of the string, in that case the last character is the entire contents of the string and the event is ignored.

5.3 Inter package associations

Associations between `ClassView`s are shown using lines or a combination of lines and a diamond. These lines are drawn using the `LineSegment` class as is described above. If the `ClassView` or the diamond is moved by the user, the `LineSegment` has to update its size and/or location. When the `ClassView`s that play a role in the association are not all inside the same package, things become a little more complicated.

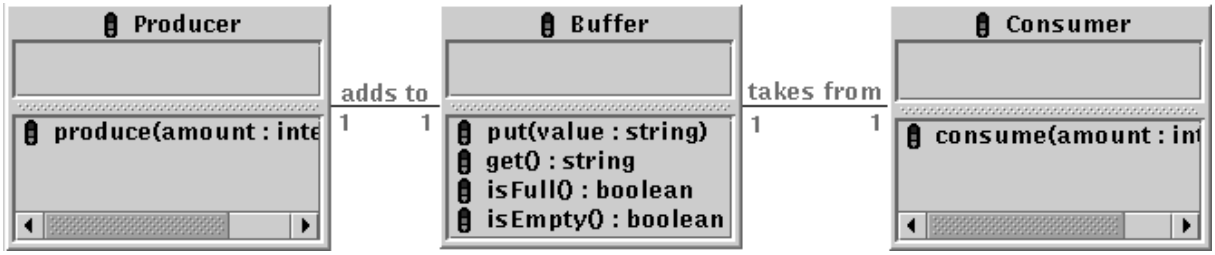
If the user moves the package that contains the `ClassView`, the `LineSegment` has to update its size and location as well. Also, when the `PackageView` is showing its scrollbars and the user uses them to scroll the `PackageView`, the position of the `ClassView` on the screen changes and the `LineSegment` needs to update its size and location. When the user scrolls the `PackageView` so that the `ClassView` is no longer visible, the `LineSegment` should disappear as well.

Above, it was explained that the components in Java have a specific z-ordering. Some components are drawn in front of others. The editor should prevent a `PackageView` to be draw on top of the `LineSegment` that goes to one of the `PackageView`'s children.

6 An example of the use of the editor

6.1 Introduction

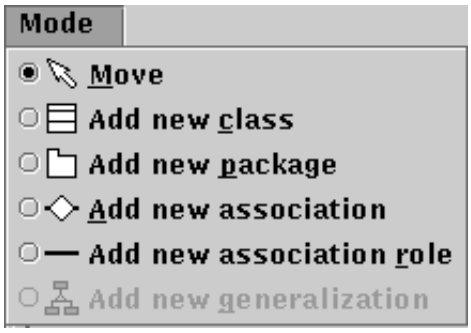
In this section a step by step walk-thru will be given of the editor. This is done by creating the producer-consumer pattern. The producer-consumer pattern consists of three classes: The `Buffer`, the `Producer` and the `Consumer`.
The `Buffer`: this class holds the items that are produced by the `Producer` until they are consumed by the `Consumer`. The `Producer` can add items to the buffer by calling the `put(value: string)` operation. The capacity of the buffer is limited so the `Producer` cannot add new items to the buffer if the operation `isFull()` : `boolean` returns `true`. The `Consumer` can take one item out of the buffer by calling the operation `get()` : `string` unless `isEmpty()` : `boolean` returns `true` in which case the buffer doesn't contain any items anymore.
The `Producer` can be told to produce more items by calling the operation `produce(amount: integer)`.
The `Consumer` consumes items when the `consume(amount: integer)` operation is called.



In the next part of this section, the producer-consumer pattern will be created. A fairly detailed description of the steps that are required are given so that the reader becomes familiar with the functionality of the editor.

6.2 Step by step walk-thru

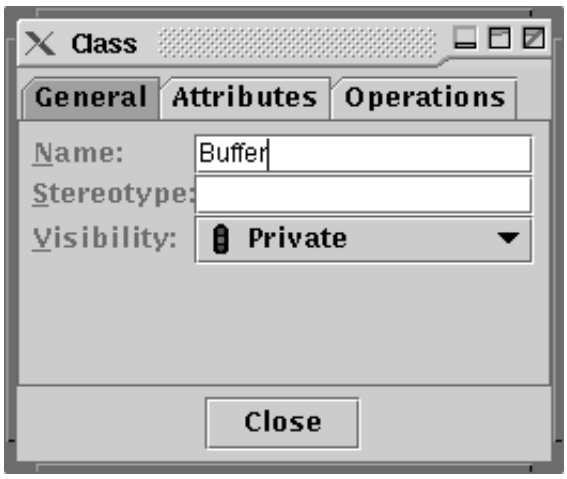
When the editor is started (At Leiden run `/home/most/run` on a Sun Ultra), it will open two windows. The model tree at the left of the screen and a view of the `'Global'` package at the center of the screen. This example will focus on the window in the center of the screen, the model tree will be ignored for now.



At the top of the package-view there is a menu bar with one menu item, the `Mode` menu. This menu allows the user to change the mode the window is in. Usually the window is in `Move` mode. In this mode the user can move and resize the contents of the package view and the user can select or deselect elements. The other modes allow the user to add new things to the package model (and thus the package view). There is a mode to add a new class to the package, a mode to add a new package inside the package, a mode to add a new association and a mode to add a new package role.

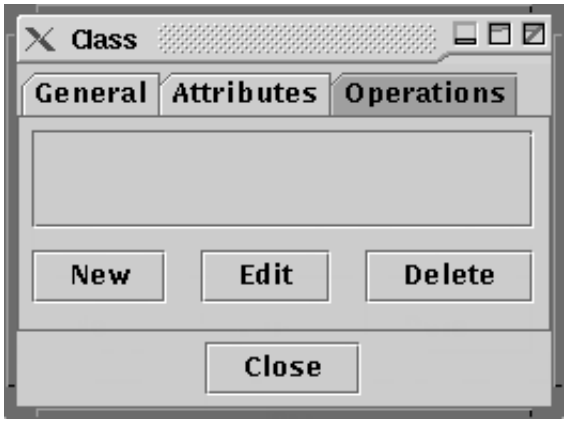
All the different modes that are in the `Mode` menu are also visible as icons on the toolbar just below the menu bar. Selecting a mode with the menu has the same effect as selecting one with an icon on the toolbar, so use the method that is more convenient to you.

The first thing to do is to add the `Buffer` class. For that, the window should be in the ‘Add new class’ mode. Either select the second menu item of the `Mode` menu or select the second icon of the toolbar. Move the mouse to the location where you want the `buffer` class to be created and press the left mouse button.

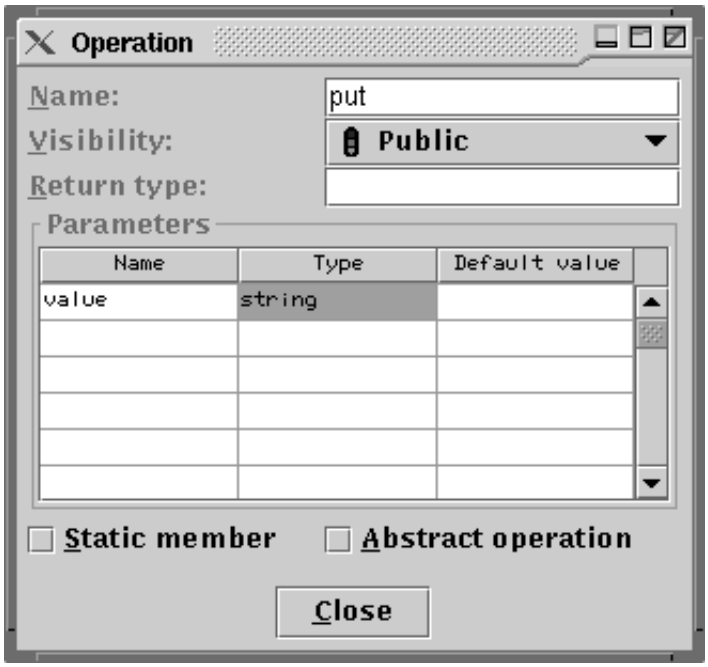


Within seconds a dialog should appear. This dialog shows the name, stereotype and visibility of the class. The newly created class doesn’t have a name yet, so the name text field is empty. To give the class a name, click the left mouse button on the white text field next to the

‘Name:’ label. Type the name of the class, for example ‘Buffer’. The buffer class has no stereotype, so the stereotype text field can be left empty. Below the stereotype, there is the visibility combo box. By default, the visibility is Private. To change this to Public, click on the word Private. A drop down box will appear with the three choices Public, Protected and Private. Click the left mouse button on the word Public. The drop down box will disappear and the visibility will now be Public.



At the top of the dialog there are the tabs General, Attributes and Operations. The General tab is the one that is selected. By clicking with the left mouse button on the Operations tab, the name, stereotype and visibility fields will disappear, the operations of the class will be shown. Because the class was just created, it doesn’t have operations yet. To create a new operation, click on the New button.



A new dialog will appear. This is the operations dialog. At the top there is the name text field. Click on the white text field next to the ‘Name:’ label to change the name. Enter the name of the operation, e.g. ‘put’.

Below the name text field there is the visibility combo box. It works just like the visibility field of the class. You can use it to change the visibility of the operation to public.
Below the visibility field is the return type field. If the operation doesn't return a value, you can leave it empty. If it does return a value, the type of the value can be specified here.
Below the return type are the parameters. The information of the parameters is arranged in a grid. Each row represents one parameter. Each rows contains the name, type and default value of one parameter. To change a value, click on the cell and type in the new value.
To give the 'put' operation one parameter, 'value', do the following: Click with the left mouse button on the top most field of the 'Name' column. Then enter the name of the parameter: 'value'. Click the mouse on the field right next to it, in the 'Type' column. Enter the type of the parameter, 'string'. To the right of the type field there is the field for the default value. You may leave this field empty if no default value is needed. Otherwise enter the default value in this field. Finally press enter to confirm your changes and click with the mouse on the 'Close' button to dismiss the dialog. The class dialog will become visible again. By pressing the 'New' button again the `get`, `isFull` and `isEmpty` operations can be added. Once this is done the 'Close' button can be used to close this dialog as well.

Now that the `Buffer` class is created, it is time to create the `Producer` class. Set the mode of the window to 'Add new class' and click the left mouse button where the `Producer` class should appear. Giving the class it's name ('Producer') and changing the visibility ('Public') works the same as above with the `Buffer`.

Once the `Producer` is created create the `Consumer` in a similar fashion. Next are the associations between the `Producer` and the `Buffer` and between the `Consumer` and the `Buffer`. Normally associations are created by switching to the 'Add new association' mode. In this mode a new association, shown as a diamond, can be created. After the diamond is visible the different roles that the classes play in the association can be shown with the 'Add new association role' mode. However, in `UML` and thus in `Socca` binary associations aren't usually shown with a diamond but simply with a straight line. To do this in the editor, don't use the 'Add new association' mode (as that would create the diamond), but switch directly to the 'Add new association role' mode. Once in this mode, press the left mouse button on the `Producer` class and drag the mouse (while holding the mouse button down) to the `Buffer` class. Once the mouse is on the `Buffer` class, release the left mouse button. A line will appear, this is the binary association. Click on it with the left mouse button to raise and select it. At the end points of the line there are two yellow circles. By dragging these yellow circles around, the line can be moved.

The classes can be moved around by pressing the left mouse button on their name, drag it to the new location and release the mouse button. The class dialog can be shown by double clicking on the name of the class or selecting 'Edit...' from the pop-up menu. The pop-up menu is shown by pressing (and holding) the right mouse button on the class. Classes can be deleted by selecting 'Delete' from the pop-up menu. There are two other items in the pop-up menu: 'Suppress' and 'Unsuppress'. Suppress will hide the contents of the class. This can be used when the contents of the class is not important in that window. E.g. when one wants to give a general overview of a large program, most of the details can be left out by suppressing the classes. The classes will still be visible, but their contents will be hidden, thereby simplifying the picture. When you want access to the contents, select the 'Unsuppress' item

from the pop-up menu.

7 Advantages and disadvantages of Java

Right from the start of this project using Java as the programming language was a requirement. This choice was not arbitrary, there are some very compelling reasons to choose Java. Some of these advantages are discussed here. Some of the disadvantages of Java will be highlighted as well.

7.1 Advantages of Java

- Java is platform independent, a program can be used on every operating system that supports Java. When a Java program is compiled, the output of the compiler is not assembly like the output of a normal compiler but it is a platform independent byte code. This byte code can be interpreted by the Java Virtual Machine (JVM). Every operating system that has a JVM can be used to execute the program. The program doesn't even need to be recompiled to work on a different platform. The result of this byte code is that processor specific details and operating specific details are dealt with on the target computer. The programmer will be able to create programs that run on every popular platform without worrying about machine specific details.

- Java is very object-oriented. This object oriented programming will lead to programs that are easier to understand, extent and maintain.

- Java has no pointers. The number one cause of bugs in programs written in C or C++ are incorrect use of pointers and incorrect memory management. Java uses references instead of pointers. A reference is similar to pointers, except that the programmer cannot use it as if it were an integer. This restriction has the effect that the language has more control over the things a reference can point to. The Java compiler makes sure that a reference only points to an object of the correct type or to `null`. Secondly, because the references cannot start to point to a different object without the JVM knowing about it, the JVM can deallocate a part of memory when the last reference to it disappears. This garbage collection greatly simplifies programming and prevents a large number of bugs.

- Several advanced programming concept are included in the language. Multi-threading, synchronization and exception handling and remote method invocation (RMI) are native parts of the language and not add-on libraries as in other languages. This means that the programmer can use these concepts easily and create programs that are more robust and react faster than variants of the same program written in a different language. These concepts are not used in the current state of the editor but will be essential once multi-user capabilities are introduced.

- Java can be used on the World Wide Web. Most popular browsers can run Java programs, called *applets*, inside a page. People will be able to use the editor everywhere on the world. They don't need to explicitly download or install the program. In practice, however, it will take some time before this is a usable option. Most of the browsers only support Java 1.0 or a subset of Java 1.1 . Since the editor uses Java 1.1 extensively it will

take several months, if not longer, before the editor can be used inside a browser.

- Java is so over-hyped that everybody is impressed when they learn that Java was used.

7.2 Disadvantages

- The object oriented nature of Java means that large portions of the `Abstract Window Toolkit` (AWT) and of the `Java Foundation Classes` (JFC) are private. If some functionality of these toolkits needs to be overridden by the programmer, but the designer of the toolkit had not anticipated this need, this encapsulation can complicate the live of the programmer. This frustrating effect has hindered the progress of the editor at several occasions. One of them is the event handling as discussed before.

- Java is a very young language. Support in operating systems and browsers has only been added recently. Support for the less popular platforms may not even exist yet. Bug fixes in the language, the libraries and the compiler are made on a regular basis.

- Java has been a popular language from the start. Backwards compatibility is an important issue that slows down progress. Some of the less fortunate choices in the earlier versions of Java (event handling and native peers) were corrected in Java 1.1 and Java 1.2 but their effect is still noticeable. If the growth of Java had been more gradual, the language could have been simpler and more elegant.

8 Future Work

The current implementation of the editor is limited in its functionality. Future versions might extend this functionality in certain areas. A few of these possible extensions are discussed here.

- More of UML's class diagram. The editor supports only a subset of the class diagrams that UML allows. To equal UML the editor must be extended to allow more kinds of relations and the ability to add notes to parts of the model. Generalizations, aggregations and dependencies should be added and at the same time Socca's uses-relations might be added.

- More of UML. Besides the class diagram, UML support many more diagrams. Some of these, if compatible with Socca, might be included. Beside the class diagrams UML also has use-case diagrams, sequence diagrams, collaboration diagrams, state diagrams, activity diagrams and implementation diagrams. The state diagrams and activity diagrams don't add much if Socca's internal and external STDs are supported. The other diagrams might add some useful functionality.

- Socca's internal and external STDs. Socca has three types of diagrams, the class diagram, the internal STDs and the external STDs. To complete the Socca support, the editor has to support least these internal and external STDs.

- Automatic generation of code. Most of the modern CASE tools are able to automatically generate source code that the programmer can use as a basis of the program that is to be written. It would be nice if the editor is extended to generate source code for several popular programming languages.

- Full round-trip engineering. It happens quite often that while the programmers are implementing the program, a change has to be made to the design. If the editor supports full round-trip engineering, the code that has been generated by the editor and subsequently modified by the programmers can be read back in, changes can be made and the code can be written out again, without losing the implementation that was added by the programmers.

- Socca process simulation/enacting. Socca's STDs can be used in a simulation of the program. Even before the program is fully implemented, certain aspects of the program can be shown using a simulation. Especially the interaction of the different parts of the program can be shown. The editor might show the STDs of the program and show which state it is currently in and which transition will be used in the next step.

- Multi-user. In most enterprises it is rare that a single person designs and implements a program. It is much more common that there is a small team of developers all working on the program. The editor should allow them all to work simultaneously on the design of the program.

- Version control system. Several studies have been made of multi-user development environments and each time it was concluded that, if the group of developers is not very small, it is more efficient to have the developers work on a model that is known to be correct, rather than with the most up to date version. The model is incrementally created and each time a developer finishes a step in the development he/she 'checks in' his/her changes. Once the changes are checked in, they become available to the other developers. In other words, the other developers don't see the inconsistent state the model is in while the developer is making his/her changes but only see the consistent end-result. The editor must use a version control system to support this.

- Repository of patterns. Certain problems show up in different programs over and over again. It would be useful to have a collection of these problems and their solutions so that the developers don't have to reinvent the wheel. Gamma et al. wrote the book *Design Patterns*[4] for this purpose, but it would be even more useful to be able to cut-and-paste these patterns into a design and to easily integrate them there. Over time this repository can be updated and extended to increase its usefulness.

- Above it was mentioned that it is desirable to have an editor that is able to generate code. The Socca modeling techniques can describe the program in such detail that most of the program can be automatically generated from the model. There are, however, a few issues that have to be addressed before efficient programs can be generated.

  The Socca architecture uses STDs to describe the behavior of a class. Each operation has one or more STDs and there are one or more STDs for the whole class. The STDs are assumed to be executed on state-machines that each run on its own CPU. This theoretical model is, although simple and powerful, very impractical. Current computers have one or in rare cases a few CPUs. There can be hundreds or even thousands of STDs when a non-trivial program is modeled. The state-machines for these STDs cannot each have their own CPU. Running each state-machine in a thread of a multi-threaded program isn't practical either since the overhead of thread switching will make the program inefficient.

  More research is needed to find a general method that will map the STDs of Socca onto a number of threads in such a way that most or all of the parallelism of the model is preserved while keeping the number of threads low so that the generated program runs efficiently. If the number of threads is too high, the user may be able to manually make a more efficient program and the code-generation becomes a less attractive feature of the editor.

  One solution that was suggested is to extend Socca and add a notation to group parts of the model in a single thread. The user can use this notation to annotate the model and explicitly state that within a group of STDs, parallelism is not required. Since these STDs are then known not to run at the same time, the code generation portion of the editor can map the state-machines for these STDs to a single thread. An example of such a mapping is to interleave the instructions of the state-machines. This results in sequential execution that closely resembles parallel execution. This is exactly what an operation system would do when multiple threads are run on a single CPU, except that

the interleaving (scheduling) is not done at runtime but at code-generation time.

One intuitive grouping of STDs would be to group all STDs of a class together; in other words, within objects of such a class there is one thread to run the state-machines. The state machines would not run in parallel but their execution would be interleaved.

9 Conclusion

In this thesis the development of a prototype editor for Socca has been described. The general architecture of the prototype was discussed and the model-view architecture was highlighted. The classes of the editor were each shortly described. Some of the more significant problems that were encountered during the development of the prototype were discussed. After that, a step by step walk-thru was given so that the reader could become familiar with the functionality of the editor. The advantages and disadvantages of the Java programming languages were discussed to motivate the choice to use Java. And finally, a number of possible directions for future work were presented.

10 References

1. Gregor Engels and Luuk P.J. Groenewegen. *SOCCA: Specifications of Coordinated and Cooperative Activities*. In A.Finkelstein, J.Kramer, and B.A. Nuseibeh, editors, Software Process Modelling and Technology, pages 71-102. Research Studies Press, Taunton, 1994.

2. Tineke de Bunje, Gregor Engels, Luuk P.J. Groenewegen, Aart Matsinger and Mark Rijnbeek. *Industrial Maintenance Modelled in SOCCA: an Experience Report*. Proceedings of the Fourth International Conference on the Software Process, improvement and practice, page 13-26. Brington, UK, 1996. IEEE Computer Society Press, Los Alamitos, California.

3. Glen E. Krasner and Stephen T. Pope. *A cookbook for using the model view controller user interface paradigm in Smalltalk-80*. Journal of Object Oriented Programming, 1(3):26-49, August/September 1988.

4. Gamma, Helm, Johnson and Vlissides. *Design Patterns*. Addison-Wesley, Reading, MA, 1995.

11 Appendix A: The documentation of the classes

11.1 Class SoccaEditor

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Panel
                           |
                           +----java.applet.Applet
                                   |
                                   +----com.sun.java.swing.JApplet
                                           |
                                           +----SoccaEditor
```

---

public class **SoccaEditor**
extends JApplet

The class SoccaEditor is the main class. This class contains the "main" method that is called when the Socca editor is started. The "init" method is called when the editor is started as an applet inside a browser.

Constructor Index

- **SoccaEditor**()

Method Index

- **init**()
  This method is called when the editor is run as an applet.
- **main**(String[])
  This method is called when the editor is run as an application.

Constructors

- **SoccaEditor**

```
public SoccaEditor()
```

- **main**

```
public static void main(String args[])
```

This method is called when the editor is run as an application. It creates and shows a MainWindow

See Also:
MainWindow

- **init**

```
public void init()
```

This method is called when the editor is run as an applet. It, too, creates and shows a MainWindow

Overrides:
init in class Applet
See Also:
MainWindow

11.2 Class ModelChangeEvent

```
java.lang.Object
   |
   +----java.util.EventObject
           |
           +----ModelChangeEvent
```

---

public class **ModelChangeEvent**
extends EventObject

When something in a model changes, the model sends out a ModelChangeEvent. This event is send to all the views that have registered themselves as event listeners.

Variable Index

- **ABSTRACT**
  A function is now or is no longer abstract
- **AGGREGATE**
  A attribute is now or is no longer an aggregate
- **ASSOCIATIONS**
  An association was added to or removed from a class
- **ATTRIBUTES**
  An attribute was added to or removed from a class
- **CLASSMEMBER**
  An attribute or operation is now or is no longer a classmember
- **DERIVED**
  A attribute is now or is no longer derived from other attributes
- **ELEMENTS**
  An element (= a package or a class) was added to or removed from a package
- **GENERALIZATIONS**
  An generalization was added to or removed from a class
- **INITIAL_VALUE**
  The initial value of an attribute has changed
- **MULTIPLICITY**
  The multiplicity of an association has changed
- **NAME**
  The name of a class/package/attribute/etc has changed
- **NAVIGABLE**
  An association is now or is no longer navigable
- **NOTES**
  There was a note added to or removed from a model
- **OPERATIONS**
  An operation was added to or removed from a class

- **OWNER**
  The owner of a element or member has changed
- **PARAMETERS**
  The parameters of an operation have changed
- **RETURNTYPE**
  The return type of an operation has changed
- **ROLES**
  A role was added to or removed from an association
- **STEREOTYPE**
  The stereotype of a class/package/association/etc has changed
- **TYPE**
  The type of an attribute or parameter has changed
- **type**
  The type of the event.
- **USES_RELATIONS**
  A uses-relation was added to or removed from a class
- **value**
  The value of the changed variable at the moment the event was fired
- **VALUE_ADDED**
  A value was added to the model
- **VALUE_CHANGED**
  A value in the model has changed
- **VALUE_REMOVED**
  A value was removed from the model
- **VISIBILITY**
  The visibility of an element or member has changed
- **what**
  What has been changed, this one of the 22 constants above.

Constructor Index

- **ModelChangeEvent**(Object, int, int, Object)
  Create a new event.

Method Index

- **getType**()
  Return the type of the event; VALUE_CHANGED, VALUE_ADDED or VALUE_REMOVED
- **getValue**()
  Return the value of the changed variable at the moment the event was fired
- **getWhat**()
  Return what has been changed.

Variables

- **VALUE_CHANGED**

```
public static final int VALUE_CHANGED
```

A value in the model has changed

- **VALUE_ADDED**

```
public static final int VALUE_ADDED
```

A value was added to the model

- **VALUE_REMOVED**

```
public static final int VALUE_REMOVED
```

A value was removed from the model

- **ABSTRACT**

```
public static final int ABSTRACT
```

A function is now or is no longer abstract

- **AGGREGATE**

```
public static final int AGGREGATE
```

A attribute is now or is no longer an aggregate

- **ASSOCIATIONS**

```
public static final int ASSOCIATIONS
```

An association was added to or removed from a class

- **ATTRIBUTES**

```
public static final int ATTRIBUTES
```

An attribute was added to or removed from a class

- **CLASSMEMBER**

```
public static final int CLASSMEMBER
```

An attribute or operation is now or is no longer a classmember

- **DERIVED**

```
public static final int DERIVED
```

A attribute is now or is no longer derived from other attributes

- **ELEMENTS**

```
public static final int ELEMENTS
```

An element (= a package or a class) was added to or removed from a package

- **GENERALIZATIONS**

```
public static final int GENERALIZATIONS
```

An generalization was added to or removed from a class

- **INITIAL_VALUE**

```
public static final int INITIAL_VALUE
```

The initial value of an attribute has changed

- **MULTIPLICITY**

```
public static final int MULTIPLICITY
```

The multiplicity of an association has changed

- **NAME**

```
public static final int NAME
```

The name of a class/package/attribute/etc has changed

- **NAVIGABLE**

```
public static final int NAVIGABLE
```

An association is now or is no longer navigable

- **NOTES**

```
public static final int NOTES
```

There was a note added to or removed from a model

- **OPERATIONS**

```
public static final int OPERATIONS
```

An operation was added to or removed from a class

- **OWNER**

```
public static final int OWNER
```

The owner of a element or member has changed

- **PARAMETERS**

```
public static final int PARAMETERS
```

The parameters of an operation have changed

- **ROLES**

```
public static final int ROLES
```

A role was added to or removed from an association

- **RETURNTYPE**

```
public static final int RETURNTYPE
```

The return type of an operation has changed

- **STEREOTYPE**

```
public static final int STEREOTYPE
```

The stereotype of a class/package/association/etc has changed

- **TYPE**

```
public static final int TYPE
```

The type of an attribute or parameter has changed

- **USES_RELATIONS**

```
public static final int USES_RELATIONS
```

A uses-relation was added to or removed from a class

- **VISIBILITY**

```
public static final int VISIBILITY
```

The visibility of an element or member has changed

- **type**

```
private int type
```

The type of the event. This is one of VALUE_CHANGED, VALUE_ADDED and VALUE_REMOVED.

See Also:
getType

what

```
private int what
```

What has been changed, this one of the 22 constants above.

See Also:
getWhat

value

```
private Object value
```

The value of the changed variable at the moment the event was fired

See Also:
getValue

Constructors

ModelChangeEvent

```
public ModelChangeEvent(Object source,
                        int type,
                        int what,
                        Object value)
```

Create a new event.

Parameters:
source - the source of the event.
type - the type of the event; VALUE_CHANGED, VALUE_ADDED or VALUE_REMOVED.
what - what has been changed
value - the value of the changed variable at the moment the event was fired

Methods

getType

```
public int getType()
```

Return the type of the event; VALUE_CHANGED, VALUE_ADDED or VALUE_REMOVED

- **getWhat**

```
public int getWhat()
```

Return what has been changed.

- **getValue**

```
public Object getValue()
```

Return the value of the changed variable at the moment the event was fired

11.3 Interface ModelChangeListener

public interface **ModelChangeListener**
extends EventListener

All views that are interested in receiving ModelChangeEvents should implement this interface and register themself with the model.

Method Index

- **valueAdded**(ModelChangeEvent)
  This method is called when a value is added to a model
- **valueChanged**(ModelChangeEvent)
  This method is called when a value in the model has changed
- **valueRemoved**(ModelChangeEvent)
  This method is called when a value is removed from a model

Methods

valueAdded

```
public abstract void valueAdded(ModelChangeEvent e)
```

This method is called when a value is added to a model

valueRemoved

```
public abstract void valueRemoved(ModelChangeEvent e)
```

This method is called when a value is removed from a model

valueChanged

```
public abstract void valueChanged(ModelChangeEvent e)
```

This method is called when a value in the model has changed

11.4 Class Visibility

```
java.lang.Object
   |
   +----Visibility
```

---

public class **Visibility**
extends Object
implements Serializable

This class represents the visibility of classes, packages, attributes and operations. There are three types of visibility: public, protected and private. These visibility types correspond to the visibility that one might be familiar with from OO languages like C++ or Java. When the visibility of a class/package/attribute/operation is shown or the screen a little trafic light is shown. The colors red, yellow and green correspond to private, protected and public respectively.

Variable Index

- **icon**
- **label**
- **PRIVATE**
  Private visibility nobody but the class itself can access this element/member
- **PROTECTED**
  Protected visibility, only the the subclasses can access it
- **PUBLIC**
  Public visibility, everybody can access this element/member
- **text**

Constructor Index

- **Visibility**(String, Icon)
  This constructor is private since there are only three visibility types and they are already created as static members of this class, no other visibility types can be created.

Method Index

- **getRenderer**()
  This method returns a ListCellRenderer that can be used to render listboxes in a dialog box.

- **toIcon**()
  Convert the visibility to a trafic light icon.
- **toString**()
  Convert the visibility one of the strings "Public", "Protected" or "Private"

Variables

- **PUBLIC**

```
public static final Visibility PUBLIC
```

  Public visibility, everybody can access this element/member

- **PROTECTED**

```
public static final Visibility PROTECTED
```

  Protected visibility, only the the subclasses can access it

- **PRIVATE**

```
public static final Visibility PRIVATE
```

  Private visibility nobody but the class itself can access this element/member

- **text**

```
private String text
```

- **icon**

```
private Icon icon
```

- **label**

```
private JLabel label
```

Constructors

- **Visibility**

```
private Visibility(String text,
                   Icon icon)
```

  This constructor is private since there are only three visibility types and they are already created as static members of this class, no other visibility types can be created.

Methods

- **toIcon**

```
public Icon toIcon()
```

Convert the visibility to a trafic light icon.

- **toString**

```
public String toString()
```

Convert the visibility one of the strings "Public", "Protected" or "Private"

Overrides:
toString in class Object

- **getRenderer**

```
public static ListCellRenderer getRenderer()
```

This method returns a ListCellRenderer that can be used to render listboxes in a dialog box.

11.5 Class model.AssociationModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.AssociationModel
```

---

public class **AssociationModel**
extends ModelBase

The class AssociationModel represents an association between classes. Each of these classes play a certain role in the association. These roles are represented with the class AssociationRoleModel.

See Also:
AssociationRoleModel

- **name**
  The name of the association.
- **owner**
  The owner of this association.
- **roles**
  The AssociationRoleModels that are part of this association

- **stereotype**
  An optional stereotype of the association

Constructor Index

- **AssociationModel**()
  The constructor of AssociationModel sets the name and sterotype to an empty string and the owner to null.

Method Index

- **addRole**(AssociationRoleModel)
  Add a new AssocationRoleModel to the association.
- **getName**()
  Return the name of the association.
- **getOwner**()
  Return the PackageModel that owns this association or null if there is currently no owner.
- **getRole**(int)
  Get one of the AssocationRoleModels.
- **getRoleCount**()
  Return the number of AssocationRoleModels.
- **getStereotype**()
  Return the stereotype of the association or null if there is none.
- **removeRole**(AssociationRoleModel)
  Remove an AssocationRoleModel from the association.
- **setName**(String)
  Change the name of the association.
- **setOwner**(PackageModel)
  Set the owner of this association.
- **setStereotype**(String)
  Set the stereotype.
- **toString**()
  Convert this association to a string.

Variables

- **name**

```
private String name
```

  The name of the association.

- **stereotype**

```
private String stereotype
```

An optional stereotype of the association

- **owner**

```
private PackageModel owner
```

The owner of this association.

- **roles**

```
private Vector roles
```

The AssociationRoleModels that are part of this association

Constructors

- **AssociationModel**

```
public AssociationModel()
```

The constructor of AssociationModel sets the name and sterotype to an empty string and the owner to null.

Methods

- **getName**

```
public synchronized String getName()
```

Return the name of the association.

- **setName**

```
public synchronized void setName(String name)
```

Change the name of the association. This method will send out an event to all the views to allow them to update themselves.

Parameters:

name - the new name

- **getStereotype**

```
public synchronized String getStereotype()
```

Return the stereotype of the association or null if there is none.

- **setStereotype**

```
public synchronized void setStereotype(String stereotype)
```

Set the stereotype. This method will send out an event to all the views to allow them to update themselves.

Parameters:
stereotype - the new stereotype, use null or "" if there is no stereotype

- **getRoleCount**

```
public synchronized int getRoleCount()
```

Return the number of AssocationRoleModels.

- **getRole**

```
public synchronized AssociationRoleModel getRole(int index)
```

Get one of the AssocationRoleModels.

Parameters:
index - the number of the role. This index should be between 0 and getRoleCount()

- **addRole**

```
public synchronized void addRole(AssociationRoleModel a)
```

Add a new AssocationRoleModel to the association. An role model may only added once, adding it again will have no effect. This method will send out an event to all the views to allow them to update themselves.

Parameters:
a - the new role-model that will be added.

- **removeRole**

```
public synchronized void removeRole(AssociationRoleModel a)
```

Remove an AssocationRoleModel from the association. If the role model is not part of the association calling this method will have no effect.

Parameters:
a - the AssociationRoleModel that will be removed

- **getOwner**

```
public synchronized PackageModel getOwner()
```

Return the PackageModel that owns this association or null if there is currently no owner.

setOwner

```
public synchronized void setOwner(PackageModel owner)
```

Set the owner of this association.

toString

```
public synchronized String toString()
```

Convert this association to a string. This will return the name of the association or if there is no name the string "<unnamed>"

Overrides:

toString in class Object

11.6 Class model.AssociationRoleModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.AssociationRoleModel
```

---

public class **AssociationRoleModel**
extends ModelBase

The class AssociationModel represents an association between classes. Each of these classes play a certain role in the association. These roles are represented with the class AssociationRoleModel.

See Also:
AssociationModel

Variable Index

- **AGGREGATE**
- **aggregate**
  The association can be a normal association, an aggregate or a composite; This variable specifies which it is.
- **association**
  The association this role is a part of
- **COMPOSITE**
- **klass**
  The class that plays a role in the association
- **multiplicity**
  The multiplicity of the role
- **name**
  The name of the role
- **navigable**
  Is this association navigable in this direction
- **NORMAL**
- **ordered**
  Are the objects that play this role ordered or is there no inherent ordering

Constructor Index

- **AssociationRoleModel**(ClassModel)

Method Index

- **getAggregate**()
  Return whether the association is a NORMAL association, an AGGREGATE or a COMPOSITE.
- **getAssociation**()
  Return the AssociationModel this role belongs to.
- **getKlass**()
  Return the class that plays this role in the association.
- **getMultiplicity**()
  Return the multiplicity of the role.
- **getName**()
  Return the name of the role.
- **isNavigable**()
  Return whether this role is navigable in this direction
- **isOrdered**()
  Return whether the objects that play this role are ordered or not.
- **setAggregate**(int)
  Change whether the association is a normal association, an aggregate or a composite.
- **setAssociation**(AssociationModel)
  Change the association of which this role is a part.
- **setMultiplicity**(String)
  Change the multiplicity of the role.
- **setName**(String)
  Change the name of the role.
- **setNavigable**(boolean)
  Set whether this role is navigable in this direction
- **setOrdered**(boolean)
  Set whether the objects that play this role are ordered or not.
- **toString**()
  Convert this role to a string.

Variables

- **NORMAL**

```
public static final int NORMAL
```

- **AGGREGATE**

```
public static final int AGGREGATE
```

- **COMPOSITE**

```
public static final int COMPOSITE
```

- **klass**

```
private ClassModel klass
```

  The class that plays a role in the association

- **association**

```
private AssociationModel association
```

  The association this role is a part of

- **name**

```
private String name
```

  The name of the role

- **multiplicity**

```
private String multiplicity
```

  The multiplicity of the role

- **aggregate**

```
private int aggregate
```

  The association can be a normal association, an aggregate or a composite; This variable specifies which it is.

- **navigable**

```
private boolean navigable
```

  Is this association navigable in this direction

- **ordered**

```
private boolean ordered
```

  Are the objects that play this role ordered or is there no inherent ordering

Constructors

- **AssociationRoleModel**

```
public AssociationRoleModel(ClassModel klass)
```

Methods

- **getAssociation**

```
public synchronized AssociationModel getAssociation()
```

Return the AssociationModel this role belongs to.

- **setAssociation**

```
synchronized void setAssociation(AssociationModel association)
```

Change the association of which this role is a part. This method does _NOT_ send out any events since it is meant to be used to change a role without an association into a role that belongs to an association, after that the role is not suppost to change associations.

- **getKlass**

```
public synchronized ClassModel getKlass()
```

Return the class that plays this role in the association. This method is called getKlass and not getClass because java.lang.Object already contains a getClass operation that can’t be overwritten.

- **getName**

```
public synchronized String getName()
```

Return the name of the role.

- **setName**

```
public synchronized void setName(String name)
```

Change the name of the role. This method sends out an event to notify the views of the change.

- **getMultiplicity**

```
public synchronized String getMultiplicity()
```

Return the multiplicity of the role.

- **setMultiplicity**

```
public synchronized void setMultiplicity(String multiplicity)
```

Change the multiplicity of the role. This method sends out an event to notify the views of the change.

- **getAggregate**

```
public synchronized int getAggregate()
```

Return whether the association is a NORMAL association, an AGGREGATE or a COMPOSITE.

- **setAggregate**

```
public synchronized void setAggregate(int aggregate)
```

Change whether the association is a normal association, an aggregate or a composite.

Parameters:
aggregate - valid values are NORMAL, AGGREGATE and COMPOSITE.

- **isNavigable**

```
public synchronized boolean isNavigable()
```

Return whether this role is navigable in this direction

- **setNavigable**

```
public synchronized void setNavigable(boolean navigable)
```

Set whether this role is navigable in this direction

- **isOrdered**

```
public synchronized boolean isOrdered()
```

Return whether the objects that play this role are ordered or not.

- **setOrdered**

```
public synchronized void setOrdered(boolean ordered)
```

Set whether the objects that play this role are ordered or not.

- **toString**

```
public synchronized String toString()
```

Convert this role to a string. This will return the name of the role or the string "<unnamed>" if it has no name.

Overrides:
toString in class Object

11.7 Class model.AttributeModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.MemberModel
                   |
                   +----model.AttributeModel
```

---

public class **AttributeModel**
extends MemberModel

The class AttributeModel represents one attribute of a class.

Variable Index

- **derived**
  Is the attribute derived from other attributes and thus its value is determined based on these attributes.
- **initialValue**
  The initial value of the attribute.
- **type**
  The data type of the attribute.

Constructor Index

- **AttributeModel**()

Method Index

- **getInitialValue**()
  Return the initial value of the attribute.
- **getType**()
  Return the data type of this attribute.
- **isDerived**()
  Return whether the value of the attribute is derived from other attributes.
- **setDerived**(boolean)
  Set whether the value of the attribute is derived from other attributes.
- **setInitialValue**(String)
  Change the initial value of the attribute.

- **setType**(String)
  Change the data type of the attribute.
- **toString**()
  Convert the attribute to a string.
- **toString**(boolean)
  Convert the attribute to a string.

Variables

type

```
private String type
```

The data type of the attribute.

initialValue

```
private String initialValue
```

The initial value of the attribute. That is, the value it will have before any value is assigned to it.

derived

```
private boolean derived
```

Is the attribute derived from other attributes and thus its value is determined based on these attributes.

Constructors

AttributeModel

```
public AttributeModel()
```

Methods

getType

```
public synchronized String getType()
```

Return the data type of this attribute.

setType

```
public synchronized void setType(String type)
```

Change the data type of the attribute. This method will notify the views by sending an event them.

getInitialValue

```
public synchronized String getInitialValue()
```

Return the initial value of the attribute.

setInitialValue

```
public synchronized void setInitialValue(String value)
```

Change the initial value of the attribute. This method will notify the views by sending an event to them.

isDerived

```
public synchronized boolean isDerived()
```

Return whether the value of the attribute is derived from other attributes.

setDerived

```
public synchronized void setDerived(boolean derived)
```

Set whether the value of the attribute is derived from other attributes.

toString

```
public synchronized String toString()
```

Convert the attribute to a string. This method calls toString(false)

Overrides:

toString in class Object

toString

```
public synchronized String toString(boolean terse)
```

Convert the attribute to a string.

Parameters:

terse - If true then only the name (or "<unnamed>" if there is no name) is returned. If false then the name is optionally prefixed with a "/" and/or a "$" to show that the attribute is derived or a class member, respectively. The name is followed by the data type and the initial value (if there is one).

11.8 Class model.ClassModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.ElementModel
                   |
                   +----model.ClassModel
```

---

public class **ClassModel**
extends ElementModel

The class ClassModel represents one class. This class keeps track of the attributes and operations of the class and the relations it has.

Variable Index

- **associationRoles**
  All the roles the class plays in various associations
- **attributes**
  The attributes of the class.
- **operations**
  The operations of the class.
- **usesRelations**
  All the uses-relations that class has

Constructor Index

- **ClassModel**()

Method Index

- **addAssociationRole**(AssociationRoleModel)
  Add a new role to the class.
- **addAttribute**(AttributeModel)
  Add a new attribute to the class.
- **addOperation**(OperationModel)
  Add a new operation to the class.
- **addUsesRelation**(UsesRelationModel)
- **getAssociationRole**(int)
  Return one specific association-role.

- **getAssociationRoleCount**()
  Return the number of association-roles this class has.
- **getAttribute**(int)
  Return one specific attribute.
- **getAttributeCount**()
  Return the number of attributes this class has.
- **getOperation**(int)
  Return one specific operation.
- **getOperationCount**()
  Return the number of operations this class has.
- **getUsesRelation**(int)
- **getUsesRelationCount**()
- **removeAllAssociationRoles**()
  Remove all roles from the class.
- **removeAllAttributes**()
  Remove all attributes from the class.
- **removeAllOperations**()
  Remove all operations from the class.
- **removeAllUsesRelations**()
- **removeAssociationRole**(AssociationRoleModel)
  Remove a role from the class.
- **removeAttribute**(AttributeModel)
  Remove an attribute from the class.
- **removeOperation**(OperationModel)
  Remove an operation from the class.
- **removeUsesRelation**(UsesRelationModel)

Variables

attributes

```
private Vector attributes
```

The attributes of the class.

operations

```
private Vector operations
```

The operations of the class.

associationRoles

```
private Vector associationRoles
```

All the roles the class plays in various associations

usesRelations

```
private Vector usesRelations
```

All the uses-relations that class has

ClassModel

```
public ClassModel()
```

getAttributeCount

```
public synchronized int getAttributeCount()
```

Return the number of attributes this class has.

getAttribute

```
public synchronized AttributeModel getAttribute(int index)
```

Return one specific attribute.

Parameters:

index - the number of the attribute. This value should be between 0 and getAttributeCount()

addAttribute

```
public synchronized void addAttribute(AttributeModel a)
```

Add a new attribute to the class. If the attribute is already added, this method does nothing. In all other cases, the method will notify the views of the addition by sending out an event.

Parameters:

a - the new attribute

removeAttribute

```
public synchronized void removeAttribute(AttributeModel a)
```

Remove an attribute from the class. If the class doens’t have the attribute, nothing happens. In all other cases, the method will notify the views of the removal by sending out an event.

Parameters:

a - the attribute that will be removed.

- **removeAllAttributes**

```
public synchronized void removeAllAttributes()
```

Remove all attributes from the class. This method calls removeAttribute for all attributes of the class.

- **getOperationCount**

```
public synchronized int getOperationCount()
```

Return the number of operations this class has.

- **getOperation**

```
public synchronized OperationModel getOperation(int index)
```

Return one specific operation.

Parameters:

index - the number of the operations. This value should be between 0 and getOperationCount()

- **addOperation**

```
public synchronized void addOperation(OperationModel o)
```

Add a new operation to the class. If the operation is already added, this method does nothing. In all other cases, the method will notify the views of the addition by sending out an event.

Parameters:

o - the new operation

- **removeOperation**

```
public synchronized void removeOperation(OperationModel o)
```

Remove an operation from the class. If the class doens't have the operation, nothing happens. In all other cases, the method will notify the views of the removal by sending out an event.

Parameters:

o - the operation that will be removed.

- **removeAllOperations**

```
public synchronized void removeAllOperations()
```

Remove all operations from the class. This method calls removeOperation for all operations of the class.

- **getAssociationRoleCount**

```
public synchronized int getAssociationRoleCount()
```

Return the number of association-roles this class has.

- **getAssociationRole**

```
public synchronized AssociationRoleModel getAssociationRole(int index)
```

Return one specific association-role.

Parameters:

index - the number of the role. This value should be between 0 and getAssociationRoleCount()

- **addAssociationRole**

```
synchronized void addAssociationRole(AssociationRoleModel a)
```

Add a new role to the class. If the role is already added, this method does nothing. In all other cases, the method will notify the views of the addition by sending out an event.

Parameters:

o - the new role

- **removeAssociationRole**

```
public synchronized void removeAssociationRole(AssociationRoleModel a)
```

Remove a role from the class. If the class doens't have the role, nothing happens. In all other cases, the method will notify the views of the removal by sending out an event.

Parameters:

a - the role that will be removed.

- **removeAllAssociationRoles**

```
public synchronized void removeAllAssociationRoles()
```

Remove all roles from the class. This method calls removeAssociationRole for all roles of the class.

- **getUsesRelationCount**

```
public synchronized int getUsesRelationCount()
```

- **getUsesRelation**

```
public synchronized UsesRelationModel getUsesRelation(int index)
```

addUsesRelation

```
public synchronized void addUsesRelation(UsesRelationModel u)
```

removeUsesRelation

```
public synchronized void removeUsesRelation(UsesRelationModel u)
```

removeAllUsesRelations

```
public synchronized void removeAllUsesRelations()
```

11.9 Class model.ElementModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.ElementModel
```

---

public class **ElementModel**
extends ModelBase

This class represents one element. It is the base class for ClassModel and PackageModel.

See Also:
ClassModel, PackageModel

Variable Index

- **generalizations**
  The generalizations of this element
- **name**
  The name of the element
- **owner**
  The package that owns this element
- **stereotype**
  The stereotype of the element
- **visibility**
  The visibility (private, protected or public) of this element

Constructor Index

- **ElementModel**()
  This constructor is package private because ElementModel should never be directly instanciated, instead a ClassModel or a PackageModel should be created.

Method Index

- **addGeneralization**(GeneralizationModel)
  Add a new generalization to this element.
- **getGeneralization**(int)
  Return one specifiec generalization model.

- **getGeneralizationCount**()
  Return the number of generalizations this element has.
- **getName**()
  Return the name of the element.
- **getOwner**()
  Return the package that owns this element.
- **getStereotype**()
  Return the stereotype of the element or null if there is none.
- **getVisibility**()
  Return the visibility of this element.
- **removeGeneralization**(GeneralizationModel)
  Remove a generalization from this element.
- **setName**(String)
  Change the name of the association.
- **setOwner**(PackageModel)
  Change the owner of this element.
- **setStereotype**(String)
  Set the stereotype.
- **setVisibility**(Visibility)
  Set the visibility of this element.
- **toString**()
  Convert this element to a string.

Variables

name

```
private String name
```

The name of the element

stereotype

```
private String stereotype
```

The stereotype of the element

owner

```
private PackageModel owner
```

The package that owns this element

visibility

```
private Visibility visibility
```

The visibility (private, protected or public) of this element

- **generalizations**

```
private Vector generalizations
```

The generalizations of this element

Constructors

- **ElementModel**

```
ElementModel()
```

This constructor is package private because ElementModel should never be directly instanciated, instead a ClassModel or a PackageModel should be created.

Methods

- **getName**

```
public synchronized String getName()
```

Return the name of the element.

- **setName**

```
public synchronized void setName(String name)
```

Change the name of the association. This method will send out an event to all the views to allow them to update themselves.

Parameters:
name - the new name

- **getStereotype**

```
public synchronized String getStereotype()
```

Return the stereotype of the element or null if there is none.

- **setStereotype**

```
public synchronized void setStereotype(String stereotype)
```

Set the stereotype. This method will send out an event to all the views to allow them to update themselves.

Parameters:
stereotype - the new stereotype, use null or "" if there is no stereotype

- **getGeneralizationCount**

```
public synchronized int getGeneralizationCount()
```

Return the number of generalizations this element has.

- **getGeneralization**

```
public synchronized GeneralizationModel getGeneralization(int index)
```

Return one specifiec generalization model.

Parameters:

index - the number of the generalization. This value should be between 0 and getGeneralizationCount()

- **addGeneralization**

```
synchronized void addGeneralization(GeneralizationModel g)
```

Add a new generalization to this element. If the generalization is already added, this method does nothing. In all other cases, the method will notify the views of the addition by sending out an event.

Parameters:

g - the new generalization

- **removeGeneralization**

```
public synchronized void removeGeneralization(GeneralizationModel g)
```

Remove a generalization from this element. If the element doens’t have the generalization, nothing happens. In all other cases, the method will notify the views of the removal by sending out an event.

Parameters:

g - the generalization that will be removed.

- **getOwner**

```
public synchronized PackageModel getOwner()
```

Return the package that owns this element.

- **setOwner**

```
public synchronized void setOwner(PackageModel owner)
```

Change the owner of this element. This method will notify the views of the change by sending out an event.

getVisibility

```
public synchronized Visibility getVisibility()
```

Return the visibility of this element.

setVisibility

```
public synchronized void setVisibility(Visibility visibility)
```

Set the visibility of this element. This method will notify the views of the change by sending out an event.

toString

```
public synchronized String toString()
```

Convert this element to a string. This will return the name of the element or if there is no name the string "<unnamed>"

Overrides:

toString in class Object

11.10 Class model.GeneralizationModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.GeneralizationModel
```

---

public class **GeneralizationModel**
extends ModelBase

This class represents a generalization relation between elements.

Variable Index

- **name**
  An optional name for the generalization, may be null
- **subtype**
  The element that is the sub-type
- **supertype**
  The element that is the super-type

Constructor Index

- **GeneralizationModel**(ElementModel, ElementModel)

Method Index

- **getName**()
  Return the name of the generalization.
- **getSubtype**()
  Return the sub-type of the generalization.
- **getSupertype**()
  Return the super-type of the generalization.
- **setName**(String)
  Change the name of the generalization.

Variables

- **name**

```
private String name
```

An optional name for the generalization, may be null

- **supertype**

```
private ElementModel supertype
```

The element that is the super-type

- **subtype**

```
private ElementModel subtype
```

The element that is the sub-type

Constructors

- **GeneralizationModel**

```
public GeneralizationModel(ElementModel supertype,
                           ElementModel subtype)
```

Methods

- **getName**

```
public synchronized String getName()
```

Return the name of the generalization.

- **setName**

```
public synchronized void setName(String name)
```

Change the name of the generalization. This method will send out an event to all the views to allow them to update themselves.

- **getSupertype**

```
public synchronized ElementModel getSupertype()
```

Return the super-type of the generalization.

- **getSubtype**

```
public synchronized ElementModel getSubtype()
```

Return the sub-type of the generalization.

11.11 Class model.MemberModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.MemberModel
```

---

public class **MemberModel**
extends ModelBase

A class can have attributes and operations. This class has all the things that attibutes and operation have in common.

Variable Index

- **classMember**
  Is the member part of an object or part of the class (static)
- **name**
  The name of the member
- **owner**
  The class which is the owner of this member
- **visibility**
  The visibility (private, protected or public) of the member

Constructor Index

- **MemberModel**()

Method Index

- **getName**()
  Return the name of the member
- **getOwner**()
  Return the class which owns this member.
- **getVisibility**()
  Return the visibility of the member
- **isClassMember**()
  Is this member a class member (static).
- **setClassMember**(boolean)
  Set whether the member is a class member.

- **setName**(String)
  Change the name of the member.
- **setOwner**(ClassModel)
  Set the owner of this member.
- **setVisibility**(Visibility)
  Change the visibility of the member.

Variables

- **name**

```
private String name
```

  The name of the member

- **visibility**

```
private Visibility visibility
```

  The visibility (private, protected or public) of the member

- **classMember**

```
private boolean classMember
```

  Is the member part of an object or part of the class (static)

- **owner**

```
private ClassModel owner
```

  The class which is the owner of this member

Constructors

- **MemberModel**

```
MemberModel()
```

Methods

- **getName**

```
public synchronized String getName()
```

  Return the name of the member

- **setName**

```
public synchronized void setName(String name)
```

Change the name of the member. This method will send out an event to all the views to allow them to update themselves.

Parameters:
name - the new name

- **getVisibility**

```
public synchronized Visibility getVisibility()
```

Return the visibility of the member

- **setVisibility**

```
public synchronized void setVisibility(Visibility visibility)
```

Change the visibility of the member. This method will send out an event to all the views to allow them to update themselves.

Parameters:
visibility - the new visibility

- **isClassMember**

```
public synchronized boolean isClassMember()
```

Is this member a class member (static).

- **setClassMember**

```
public synchronized void setClassMember(boolean classMember)
```

Set whether the member is a class member. This method will send out an event to all the views to allow them to update themselves.

- **getOwner**

```
public synchronized ClassModel getOwner()
```

Return the class which owns this member.

- **setOwner**

```
public synchronized void setOwner(ClassModel owner)
```

Set the owner of this member. This method will send out an event to all the views to allow them to update themselves.

Parameters:
owner - the new class that will own this member

11.12 Class model.ModelBase

```
java.lang.Object
   |
   +----model.ModelBase
```

---

public class **ModelBase**
extends Object
implements Serializable

ModelBase is the base class of the model hierarchy. This class contains all the functionality needed to send events to the views when a model changes.

Variable Index

- **listenerList**
  The list of listeners.
- **notes**
  Each model can have a number of notes attached to it.
- **properties**
  A set of properties that views can associate with a model.

Constructor Index

- **ModelBase**()
  The constructor of ModelBase is package-private; use the constructor of a derived class to instantiate a model.

Method Index

- **addModelChangeListener**(ModelChangeListener)
  Add a listener to the list that's notified each time a change to the data model occurs.
- **addNote**(NoteModel)
  Add a note to the model.
- **fireValueAdded**(int, Object)
  Fire an event signaling that a value was added to the model.
- **fireValueChanged**(int, Object)
  Fire an event signaling that a value of the model has changed.
- **fireValueRemoved**(int, Object)
  Fire an event signaling that a value was removed from the model.

- **getNote**(int)
  Get a note.
- **getNoteCount**()
  Get the number of Notes.
- **getProperty**(String)
  Get a property that a view has associated with this model.
- **removeModelChangeListener**(ModelChangeListener)
  Remove a listener from the list that's notified each time a change to the data model occurs.
- **removeNote**(NoteModel)
  Remove a note from the model.
- **setProperty**(String, Object)
  Set a property in the hashtable of this model.

Variables

listenerList

```
private transient EventListenerList listenerList
```

The list of listeners. All objects that are interested in receiving events when the model changes should implement the interface ModelChangeListener and register themself with the addModelChangeListener method below. The addModelChangeListener method will add the interested object as a lister to this listenerList.

properties

```
private Hashtable properties
```

A set of properties that views can associate with a model. The model itself will not use any of the information that is stored in this hashtable, but the views that store the information may later you it for themselves. Examples of things to store in here: the size and the location of a view.

See Also:
getProperty, setProperty

notes

```
private Vector notes
```

Each model can have a number of notes attached to it. This is currently not used/implemented in the views, but this vector can be used if such implementation is later added

Constructors

ModelBase

```
ModelBase()
```

The constructor of ModelBase is package-private; use the constructor of a derived class to instantiate a model.

- **getProperty**

```
public synchronized Object getProperty(String propName)
```

Get a property that a view has associated with this model. Views can associate a property to this model by calling setProperty()

Parameters:
propName - the name of the property
Returns:
the value of the property
See Also:
setProperty

- **setProperty**

```
public synchronized void setProperty(String propName,
                                     Object value)
```

Set a property in the hashtable of this model. The model itself will not use any of the information that is stored in this hashtable, but the views that store the information may later you it for themselves. Examples of things that can be stored: the size of a view and the location of a view.

Parameters:
propName - the name of the property
value - the value of the property
See Also:
getProperty

- **getNoteCount**

```
public synchronized int getNoteCount()
```

Get the number of Notes.

Returns:
the number of notes attachted to this model

- **getNote**

```
public synchronized NoteModel getNote(int index)
```

Get a note.

Parameters:
index - the number of the note. The notes are numbered from 0 to getNoteCount().
Returns:
the note
See Also:
addNote, removeNote

addNote

```
public synchronized void addNote(NoteModel n)
```

Add a note to the model.

Parameters:
n - the note to add.
See Also:
getNote, removeNote

removeNote

```
public synchronized void removeNote(NoteModel n)
```

Remove a note from the model. Calling this method with a note that is not attached to the model is legal, it that case the call does nothing.

Parameters:
n - the note to remove
See Also:
getNote, addNote

addModelChangeListener

```
public void addModelChangeListener(ModelChangeListener l)
```

Add a listener to the list that’s notified each time a change to the data model occurs.

Parameters:
l - the ModelChangeListener
See Also:
removeModelChangeListener

removeModelChangeListener

```
public void removeModelChangeListener(ModelChangeListener l)
```

Remove a listener from the list that's notified each time a change to the data model occurs.

Parameters:
l - the ModelChangeListener
See Also:
addModelChangeListener

fireValueChanged

```
protected void fireValueChanged(int what,
                                Object value)
```

Fire an event signaling that a value of the model has changed.

Parameters:
what - an integer identifying which type of value has been changed. See ModelChangeEvent for a list of valid values for parameter what.
value - the new value of the variable.
See Also:
ModelChangeEvent

fireValueAdded

```
protected void fireValueAdded(int what,
                              Object value)
```

Fire an event signaling that a value was added to the model.

Parameters:
what - an integer identifying which type of value has been added. See ModelChangeEvent for a list of valid values for parameter what.
value - the new value.
See Also:
ModelChangeEvent

fireValueRemoved

```
protected void fireValueRemoved(int what,
                                Object value)
```

Fire an event signaling that a value was removed from the model.

Parameters:
what - an integer identifying which type of value has been removed. See ModelChangeEvent for a list of valid values for parameter what.
value - the value just before it was removed.
See Also:
ModelChangeEvent

11.13 Class model.NoteModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.NoteModel
```

---

public class **NoteModel**
extends ModelBase

The class NoteModel allows notes to be added to another model. A note is just a piece of text that helps the designer to clarify the design but has no meaning to the computer.

Variable Index

- **owner**
  To which model this note belongs
- **text**
  The text of the note.

Constructor Index

- **NoteModel**(ModelBase)
  Constructor; creates an empty note.
- **NoteModel**(ModelBase, String)
  Constructor; creates an note with specified text.

Method Index

- **getOwner**()
  Return the model to which this note belongs.
- **getText**()
  Return the text of the note.
- **setOwner**(ModelBase)
  Change the owner of this note.
- **setText**(String)
  Change the text of the note.

Variables

- **text**

```
private String text
```

The text of the note.

- **owner**

```
private ModelBase owner
```

To which model this note belongs

Constructors

- **NoteModel**

```
public NoteModel(ModelBase owner)
```

Constructor; creates an empty note.

- **NoteModel**

```
public NoteModel(ModelBase owner,
                 String text)
```

Constructor; creates an note with specified text.

Methods

- **getOwner**

```
public synchronized ModelBase getOwner()
```

Return the model to which this note belongs.

- **setOwner**

```
public synchronized void setOwner(ModelBase owner)
```

Change the owner of this note.

- **getText**

```
public synchronized String getText()
```

Return the text of the note.

- **setText**

```
public synchronized void setText(String text)
```

Change the text of the note.

11.14 Class model.OperationModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.MemberModel
                   |
                   +----model.OperationModel
```

---

public class **OperationModel**
extends MemberModel

The class OperationModel represents one operation of a class.

Variable Index

- **abstractOperation**
  Will this operation be implemented in the class to which it belongs or will it be abstract and be implemented in a subclass.
- **parameters**
  The parameter of the operation
- **returnType**
  The return type of the operation.

Constructor Index

- **OperationModel**()

Method Index

- **addParameter**(ParameterModel)
  Add a new parameter
- **getParameter**(int)
  Return one specific parameter
- **getParameterCount**()
  Return the number of parameters this operation has.
- **getReturnType**()
  Return the return type of this operation
- **isAbstract**()
  Return if the operation is abstract.

- **removeParameter**(int)
  Remove a parameter
- **setAbstract**(boolean)
  Set whether the operation is abstract.
- **setParameter**(int, ParameterModel)
  Set one parameter
- **setParameters**(Vector)
  Set all the parameters of the operation
- **setReturnType**(String)
  Change the return type of this operation.
- **toString**()
  Convert the operation to a string.
- **toString**(boolean)
  Convert the operation to a string.

Variables

parameters

```
private Vector parameters
```

The parameter of the operation

returnType

```
private String returnType
```

The return type of the operation.

abstractOperation

```
private boolean abstractOperation
```

Will this operation be implemented in the class to which it belongs or will it be abstract and be implemented in a subclass.

Constructors

OperationModel

```
public OperationModel()
```

Methods

getParameterCount

```
public synchronized int getParameterCount()
```

Return the number of parameters this operation has.

- **getParameter**

```
public synchronized ParameterModel getParameter(int index)
```

Return one specific parameter

- **setParameter**

```
public synchronized void setParameter(int index,
                                      ParameterModel param)
```

Set one parameter

- **addParameter**

```
public synchronized void addParameter(ParameterModel param)
```

Add a new parameter

- **removeParameter**

```
public synchronized void removeParameter(int index)
```

Remove a parameter

- **setParameters**

```
public synchronized void setParameters(Vector newParams)
```

Set all the parameters of the operation

- **getReturnType**

```
public synchronized String getReturnType()
```

Return the return type of this operation

- **setReturnType**

```
public synchronized void setReturnType(String type)
```

Change the return type of this operation. This method will notify the views by sending an event to them.

- **isAbstract**

```
public synchronized boolean isAbstract()
```

Return if the operation is abstract.

setAbstract

```
public synchronized void setAbstract(boolean abstr)
```

Set whether the operation is abstract. This method will notify the views by sending an event to them.

toString

```
public synchronized String toString(boolean terse)
```

Convert the operation to a string.

Parameters:

terse - If true then only the name (or "<unnamed>" if there is no name) is returned. If false then the name is optionally prefixed with a '$' to show that the operation is a class member. The name is followed by a list of parameters (if there are any) and an optional return type.

toString

```
public synchronized String toString()
```

Convert the operation to a string. This calls toString(false);

Overrides:

toString in class Object

11.15 Class model.PackageModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.ElementModel
                   |
                   +----model.PackageModel
```

---

public class **PackageModel**
extends ElementModel

This class contains the information about one package

Variable Index

- **associations**
  The associations between classes inside this package
- **elements**
  The elements (packages and classes) inside this package

Constructor Index

- **PackageModel**()

Method Index

- **addAssociation**(AssociationModel)
  Add a new association.
- **addElement**(ElementModel)
  Add a new element to the package
- **getAssociation**(int)
  Get one specific association.
- **getAssociationCount**()
  Get the number of association between classes inside the package.
- **getElement**(int)
  Get one specific element from the package
- **getElementCount**()
  Return the number of elements in the package
- **removeAssociation**(AssociationModel)
  Remove an association from the package.

- **removeElement**(ElementModel)
  Remove an element from the package

Variables

- **associations**

```
private Vector associations
```

  The associations between classes inside this package

- **elements**

```
private Vector elements
```

  The elements (packages and classes) inside this package

Constructors

- **PackageModel**

```
public PackageModel()
```

Methods

- **getElementCount**

```
public synchronized int getElementCount()
```

  Return the number of elements in the package

- **getElement**

```
public synchronized ElementModel getElement(int index)
```

  Get one specific element from the package

- **addElement**

```
public synchronized void addElement(ElementModel e)
```

  Add a new element to the package

- **removeElement**

```
public synchronized void removeElement(ElementModel e)
```

  Remove an element from the package

- **getAssociationCount**

```
public synchronized int getAssociationCount()
```

Get the number of association between classes inside the package.

- **getAssociation**

```
public synchronized AssociationModel getAssociation(int index)
```

Get one specific association.

- **addAssociation**

```
public synchronized void addAssociation(AssociationModel a)
```

Add a new association.

- **removeAssociation**

```
public synchronized void removeAssociation(AssociationModel a)
```

Remove an association from the package.

11.16 Class model.ParameterModel

```
java.lang.Object
   |
   +----model.ParameterModel
```

---

public class **ParameterModel**
extends Object
implements Serializable

This class contains the information about one parameter of a operations.

Variable Index

- **defaultValue**
  The default value of the parameter, may be null
- **name**
  The name of the parameter
- **type**
  The data type of the parameter

Constructor Index

- **ParameterModel**(String, String, String)
  Constructor; create a new parameter.

Method Index

- **getDefaultValue**()
  Return the default value of the parameter
- **getName**()
  Return the name of the parameter
- **getType**()
  Return the data type of the parameter

Variables

- **name**

```
private String name
```

The name of the parameter

- **type**

```
private String type
```

The data type of the parameter

- **defaultValue**

```
private String defaultValue
```

The default value of the parameter, may be null

Constructors

- **ParameterModel**

```
public ParameterModel(String name,
                      String type,
                      String defaultValue)
```

Constructor; create a new parameter.

Parameters:

name - the name of the parameter
type - the data type of the parameter
defaultValue - the default value of the parameter, may be null

Methods

- **getName**

```
public synchronized String getName()
```

Return the name of the parameter

- **getType**

```
public synchronized String getType()
```

Return the data type of the parameter

- **getDefaultValue**

```
public synchronized String getDefaultValue()
```

Return the default value of the parameter

11.17 Class model.UsesRelationModel

```
java.lang.Object
   |
   +----model.ModelBase
           |
           +----model.UsesRelationModel
```

---

public class **UsesRelationModel**
extends ModelBase

Variable Index

- **operations**
- **provider**
- **user**

Constructor Index

- **UsesRelationModel**()

Variables

- **user**

```
private ClassModel user
```

- **provider**

```
private ClassModel provider
```

- **operations**

```
private Vector operations
```

Constructors

- **UsesRelationModel**

```
public UsesRelationModel()
```

11.18 Class view.AssociationDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----com.sun.java.swing.JDialog
                                           |
                                           +----view.AssociationDialog
```

---

public class **AssociationDialog**
extends JDialog
implements ActionListener, DocumentListener, ModelChangeListener

This is a dialog box that allows the user to edit an Association and its roles

---

Variable Index

- **closeButton**
  The close button, when the user click on it, the dialog disappears
- **model**
  The model of which this is a view
- **name**
  A text-field that shows the name of the association
- **stereotype**
  A text-field that shows the stereotype of the association
- **tabbedPane**
  The TabbedPane in which the roles are shown

Constructor Index

- **AssociationDialog**(Frame, AssociationModel)
  Constructor; creates the dialog box for AssociationModel model

Method Index

- **actionPerformed**(ActionEvent)
  This method is called when the user has pushed a button or pressed the enter-key in a text field.
- **changedUpdate**(DocumentEvent)
  This method is called when the user has changed something in a text-field that is neither an insert nor a remove.
- **handleDocumentEvent**(DocumentEvent)
  This method is called when the value in the name or stereotype text-fields has changed.
- **insertUpdate**(DocumentEvent)
  This method is called when the user has entered something in a text-field.
- **removeUpdate**(DocumentEvent)
  This method is called when the user has removed something from a text-field.
- **showRole**(AssociationRoleModel)
  This method makes sure that the tabbed pane shows the specified role.
- **valueAdded**(ModelChangeEvent)
  This method is called when a value is added to the model.
- **valueChanged**(ModelChangeEvent)
  This method is called when a value in the model has changed.
- **valueRemoved**(ModelChangeEvent)
  This method is called when a value is removed from the model.

Variables

model

```
private AssociationModel model
```

The model of which this is a view

tabbedPane

```
private JTabbedPane tabbedPane
```

The TabbedPane in which the roles are shown

name

```
private JTextField name
```

A text-field that shows the name of the association

stereotype

```
private JTextField stereotype
```

A text-field that shows the stereotype of the association

closeButton

```
private JButton closeButton
```

The close button, when the user click on it, the dialog disappears

Constructors

AssociationDialog

```
public AssociationDialog(Frame frame,
                         AssociationModel model)
```

Constructor; creates the dialog box for AssociationModel model

Parameters:

frame - the window this dialog belongs to.
model - the model of which this is a view

Methods

showRole

```
public void showRole(AssociationRoleModel arm)
```

This method makes sure that the tabbed pane shows the specified role.

Parameters:

arm - the role to show in the tabbed pane

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

This method is called when the user has pushed a button or pressed the enter-key in a text field. This method is public as an implementation detail and should not be called by a programmer.

insertUpdate

```
public void insertUpdate(DocumentEvent e)
```

This method is called when the user has entered something in a text-field. This method is public as an implementation detail and should not be called by a programmer.

removeUpdate

```
public void removeUpdate(DocumentEvent e)
```

This method is called when the user has removed something from a text-field. This method is public as an implementation detail and should not be called by a programmer.

changedUpdate

```
public void changedUpdate(DocumentEvent e)
```

This method is called when the user has changed something in a text-field that is neither an insert nor a remove. This method is public as an implementation detail and should not be called by a programmer.

handleDocumentEvent

```
private void handleDocumentEvent(DocumentEvent e)
```

This method is called when the value in the name or stereotype text-fields has changed. This method will call the setName or setStereotype methods of the model to propagate the change.

valueChanged

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when a value in the model has changed. This method will update the values in the text-fields. This method is public as an implementation detail and should not be called by a programmer.

valueAdded

```
public void valueAdded(ModelChangeEvent e)
```

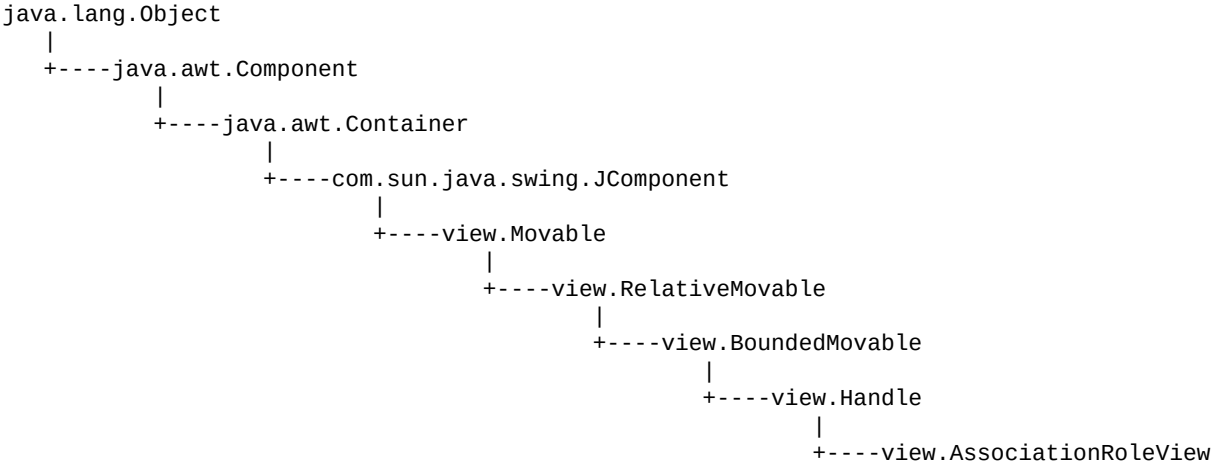
This method is called when a value is added to the model. This method is public as an implementation detail and should not be called by a programmer.

valueRemoved

```
public void valueRemoved(ModelChangeEvent e)
```

This method is called when a value is removed from the model. This method is public as an implementation detail and should not be called by a programmer.

11.19 Class view.AssociationRoleView



---

public class **AssociationRoleView**
extends Handle
implements ModelChangeListener

This is a view of the AssociationRoleModel. It shows the name and multiplicity of the role.

Variable Index

- **classView**
  The view of the class that plays the role in the association
- **container**
  The container that is the parent of the name and multiplicity labels
- **model**
  The model of which this is a view
- **multiplicity**
  A label that shows the multiplicity of the role
- **multiplicityContainer**
  The container that is the parent of the multiplicity label.
- **name**
  A label that shows the name of the role
- **nameContainer**
  The container that is the parent of the name label, it moves relative to this.

Constructor Index

- **AssociationRoleView**(AssociationRoleModel, ClassView, Container)
  The constructor of the AssociationRoleView class.

Method Index

- **getClassView**()
  Return the view of the class that plays the role in the association.
- **raise**()
  Place the role-view at the front; raising it above all other components.
- **remove**()
  This method is called when the role-view is being removed.
- **valueAdded**(ModelChangeEvent)
  This method is doesn't do anything but needs to be there to satisfy the ModelChangeListener interface signature.
- **valueChanged**(ModelChangeEvent)
  This method is called when a value in the model changes.
- **valueRemoved**(ModelChangeEvent)
  This method is doesn't do anything but needs to be there to satisfy the ModelChangeListener interface signature.

Variables

- **model**

```
private AssociationRoleModel model
```

  The model of which this is a view

- **classView**

```
private ClassView classView
```

  The view of the class that plays the role in the association

- **container**

```
private Container container
```

  The container that is the parent of the name and multiplicity labels

- **name**

```
private JLabel name
```

  A label that shows the name of the role

nameContainer

```
private RelativeMovable nameContainer
```

The container that is the parent of the name label, it moves relative to this.

multiplicity

```
private JLabel multiplicity
```

A label that shows the multiplicity of the role

multiplicityContainer

```
private RelativeMovable multiplicityContainer
```

The container that is the parent of the multiplicity label.

Constructors

AssociationRoleView

```
public AssociationRoleView(AssociationRoleModel model,
                           ClassView classView,
                           Container container)
```

The constructor of the AssociationRoleView class.

Parameters:

model - the model of which this is a view
classView - the view of the class that plays the role in the association
container - the container that is the parent of the name and multiplicity labels.

Methods

raise

```
public void raise()
```

Place the role-view at the front; raising it above all other components.

Overrides:

raise in class Movable

getClassView

```
public ClassView getClassView()
```

Return the view of the class that plays the role in the association.

remove

```
public void remove()
```

This method is called when the role-view is being removed. It will remove the name and multiplicity containers of the screen.

valueChanged

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when a value in the model changes. This method is public as an implementation detail and should not be called by a programmer.

valueAdded

```
public void valueAdded(ModelChangeEvent e)
```

This method is doesn’t do anything but needs to be there to satisfy the ModelChangeListener interface signature. This method is public as an implementation detail and should not be called by a programmer.

valueRemoved

```
public void valueRemoved(ModelChangeEvent e)
```

This method is doesn’t do anything but needs to be there to satisfy the ModelChangeListener interface signature. This method is public as an implementation detail and should not be called by a programmer.

11.20 Class view.AssociationView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----view.Movable
                                   |
                                   +----view.Resizable
                                           |
                                           +----view.AssociationView
```

---

public class **AssociationView**
extends Resizable
implements ModelChangeListener

This class represents an association; it does this either with a line (in case of a binary association) or a diamond (is all other cases).

Variable Index

- **container**
- **diamondVisible**
  Whether the diamond is visible or not
- **lines**
  All the lines that are connected to the roles
- **mode**
  A reference to the Mode object, this is used to check in which mode the PackageWindow is
- **model**
  The model of this view
- **modelToViewMap**
  A hashtable to convert a model into a view
- **mouseDragPoint**
  The point where the user first pressed the mouse button when the user is dragging the mouse
- **name**
  The name label of the association
- **namePanel**
  The panel that contains the name and the stereotype labels
- **packageView**

- **roles**
  All the roles of this association
- **stereotype**
  The stereotype label of the association

Constructor Index

- **AssociationView**(AssociationModel, PackageView, Container)

Method Index

- **addRole**(AssociationRoleModel)
  Add a new role to this association.
- **contains**(int, int)
  This method is called by the AWT event handling to determin whether the point (x, y) lies inside of this component.
- **doResize**(int, int)
  This method is called to resize the component.
- **findFrame**()
  Find the Frame ancestor of this component.
- **findMode**(Component)
  Find the PackageWindow ancestor of this component and get its Mode.
- **getModel**()
  Return the model of which this is a view
- **mouseClicked**(MouseEvent)
  This method is called when the user clicks the mouse on the association-view.
- **mouseDragged**(MouseEvent)
  This method is called when the user drags the mouse on the association-view.
- **mousePressed**(MouseEvent)
  This method is called when the user presses a mouse button on the association-view.
- **mouseReleased**(MouseEvent)
  This method is called when the user releases the mouse button.
- **paintComponent**(Graphics)
  This method overrides the paintComponent of the super class to allow the diamond to be painted in a non-rectangular fashion.
- **remove**()
  This method is called when the association is being removed.
- **removeRole**(AssociationRoleModel)
  Remove a role from this association.
- **updateName**()
  Update the contents of the name label.
- **updatePoly**()

- **updateStereotype**()
  Update the contents of the stereotype label.
- **valueAdded**(ModelChangeEvent)
  This method is called when a role is added to the association model.
- **valueChanged**(ModelChangeEvent)
  This method is called when the name of the stereotype of the association model has changed.
- **valueRemoved**(ModelChangeEvent)
  This method is called when a role is removed from the association model.

Variables

model

```
private AssociationModel model
```

The model of this view

packageView

```
private PackageView packageView
```

container

```
private Container container
```

name

```
private JLabel name
```

The name label of the association

stereotype

```
private JLabel stereotype
```

The stereotype label of the association

namePanel

```
private JPanel namePanel
```

The panel that contains the name and the stereotype labels

roles

```
private Vector roles
```

All the roles of this association

- **lines**

```
private Hashtable lines
```

  All the lines that are connected to the roles

- **modelToViewMap**

```
private Hashtable modelToViewMap
```

  A hashtable to convert a model into a view

- **diamondVisible**

```
private boolean diamondVisible
```

  Whether the diamond is visible or not

- **mode**

```
private Mode mode
```

  A reference to the Mode object, this is used to check in which mode the PackageWindow is

- **mouseDragPoint**

```
private Point mouseDragPoint
```

  The point where the user first pressed the mouse button when the user is dragging the mouse

Constructors

- **AssociationView**

```
public AssociationView(AssociationModel model,
                       PackageView packageView,
                       Container container)
```

Methods

- **getModel**

```
public AssociationModel getModel()
```

  Return the model of which this is a view

- **addRole**

```
private void addRole(AssociationRoleModel roleModel)
```

Add a new role to this association. Called in response to an event from the model.

- **removeRole**

```
private void removeRole(AssociationRoleModel model)
```

Remove a role from this association. Called in response to an event from the model.

- **remove**

```
void remove()
```

This method is called when the association is being removed. It will remove all the lines to the class-views.

- **updateName**

```
private void updateName()
```

Update the contents of the name label. Called in response to an event from the model.

- **updateStereotype**

```
private void updateStereotype()
```

Update the contents of the stereotype label. If the stereotype is an empty string, the label is removed (so that only the name is visible and not the stereotype label). If the stereotype is not an empty string the label is added and it's value is updated.

- **doResize**

```
protected void doResize(int deltaX,
                        int deltaY)
```

This method is called to resize the component.

Overrides:
doResize in class Resizable

- **valueAdded**

```
public void valueAdded(ModelChangeEvent e)
```

This method is called when a role is added to the association model. This method is public as an implementation detail and should not be called by a programmer.

- **valueRemoved**

```
public void valueRemoved(ModelChangeEvent e)
```

This method is called when a role is removed from the association model. This method is public as an implementation detail and should not be called by a programmer.

valueChanged

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when the name of the stereotype of the association model has changed. This method is public as an implementation detail and should not be called by a programmer.

mouseClicked

```
public void mouseClicked(MouseEvent e)
```

This method is called when the user clicks the mouse on the association-view. When the user double clicks on the association-view an AssociationDialog is created and shown, allowing the user to edit the values in the association model. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
mouseClicked in class Movable

mouseDragged

```
public void mouseDragged(MouseEvent e)
```

This method is called when the user drags the mouse on the association-view. If the current mode is the AddAssociationRole mode a line is shown from the point where the user pressed the mouse button down to the current position of the mouse. If the user releases the mouse button on a class-view, a new association role is created. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
mouseDragged in class Resizable

mousePressed

```
public void mousePressed(MouseEvent e)
```

This method is called when the user presses a mouse button on the association-view. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
mousePressed in class Resizable

mouseReleased

```
public void mouseReleased(MouseEvent e)
```

This method is called when the user releases the mouse button. If the user has dragged the mouse to a class-view and the current mode is the AddAssociationRole mode, a new association role is created. This method is public as an implementation detail and should

not be called by a programmer.

Overrides:
mouseReleased in class Resizable

paintComponent

```
public void paintComponent(Graphics g)
```

This method overrides the paintComponent of the super class to allow the diamond to be painted in a non-rectangular fashion. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
paintComponent in class JComponent

updatePoly

```
private void updatePoly()
```

contains

```
public boolean contains(int x,
                        int y)
```

This method is called by the AWT event handling to determin whether the point (x, y) lies inside of this component. True is returned if this is the case, false otherwise. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
contains in class JComponent

findFrame

```
protected Frame findFrame()
```

Find the Frame ancestor of this component. The frame is needed to show dialog boxes.

findMode

```
protected Mode findMode(Component c)
```

Find the PackageWindow ancestor of this component and get its Mode.

11.21 Class view.AttributeDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----com.sun.java.swing.JDialog
                                           |
                                           +----view.AttributeDialog
```

---

public class **AttributeDialog**
extends JDialog
implements ActionListener, DocumentListener, ModelChangeListener

This is a dialog box that allows the user to edit an attribute.

Variable Index

- **classMember**
  A checkbox that shows whether the attribute is a normal member or a class member (static)
- **closeButton**
  The close button, when the user click on it, the dialog disappears
- **derived**
  A checkbox that show whether the value of the attribute is derived from other attributes
- **initialValue**
  A text-field that shows the initial value of the attribute
- **model**
  The model of which this is a view
- **name**
  A text-field that shows the name of the attribute
- **type**
  A text-field that shows the data type of the attribute
- **visibility**
  A combobox that shows the visibility of the attribute

Constructor Index

- **AttributeDialog**(Frame, AttributeModel)
  Constructor; creates the dialog box for AttributeModel model

Method Index

- **actionPerformed**(ActionEvent)
  This method is called when the user has pushed a button or pressed the enter-key in a text field.
- **changedUpdate**(DocumentEvent)
  This method is called when the user has changed something in a text-field that is neither an insert nor a remove.
- **handleDocumentEvent**(DocumentEvent)
  This method is called when the value in the name, type or initial value text-fields has changed.
- **insertUpdate**(DocumentEvent)
  This method is called when the user has entered something in a text-field.
- **removeUpdate**(DocumentEvent)
  This method is called when the user has removed something from a text-field.
- **valueAdded**(ModelChangeEvent)
- **valueChanged**(ModelChangeEvent)
  This method is called when a value in the model has changed.
- **valueRemoved**(ModelChangeEvent)

Variables

model

```
private AttributeModel model
```

The model of which this is a view

name

```
private JTextField name
```

A text-field that shows the name of the attribute

initialValue

```
private JTextField initialValue
```

A text-field that shows the initial value of the attribute

type

```
private JTextField type
```

A text-field that shows the data type of the attribute

- **visibility**

```
private JComboBox visibility
```

A combobox that shows the visibility of the attribute

- **classMember**

```
private JCheckBox classMember
```

A checkbox that shows whether the attribute is a normal member or a class member (static)

- **derived**

```
private JCheckBox derived
```

A checkbox that show whether the value of the attribute is derived from other attributes

- **closeButton**

```
private JButton closeButton
```

The close button, when the user click on it, the dialog disappears

Constructors

- **AttributeDialog**

```
public AttributeDialog(Frame parent,
                       AttributeModel model)
```

Constructor; creates the dialog box for AttributeModel model

Parameters:

parent - the window this dialog belongs to.
model - the model of which this is a view

Methods

- **actionPerformed**

```
public void actionPerformed(ActionEvent e)
```

This method is called when the user has pushed a button or pressed the enter-key in a text field. This method is public as an implementation detail and should not be called by a programmer.

insertUpdate

```
public void insertUpdate(DocumentEvent e)
```

This method is called when the user has entered something in a text-field. This method is public as an implementation detail and should not be called by a programmer.

removeUpdate

```
public void removeUpdate(DocumentEvent e)
```

This method is called when the user has removed something from a text-field. This method is public as an implementation detail and should not be called by a programmer.

changedUpdate

```
public void changedUpdate(DocumentEvent e)
```

This method is called when the user has changed something in a text-field that is neither an insert nor a remove. This method is public as an implementation detail and should not be called by a programmer.

handleDocumentEvent

```
private void handleDocumentEvent(DocumentEvent e)
```

This method is called when the value in the name, type or initial value text-fields has changed. This method will call the setName, setType or setInitialValue methods of the model to propagate the change.

valueAdded

```
public void valueAdded(ModelChangeEvent e)
```

valueRemoved

```
public void valueRemoved(ModelChangeEvent e)
```

valueChanged

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when a value in the model has changed. This method will update the values in the text-fields. This method is public as an implementation detail and should not be called by a programmer.

11.22 Class view.AttributeView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----com.sun.java.swing.JLabel
                                   |
                                   +----view.AttributeView
```

---

public class **AttributeView**
extends JLabel
implements ModelChangeListener

This class shows one attribute on the screen. The attribute is shown as a trafic light icon showing the visibility, followed by the name, data type and initial value of the attribute

---

Variable Index

- **model**
  The model of which this class is a view

Constructor Index

- **AttributeView**(AttributeModel)
  Constructor; create an AttributeView for AttributeModel model

Method Index

- **deselected**()
  This method is called when the attribute is deselected.
- **getModel**()
  Return the model of which this class is a view.
- **selected**()
  This method is called when the attribute is selected.
- **valueAdded**(ModelChangeEvent)
- **valueChanged**(ModelChangeEvent)
  This method is called when a value in the model has changed.

- **valueRemoved**(ModelChangeEvent)

Variables

- **model**

```
private AttributeModel model
```

The model of which this class is a view

Constructors

- **AttributeView**

```
public AttributeView(AttributeModel model)
```

Constructor; create an AttributeView for AttributeModel model

Methods

- **valueChanged**

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when a value in the model has changed. This method will update the label and the icon. This method is public as an implementation detail and should not be called by a programmer.

- **valueAdded**

```
public void valueAdded(ModelChangeEvent e)
```

- **valueRemoved**

```
public void valueRemoved(ModelChangeEvent e)
```

- **selected**

```
public void selected()
```

This method is called when the attribute is selected.

- **deselected**

```
public void deselected()
```

This method is called when the attribute is deselected.

- **getModel**

```
public AttributeModel getModel()
```

Return the model of which this class is a view.

11.23 Class view.AttributesView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----com.sun.java.swing.JPanel
                                   |
                                   +----view.AttributesView
```

---

public class **AttributesView**
extends JPanel
implements ModelChangeListener

This class show a list of attributes; all the attributes of a class.

Variable Index

- **box**
- **model**
  The class model that owns the attributes that are shown by this class
- **modelToViewMap**
  A hashtable that maps and attribute model to a view
- **mouseListener**
  if this is not null than all mouse events that occur on the attribute views will be send to this listener.

Constructor Index

- **AttributesView**(ClassModel, MouseListener)
  The AttributesView constructor; show a list of the attributes of ClassModel model.

Method Index

- **addAttribute**(AttributeModel)
  Add a new attribute to the list
- **removeAttribute**(AttributeModel)
  Remove an attribute from the list.

- **valueAdded**(ModelChangeEvent)
  This method is called when an attribute was added to the model.
- **valueChanged**(ModelChangeEvent)
- **valueRemoved**(ModelChangeEvent)
  This method is called when an attribute was removed from the model.

Variables

- **model**

```
private ClassModel model
```

  The class model that owns the attributes that are shown by this class

- **modelToViewMap**

```
private Hashtable modelToViewMap
```

  A hashtable that maps and attribute model to a view

- **box**

```
private Box box
```

- **mouseListener**

```
private MouseListener mouseListener
```

  if this is not null than all mouse events that occur on the attribute views will be send to this listener.

Constructors

- **AttributesView**

```
public AttributesView(ClassModel model,
                      MouseListener mouseListener)
```

  The AttributesView constructor; show a list of the attributes of ClassModel model.

  Parameters:
  model - the classmodel that owns the attributes that are to be shown by this class.
  mouseListener - if this is not null than all mouse events that occur on the attribute views will be send to this listener.

Methods

- **addAttribute**

```
private void addAttribute(AttributeModel attrib)
```

Add a new attribute to the list

- **removeAttribute**

```
private void removeAttribute(AttributeModel attrib)
```

Remove an attribute from the list.

- **valueAdded**

```
public void valueAdded(ModelChangeEvent e)
```

This method is called when an attribute was added to the model. This method is public as an implementation detail and should not be called by a programmer.

- **valueRemoved**

```
public void valueRemoved(ModelChangeEvent e)
```

This method is called when an attribute was removed from the model. This method is public as an implementation detail and should not be called by a programmer.

- **valueChanged**

```
public void valueChanged(ModelChangeEvent e)
```

11.24 Class view.BoundedMovable

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----view.Movable
                                   |
                                   +----view.RelativeMovable
                                           |
                                           +----view.BoundedMovable
```

---

public class **BoundedMovable**
extends RelativeMovable

The class Movable contains the functionality that allows the user to move a component on the screen to a new location. The user can use the mouse to drag a component (that is a subclass of Movable) and this class will allow it to move.

RelativeMovable inherits from Movable and allows a component on the screen to be moved relative to another component. For example, A is a Movable and B is a RelativeMovable that moves relative to A. The user can move B around and the RelativeMovable will keep track of the relative offset to A. If the user moves A around, B will move as well to keep the offset the same.

BoundedMovable inherits from RelativeMovable. It restricts the movement of B, relative to A, so that B is always on the contours of A.

Constructor Index

- **BoundedMovable**(Movable, ModelBase, String)
  The constructor of the BoundedMovable

Method Index

- **distance**(Point, Point)
  Calculate the distance between points P and Q.
- **project**(Point, Point, Point)
  Project point P on the line between A and B.
- **updateLocation**()

Constructors

BoundedMovable

```
public BoundedMovable(Movable movable,
                      ModelBase modelBase,
                      String locationName)
```

The constructor of the BoundedMovable

Parameters:

movable - the movable on which outline the BoundedMovable will move
modelBase - the model in which the location of the BoundedMovable will be stored
locationName - the name under which the location of the BoundedMovable will be stored in the model.

Methods

updateLocation

```
protected void updateLocation()
```

Overrides:

updateLocation in class RelativeMovable

project

```
private Point project(Point p,
                      Point a,
                      Point b)
```

Project point P on the line between A and B. Return the point on the line after the projection of P. If the projection of P lies not between A and B, return A or B itself, whichever is closer.

distance

```
private int distance(Point p,
                     Point q)
```

Calculate the distance between points P and Q. This method calculates (p.x - q.x)^2 + (p.y - p.y)^2. It does not take the square root. Thus it really returns the square of the distance instead of the distance itself. For the purpose of finding the point with minimal distance this doesn’t matter.

11.25 Class view.ClassDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----com.sun.java.swing.JDialog
                                           |
                                           +----view.ClassDialog
```

---

public class **ClassDialog**
extends JDialog
implements ActionListener, DocumentListener, MouseListener, ModelChangeListener

This is a dialog box that allows the user to edit a ClassModel.

Variable Index

- **attributesView**
  A list which displays all the attributes of this class.
- **closeButton**
  The close button, when the user click on it, the dialog disappears
- **deleteAttribButton**
  Buttons to create a new attribute, edit an attribute or delete an attribute
- **deleteOperButton**
  Buttons to create a new operation, edit an operation or delete an operation
- **editAttribButton**
  Buttons to create a new attribute, edit an attribute or delete an attribute
- **editOperButton**
  Buttons to create a new operation, edit an operation or delete an operation
- **frame**
- **model**
  The class model can be edited by this dialog.
- **name**
  The text-field that shows the name of the class
- **newAttribButton**
  Buttons to create a new attribute, edit an attribute or delete an attribute
- **newOperButton**
  Buttons to create a new operation, edit an operation or delete an operation

- **operationsView**
  A list which displays all the operations of this class.
- **selectedAttrib**
  The attribute that is currently selected.
- **selectedOper**
  The operation that is currently selected.
- **stereotype**
  The text-field that shows the stereotype of the class
- **tabbedPane**
  The TabbedPane that allows switching between general class pane and the attributes and operation panes.
- **visibility**
  The combobox that shows the visibility of the class

Constructor Index

- **ClassDialog**(Frame, ClassModel)
  Constructor; creates the dialog box for ClassModel model

Method Index

- **actionPerformed**(ActionEvent)
  This method is called when the user has pushed a button or pressed the enter-key in a text field.
- **changedUpdate**(DocumentEvent)
  This method is called when the user has changed something in a text-field that is neither an insert nor a remove.
- **handleDocumentEvent**(DocumentEvent)
  This method is called when the value in the name or stereotype text-fields has changed.
- **insertUpdate**(DocumentEvent)
  This method is called when the user has entered something in a text-field.
- **mouseClicked**(MouseEvent)
  This method is called when the user that clicked the mouse on an attribute or operation.
- **mouseEntered**(MouseEvent)
- **mouseExited**(MouseEvent)
- **mousePressed**(MouseEvent)
- **mouseReleased**(MouseEvent)
- **removeUpdate**(DocumentEvent)
  This method is called when the user has removed something from a text-field.
- **valueAdded**(ModelChangeEvent)
- **valueChanged**(ModelChangeEvent)
  This method is called when a value in the model has changed.

- **valueRemoved**(ModelChangeEvent)

- **model**

```
private ClassModel model
```

  The class model can be edited by this dialog.

- **frame**

```
private Frame frame
```

- **tabbedPane**

```
private JTabbedPane tabbedPane
```

  The TabbedPane that allows switching between general class pane and the attributes and operation panes.

- **name**

```
private JTextField name
```

  The text-field that shows the name of the class

- **stereotype**

```
private JTextField stereotype
```

  The text-field that shows the stereotype of the class

- **visibility**

```
private JComboBox visibility
```

  The combobox that shows the visibility of the class

- **closeButton**

```
private JButton closeButton
```

  The close button, when the user click on it, the dialog disappears

- **attributesView**

```
private AttributesView attributesView
```

  A list which displays all the attributes of this class.

- **operationsView**

```
private OperationsView operationsView
```

A list which displays all the operations of this class.

- **selectedAttrib**

```
private AttributeView selectedAttrib
```

The attribute that is currently selected.

- **selectedOper**

```
private OperationView selectedOper
```

The operation that is currently selected.

- **newAttribButton**

```
private JButton newAttribButton
```

Buttons to create a new attribute, edit an attribute or delete an attribute

- **editAttribButton**

```
private JButton editAttribButton
```

Buttons to create a new attribute, edit an attribute or delete an attribute

- **deleteAttribButton**

```
private JButton deleteAttribButton
```

Buttons to create a new attribute, edit an attribute or delete an attribute

- **newOperButton**

```
private JButton newOperButton
```

Buttons to create a new operation, edit an operation or delete an operation

- **editOperButton**

```
private JButton editOperButton
```

Buttons to create a new operation, edit an operation or delete an operation

- **deleteOperButton**

```
private JButton deleteOperButton
```

Buttons to create a new operation, edit an operation or delete an operation

Constructors

ClassDialog

```
public ClassDialog(Frame frame,
                   ClassModel model)
```

Constructor; creates the dialog box for ClassModel model

Parameters:

frame - the window this dialog belongs to.
model - the model of which this is a view

Methods

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

This method is called when the user has pushed a button or pressed the enter-key in a text field. This method is public as an implementation detail and should not be called by a programmer.

insertUpdate

```
public void insertUpdate(DocumentEvent e)
```

This method is called when the user has entered something in a text-field. This method is public as an implementation detail and should not be called by a programmer.

removeUpdate

```
public void removeUpdate(DocumentEvent e)
```

This method is called when the user has removed something from a text-field. This method is public as an implementation detail and should not be called by a programmer.

changedUpdate

```
public void changedUpdate(DocumentEvent e)
```

This method is called when the user has changed something in a text-field that is neither an insert nor a remove. This method is public as an implementation detail and should not be called by a programmer.

handleDocumentEvent

```
private void handleDocumentEvent(DocumentEvent e)
```

This method is called when the value in the name or stereotype text-fields has changed. This method will call the setName or setStereotype methods of the model to propagate the change.

- **valueChanged**

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when a value in the model has changed. This method will update the values in the text-fields. This method is public as an implementation detail and should not be called by a programmer.

- **valueAdded**

```
public void valueAdded(ModelChangeEvent e)
```

- **valueRemoved**

```
public void valueRemoved(ModelChangeEvent e)
```

- **mouseClicked**

```
public void mouseClicked(MouseEvent e)
```

This method is called when the user that clicked the mouse on an attribute or operation. This selects the attribute/operation and if the click was a double-click that a dialog box is shown that allows the user to edit the attribute/operation. This method is public as an implementation detail and should not be called by a programmer.

- **mousePressed**

```
public void mousePressed(MouseEvent e)
```

- **mouseReleased**

```
public void mouseReleased(MouseEvent e)
```

- **mouseEntered**

```
public void mouseEntered(MouseEvent e)
```

- **mouseExited**

```
public void mouseExited(MouseEvent e)
```

11.26 Class view.ClassView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----view.Movable
                                   |
                                   +----view.Resizable
                                           |
                                           +----view.ElementView
                                                   |
                                                   +----view.ClassView
```

---

public class **ClassView**
extends ElementView

This class shows one class element on the screen. At the top of the classview there are the stereotype and name labels and the visibility icon. The visibility icon uses the colors of a traffic light to show public visibility (green), protected visibility (yellow) and private visibility (red). Below the labels and icon there is a list of attributes (AttributesView). Below the attributes is a list of operations (OperationsView).

Variable Index

- **attributesView**
  The list of attributes
- **mode**
  A reference to the PackageWindow's Mode.
- **model**
  The class model that this class is a view of
- **mouseDragPoint**
  The point where the mouse was dragged to.
- **operationsView**
  The list of operations
- **popup**
  The menu that pops up when the user presses the right mouse button on the class view

Constructor Index

- **ClassView**(ClassModel, PackageView)
  Constructor; create a view of the class model.

Method Index

- **addNotify**()
- **doResize**(int, int)
  This method is called to resize the component.
- **findMode**(Component)
  Find the PackageWindow ancestor of this component and get its Mode.
- **getModel**()
  Return the model that is shown with this view.
- **mouseClicked**(MouseEvent)
  This method is called when the user clicks a mouse button.
- **mouseDragged**(MouseEvent)
  This method is called when the user drags the mouse while holding a mouse button down.
- **mousePressed**(MouseEvent)
  This method is called when the user presses a mouse button.
- **mouseReleased**(MouseEvent)
  This method is called when the user releases a mouse button.
- **paintComponent**(Graphics)
  Override paintComponent in the super class to clear the background of the class view before drawing the component.
- **updatePoly**()
  Update the outline polygon.

Variables

model

```
private ClassModel model
```

The class model that this class is a view of

attributesView

```
private AttributesView attributesView
```

The list of attributes

operationsView

```
private OperationsView operationsView
```

The list of operations

- **popup**

```
private JPopupMenu popup
```

The menu that pops up when the user presses the right mouse button on the class view

- **mouseDragPoint**

```
private Point mouseDragPoint
```

The point where the mouse was dragged to.

- **mode**

```
private Mode mode
```

A reference to the PackageWindow's Mode.

Constructors

- **ClassView**

```
public ClassView(ClassModel model,
                 PackageView parentPackage)
```

Constructor; create a view of the class model.

Parameters:

model - the model that will be shown with this view
parentPackage - the package that owns this class

Methods

- **addNotify**

```
public void addNotify()
```

Overrides:

addNotify in class Movable

- **getModel**

```
public ClassModel getModel()
```

Return the model that is shown with this view.

- **mousePressed**

```
public void mousePressed(MouseEvent e)
```

This method is called when the user presses a mouse button. If the mouse button is the right mouse button, the popup menu is shown. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
mousePressed in class ElementView

mouseClicked

```
public void mouseClicked(MouseEvent e)
```

This method is called when the user clicks a mouse button. If the user double-clicks, then a dialog box is shown allowing the user to edit the class. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
mouseClicked in class Movable

mouseDragged

```
public void mouseDragged(MouseEvent e)
```

This method is called when the user drags the mouse while holding a mouse button down. If the current mode is AddAssociationRole then a line from the class to the current mouse position is drawn. If the mouse button is release over another class or over a AssociationView, a new association role will be created. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
mouseDragged in class Resizable

mouseReleased

```
public void mouseReleased(MouseEvent e)
```

This method is called when the user releases a mouse button. If the current mode is AddAssociationRole and the mouse button is release over another class or over a AssociationView, a new association role will be created. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
mouseReleased in class Resizable

doResize

```
protected void doResize(int x,
                        int y)
```

This method is called to resize the component.

Overrides:
doResize in class ElementView

updatePoly

```
private void updatePoly()
```

Update the outline polygon.

findMode

```
protected Mode findMode(Component c)
```

Find the PackageWindow ancestor of this component and get its Mode.

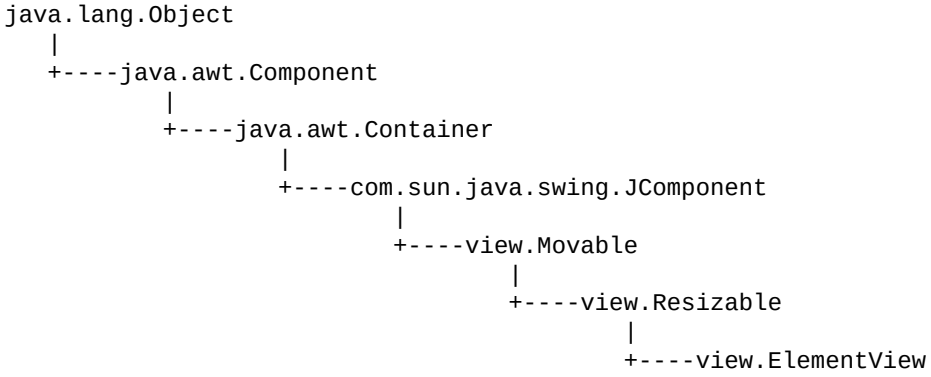
paintComponent

```
public void paintComponent(Graphics g)
```

Override paintComponent in the super class to clear the background of the class view before drawing the component.

Overrides:
paintComponent in class JComponent

11.27 Class view.ElementView



---

public class **ElementView**
extends Resizable
implements ModelChangeListener, SelectionListener

Class ElementView is the base class of ClassView and PackageView. It contains the things ClassView and PackageView have in common.

See Also:
ClassView, PackageView

Variable Index

- **contentPanel**
  The panel that shows the rest of the information of the element.
- **model**
  The model that is shown with this view.
- **nameLabel**
  The label that is used to show the name of the element.
- **namePanel**
  The panel that shows the name and stereotype of the element.
- **parentPackage**
  The package that owns this element
- **stereoTypeLabel**
  The label that is used to show the stereotype of the element.
- **suppressed**
  If the element is suppressed, only the name and stereotype is shown.
- **unsuppressedSize**
  When the element is suppressed, remember the size of the element it had when it was not suppressed.

Constructor Index

- **ElementView**(ElementModel, PackageView)
  Constructor; create a view of the element model.

Method Index

- **deselected**()
  This method is called when the element is deselected.
- **doResize**(int, int)
  This method is called to resize the component.
- **findFrame**()
  Find the Frame ancestor of this component.
- **getParentPackage**()
  Return the package that owns this element
- **isSuppressed**()
  Return whether the element is currently suppressed.
- **mousePressed**(MouseEvent)
  This method is called when the user pressed a mouse button on this element.
- **selected**()
  This method is called when the element is selected.
- **setName**(String)
  Change the name that is shown in the name label.
- **setStereotype**(String)
  Change the stereotype that is shown in the stereotype label.
- **setSuppressed**(boolean)
  Suppress or unsuppress this element.
- **setVisibility**(Visibility)
  Change the visibility that is shown in the visibility icon.
- **valueAdded**(ModelChangeEvent)
- **valueChanged**(ModelChangeEvent)
  This method is called when the name, stereotype or visibility of the model has changed.
- **valueRemoved**(ModelChangeEvent)

Variables

- **model**

```
private ElementModel model
```

  The model that is shown with this view.

- **parentPackage**

```
protected PackageView parentPackage
```

  The package that owns this element

- **namePanel**

```
protected JPanel namePanel
```

  The panel that shows the name and stereotype of the element.

- **contentPanel**

```
protected JPanel contentPanel
```

  The panel that shows the rest of the information of the element.

- **nameLabel**

```
private JLabel nameLabel
```

  The label that is used to show the name of the element.

- **stereoTypeLabel**

```
private JLabel stereoTypeLabel
```

  The label that is used to show the stereotype of the element.

- **suppressed**

```
private boolean suppressed
```

  If the element is suppressed, only the name and stereotype is shown.

- **unsuppressedSize**

```
private Dimension unsuppressedSize
```

  When the element is suppressed, remember the size of the element it had when it was not suppressed.

- **ElementView**

```
ElementView(ElementModel model,
            PackageView parentPackage)
```

  Constructor; create a view of the element model.

Parameters:
model - the model that will be shown with this view
parentPackage - the package that owns this element.

Methods

- **getParentPackage**

```
public PackageView getParentPackage()
```

Return the package that owns this element

- **setSuppressed**

```
public void setSuppressed(boolean suppressed)
```

Suppress or unsuppress this element. When the element is suppressed, only the name and stereotype is shown.

- **isSuppressed**

```
public boolean isSuppressed()
```

Return whether the element is currently suppressed.

- **setName**

```
public void setName(String name)
```

Change the name that is shown in the name label.

Overrides:
setName in class Component

- **setStereotype**

```
public void setStereotype(String stereotype)
```

Change the stereotype that is shown in the stereotype label.

- **setVisibility**

```
public void setVisibility(Visibility visibility)
```

Change the visibility that is shown in the visibility icon.

- **valueChanged**

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when the name, stereotype or visibility of the model has changed. This method calls setName, setStereotype or setVisibility to propagate the change.

valueAdded

```
public void valueAdded(ModelChangeEvent e)
```

valueRemoved

```
public void valueRemoved(ModelChangeEvent e)
```

selected

```
public void selected()
```

This method is called when the element is selected. It changes the color of the name and stereotype labels.

deselected

```
public void deselected()
```

This method is called when the element is deselected. It changes the color of the name and stereotype labels back to normal.

mousePressed

```
public void mousePressed(MouseEvent e)
```

This method is called when the user pressed a mouse button on this element. This will select the element.

Overrides:
mousePressed in class Resizable

doResize

```
protected void doResize(int x,
                        int y)
```

This method is called to resize the component.

Overrides:
doResize in class Resizable

findFrame

```
protected Frame findFrame()
```

Find the Frame ancestor of this component. The frame is needed to show dialog boxes.

11.28 Class view.Handle

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----view.Movable
                                   |
                                   +----view.RelativeMovable
                                           |
                                           +----view.BoundedMovable
                                                   |
                                                   +----view.Handle
```

---

public class **Handle**
extends BoundedMovable

A Handle is a little yellow circle that can be used to move the lines of the associations.

Constructor Index

- **Handle**(Movable, ModelBase, String)
  The constructor the the Handle.

Method Index

- **paintComponent**(Graphics)
  Paint a little yellow circle.

Constructors

Handle

```
public Handle(Movable movable,
              ModelBase modelBase,
              String locationName)
```

The constructor the the Handle.

Parameters:

movable - the movable on which outline the handle will move
modelBase - the model in which the location of the handle will be stored
locationName - the name under which the location of the handle will be stored in

the model.

Methods

- **paintComponent**

```
public void paintComponent(Graphics g)
```

Paint a little yellow circle.

Overrides:

paintComponent in class JComponent

11.29 Class view.Line

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----view.Line
```

---

public class **Line**
extends Component
implements MouseListener, MouseMotionListener, SelectionListener

Most of the relations are drawn using lines between ClassViews. The class Line draws the lines. A line can consist of one or more straight lines. Each of these straight lines is drawn by the class LineSegment.

See Also:
LineSegment

Variable Index

- **a**
  One side of the line
- **b**
  The other side of the line
- **container**
  The container to which the line segments are added
- **lineSegments**
  All the line segments of the line

Constructor Index

- **Line**(Handle, Handle, Container)
  The constructor

Method Index

- **deselected**()
  This method is called when the line is deselected.
- **mouseClicked**(MouseEvent)
- **mouseDragged**(MouseEvent)

- **mouseEntered**(MouseEvent)
- **mouseExited**(MouseEvent)
- **mouseMoved**(MouseEvent)
- **mousePressed**(MouseEvent)
  This method is called when the user presses a mouse button on a line segment.
- **mouseReleased**(MouseEvent)
- **raise**()
  Raise all the line segments to the front.
- **remove**()
  This method is called when the line is about to be removed, it will remove all the line segments from the screen.
- **selected**()
  This method is called when the line is selected.

Variables

- **a**

```
private Handle a
```

  One side of the line

- **b**

```
private Handle b
```

  The other side of the line

- **container**

```
private Container container
```

  The container to which the line segments are added

- **lineSegments**

```
private Vector lineSegments
```

  All the line segments of the line

Constructors

- **Line**

```
public Line(Handle a,
            Handle b,
            Container c)
```

The constructor

Parameters:

a - one side of the line
b - the other side of the line
c - the container to which the line segments are added

Methods

- **selected**

```
public void selected()
```

This method is called when the line is selected. This method calls the selected() method of all the line segments of this line.

- **deselected**

```
public void deselected()
```

This method is called when the line is deselected. This method calls the deselected() method of all the line segments of this line.

- **mousePressed**

```
public void mousePressed(MouseEvent e)
```

This method is called when the user presses a mouse button on a line segment. This will raise all the line segments to the front and set the line as selection.

- **mouseDragged**

```
public void mouseDragged(MouseEvent e)
```

- **mouseClicked**

```
public void mouseClicked(MouseEvent e)
```

- **mouseReleased**

```
public void mouseReleased(MouseEvent e)
```

- **mouseEntered**

```
public void mouseEntered(MouseEvent e)
```

- **mouseExited**

```
public void mouseExited(MouseEvent e)
```

- **mouseMoved**

```
public void mouseMoved(MouseEvent e)
```

- **raise**

```
public void raise()
```

Raise all the line segments to the front.

- **remove**

```
public void remove()
```

This method is called when the line is about to be removed, it will remove all the line segments from the screen.

11.30 Class view.LineSegment

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----view.LineSegment
```

---

public class **LineSegment**
extends JComponent
implements ComponentListener, ChangeListener

Most of the relations are drawn using lines between ClassViews. The class Line draws the lines. A line can consist of one or more straight lines. Each of these straight lines is drawn by the class LineSegment.

See Also:
Line

Variable Index

- **a**
  One side of the line
- **b**
  The other side of the line
- **x**
- **y1**
- **y2**

Constructor Index

- **LineSegment**(Handle, Handle)
  Create a new LineSegment that draws a line between a and b

Method Index

- **addListeners**()
- **addNotify**()

- **componentHidden**(ComponentEvent)
- **componentMoved**(ComponentEvent)
  This method is called when handle a or handle b has been moved.
- **componentResized**(ComponentEvent)
- **componentShown**(ComponentEvent)
- **contains**(int, int)
  Return true if point (ex, ey) is on or close by the line, false otherwise.
- **deselected**()
  This method is called when the line segment is deselected.
- **paint**(Graphics)
- **raise**()
  Bring the line segment to the front.
- **selected**()
  This method is called when the line segment is selected.
- **stateChanged**(ChangeEvent)
  This method is called when a scrollpane has been scrolled.
- **updateLocationAndSize**()
  If handle A or handle B have changed position this method is called to change the location and the size of the line.

Variables

a

```
private Handle a
```

One side of the line

b

```
private Handle b
```

The other side of the line

x

```
private int x
```

y1

```
private int y1
```

y2

```
private int y2
```

Constructors

LineSegment

```
public LineSegment(Handle a,
                   Handle b)
```

Create a new LineSegment that draws a line between a and b

Methods

- **addNotify**

```
public void addNotify()
```

Overrides:
addNotify in class JComponent

- **addListeners**

```
private void addListeners()
```

- **raise**

```
public void raise()
```

Bring the line segment to the front.

- **selected**

```
public void selected()
```

This method is called when the line segment is selected.

- **deselected**

```
public void deselected()
```

This method is called when the line segment is deselected.

- **updateLocationAndSize**

```
private void updateLocationAndSize()
```

If handle A or handle B have changed position this method is called to change the location and the size of the line.

- **paint**

```
public void paint(Graphics g)
```

Overrides:
paint in class JComponent

- **contains**

```
public boolean contains(int ex,
                        int ey)
```

Return true if point (ex, ey) is on or close by the line, false otherwise.

Overrides:

contains in class JComponent

- **componentMoved**

```
public void componentMoved(ComponentEvent e)
```

This method is called when handle a or handle b has been moved. This method calls updateLocationAndSize();

- **componentResized**

```
public void componentResized(ComponentEvent e)
```

- **componentShown**

```
public void componentShown(ComponentEvent e)
```

- **componentHidden**

```
public void componentHidden(ComponentEvent e)
```

- **stateChanged**

```
public void stateChanged(ChangeEvent e)
```

This method is called when a scrollpane has been scrolled. This method calls updateLocationAndSize();

11.31 Class view.MainWindow

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
                                   |
                                   +----com.sun.java.swing.JFrame
                                           |
                                           +----view.MainWindow
```

---

public class **MainWindow**
extends JFrame
implements ActionListener

This class creates the main window of the editor. It shows the main menu bar, allows files to be loaded and saved and it shows the model tree of the global package.

Variable Index

- **globalScopePackage**
  The top most package, the global scope.
- **mainWindow**
  A static reference to the (only) instance of this class
- **modelTree**
  The model tree that shows the global scope package.
- **windows**
  A list of PackageWindows

Constructor Index

- **MainWindow**(String)
  Create a new main window, load the file with the name filename if filename is not null.

Method Index

- **actionPerformed**(ActionEvent)
  This method is called when the user has chosen a new look and feel from the menu.

- **addWindow**(Window)
  Add a new window to the list of windows.
- **closeAllWindows**()
  Close all the windows in the list of windows.
- **getModelTree**()
  Return the model tree of the global scope package.
- **load**(String)
  Load a global scope package and its contents from file.
- **quit**()
  Quit the editor.
- **setGlobalScopePackage**(PackageModel)
  Change the global scope package.

Variables

mainWindow

```
public static MainWindow mainWindow
```

A static reference to the (only) instance of this class

globalScopePackage

```
private PackageModel globalScopePackage
```

The top most package, the global scope.

modelTree

```
private ModelTree modelTree
```

The model tree that shows the global scope package.

windows

```
private Vector windows
```

A list of PackageWindows

Constructors

MainWindow

```
public MainWindow(String filename)
```

Create a new main window, load the file with the name filename if filename is not null.

Methods

getModelTree

```
public ModelTree getModelTree()
```

Return the model tree of the global scope package.

- **actionPerformed**

```
public void actionPerformed(ActionEvent event)
```

This method is called when the user has chosen a new look and feel from the menu. Try to load the new look and feel and if that fails, disable the menu entry.

- **addWindow**

```
public void addWindow(Window w)
```

Add a new window to the list of windows.

- **closeAllWindows**

```
private void closeAllWindows()
```

Close all the windows in the list of windows.

- **setGlobalScopePackage**

```
private void setGlobalScopePackage(PackageModel globalScopePackage)
```

Change the global scope package. Close all the windows that show the old package and create a new window.

- **quit**

```
private void quit()
```

Quit the editor.

- **load**

```
private void load(String filename)
```

Load a global scope package and its contents from file.

11.32 Class view.MenuFactory

```
java.lang.Object
   |
   +----view.MenuFactory
```

---

public class **MenuFactory**
extends Object

This class contains only static methods that are used to create menus and toolbars.

Variable Index

- **resources**

Constructor Index

- **MenuFactory**()

Method Index

- ()
- **createMenu**(String, MyAction[])
  Create a normal menu with the actions given in parameter actions.
- **createPopupMenu**(MyAction[])
  Create a popup menu with the actions given in parameter actions.
- **createRadioMenu**(String, MyAction[])
  Create a menu with radio items with the actions given in parameter actions.
- **createRadioToolbar**(MyAction[])
  Create a tool bar with radio buttons with the actions given in parameter actions.
- **createToolbar**(MyAction[])
  Create a tool bar with the actions given in parameter actions.
- **getResourceString**(String)

Variables

- **resources**

```
private static ResourceBundle resources
```

Constructors

- **MenuFactory**

```
public MenuFactory()
```

Methods

- **createMenu**

```
public static JMenu createMenu(String name,
                               MyAction actions[])
```

Create a normal menu with the actions given in parameter actions.

- **createRadioMenu**

```
public static JMenu createRadioMenu(String name,
                                    MyAction actions[])
```

Create a menu with radio items with the actions given in parameter actions.

- **createPopupMenu**

```
public static JPopupMenu createPopupMenu(MyAction actions[])
```

Create a popup menu with the actions given in parameter actions.

- **createToolbar**

```
public static JToolBar createToolbar(MyAction actions[])
```

Create a tool bar with the actions given in parameter actions.

- **createRadioToolbar**

```
public static JToolBar createRadioToolbar(MyAction actions[])
```

Create a tool bar with radio buttons with the actions given in parameter actions.

- **getResourceString**

```
public static String getResourceString(String nm)
```

-

```
static void ()
```

11.33 Class view.Mode

```
java.lang.Object
   |
   +----view.Mode
```

---

public class **Mode**
extends Object

At any given time the PackageWindow is in a certain mode. This can be the move mode that allows the user to move, resize or select elements or it can be the addClass, addPackage or addAssociation that allows the user to add a class, package or association to the PackageView. Mode is the class that stores the current mode the PackageWindow is in.

Variable Index

- **actions**
  The actions of the mode menu and the mode toolbar.
- **ADD_ASSOCIATION_MODE**
  The mode that allows the user to add a new association
- **ADD_ASSOCIATIONROLE_MODE**
  The mode that allows the user to add a new role to the model
- **ADD_CLASS_MODE**
  The mode that allows the user to add a new class to the model
- **ADD_GENERALIZATION_MODE**
  The mode that allows the user to add a new generalization
- **ADD_PACKAGE_MODE**
  The mode that allows the user to add a new package to the model
- **currentMode**
  The mode the PackageWindow is currently in.
- **menu**
  The mode menu
- **MOVE_MODE**
  The mode that allows the user to move, resize or select elements
- **toolbar**
  The mode toolbar

Constructor Index

- **Mode**(PackageView)

Method Index

- **createMenu**()
  Create a mode menu and return it.
- **createToolbar**()
  Create a mode toolbar and return it.
- **getCurrentMode**()
  Return the currently active mode.
- **getCurrentMode**(Component)
  Return the currently active mode.
- **isAddAssociation**()
  Return whether the current mode is the add association mode.
- **isAddAssociationRole**()
  Return whether the current mode is the add association role mode.
- **isAddClass**()
  Return whether the current mode is the add class mode.
- **isAddPackage**()
  Return whether the current mode is the add package mode.
- **isMove**()
  Return whether the current mode is the move mode.
- **setCurrentMode**(Component, int)
  Change the current mode to a new mode.
- **setCurrentMode**(int)
  Change the current mode to a new mode.

Variables

MOVE_MODE

```
public static final int MOVE_MODE
```

The mode that allows the user to move, resize or select elements

ADD_CLASS_MODE

```
public static final int ADD_CLASS_MODE
```

The mode that allows the user to add a new class to the model

ADD_PACKAGE_MODE

```
public static final int ADD_PACKAGE_MODE
```

The mode that allows the user to add a new package to the model

- **ADD_ASSOCIATION_MODE**

```
public static final int ADD_ASSOCIATION_MODE
```

The mode that allows the user to add a new association

- **ADD_ASSOCIATIONROLE_MODE**

```
public static final int ADD_ASSOCIATIONROLE_MODE
```

The mode that allows the user to add a new role to the model

- **ADD_GENERALIZATION_MODE**

```
public static final int ADD_GENERALIZATION_MODE
```

The mode that allows the user to add a new generalization

- **actions**

```
private MyAction actions[]
```

The actions of the mode menu and the mode toolbar.

- **currentMode**

```
private int currentMode
```

The mode the PackageWindow is currently in.

- **menu**

```
private JMenu menu
```

The mode menu

- **toolbar**

```
private JToolBar toolbar
```

The mode toolbar

Constructors

- **Mode**

```
public Mode(PackageView packageView)
```

Methods

- **createMenu**

```
public JMenu createMenu()
```

Create a mode menu and return it.

createToolbar

```
public JToolBar createToolbar()
```

Create a mode toolbar and return it.

setCurrentMode

```
public void setCurrentMode(int mode)
```

Change the current mode to a new mode.

Parameters:
mode - the new mode

setCurrentMode

```
public static void setCurrentMode(Component c,
                                  int mode)
```

Change the current mode to a new mode.

getCurrentMode

```
public int getCurrentMode()
```

Return the currently active mode.

getCurrentMode

```
public static int getCurrentMode(Component c)
```

Return the currently active mode.

isMove

```
public boolean isMove()
```

Return whether the current mode is the move mode.

isAddClass

```
public boolean isAddClass()
```

Return whether the current mode is the add class mode.

isAddPackage

```
public boolean isAddPackage()
```

Return whether the current mode is the add package mode.

isAddAssociation

```
public boolean isAddAssociation()
```

Return whether the current mode is the add association mode.

isAddAssociationRole

```
public boolean isAddAssociationRole()
```

Return whether the current mode is the add association role mode.

11.34 Class view.ModelTree

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----com.sun.java.swing.JPanel
                                   |
                                   +----view.ModelTree
```

---

public class **ModelTree**
extends JPanel
implements ModelChangeListener

This widget shows a package model and all it contents as a tree. The user can use this tree to manipulate the model.

Variable Index

- **associationIcon**
  An icon that represents a association
- **associationRoleIcon**
  An icon that represents a association role
- **classIcon**
  An icon that represents a class
- **modelToNodeMap**
  A hashtable that maps a model to a node
- **packageIcon**
  An icon that represents a package
- **tree**
  The actual tree
- **treeModel**
  The model of the tree

Constructor Index

- **ModelTree**(PackageModel)
  Create a tree using the model packageModel

Method Index

- **findFrame**()
- **valueAdded**(ModelChangeEvent)
  This method is called when something is added to a model.
- **valueChanged**(ModelChangeEvent)
  This method is called when some value of a model has changed.
- **valueRemoved**(ModelChangeEvent)
  This method is called when something is removed from a model.

Variables

tree

```
private JTree tree
```

The actual tree

treeModel

```
private DefaultTreeModel treeModel
```

The model of the tree

modelToNodeMap

```
private Hashtable modelToNodeMap
```

A hashtable that maps a model to a node

classIcon

```
private static ImageIcon classIcon
```

An icon that represents a class

packageIcon

```
private static ImageIcon packageIcon
```

An icon that represents a package

associationIcon

```
private static ImageIcon associationIcon
```

An icon that represents a association

associationRoleIcon

```
private static ImageIcon associationRoleIcon
```

An icon that represents a association role

Constructors

ModelTree

```
public ModelTree(PackageModel packageModel)
```

Create a tree using the model packageModel

Methods

valueAdded

```
public void valueAdded(ModelChangeEvent e)
```

This method is called when something is added to a model. This method adds a node to the tree to represent the new addition to the model.

valueRemoved

```
public void valueRemoved(ModelChangeEvent e)
```

This method is called when something is removed from a model. This method will remove the corresponding node from the tree.

valueChanged

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when some value of a model has changed. This method will reload that part of the tree so that the tree show the up to date value.

findFrame

```
protected Frame findFrame()
```

11.35 Class view.Movable

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----view.Movable
```

---

public class **Movable**
extends JComponent
implements MouseListener, MouseMotionListener

The class Movable contains the functionality that allows the user to move a component on the screen to a new location. The user can use the mouse to drag a component (that is a subclass of Movable) and this class will allow it to move.

Variable Index

- **locationName**
  The name under which the location of the Movable will be stored in the model.
- **modelBase**
  The model in which the location of this Movable will be stored.
- **mouseDownPoint**
  The point at which the user first pressed a mouse button.
- **outlinePolygon**
  A polygon with the shape of the outline of this Movable

Constructor Index

- **Movable**(ModelBase, String)
  The constructor of the Movable

Method Index

- **addNotify**()
- **doMove**(int, int)
  This method is called to move the component to a new location.
- **getPolygon**()
  Return the outline polygon

- **mouseClicked**(MouseEvent)
  This method is public as an implementation detail and should not be called by a programmer.
- **mouseDragged**(MouseEvent)
  This method is public as an implementation detail and should not be called by a programmer.
- **mouseEntered**(MouseEvent)
  This method is public as an implementation detail and should not be called by a programmer.
- **mouseExited**(MouseEvent)
  This method is public as an implementation detail and should not be called by a programmer.
- **mouseMoved**(MouseEvent)
  This method is public as an implementation detail and should not be called by a programmer.
- **mousePressed**(MouseEvent)
  This method is called when the user presses a mouse button.
- **mouseReleased**(MouseEvent)
  This method is called when the user releases a mouse button.
- **raise**()
  Bring this movable to the front.
- **setLocation**(Point)

Variables

mouseDownPoint

```
protected Point mouseDownPoint
```

The point at which the user first pressed a mouse button.

modelBase

```
protected ModelBase modelBase
```

The model in which the location of this Movable will be stored.

locationName

```
protected String locationName
```

The name under which the location of the Movable will be stored in the model.

outlinePolygon

```
protected Polygon outlinePolygon
```

A polygon with the shape of the outline of this Movable

Constructors

Movable

```
public Movable(ModelBase modelBase,
               String locationName)
```

The constructor of the Movable

Parameters:

modelBase - the model in which the location of the Movable will be stored
locationName - the name under which the location of the Movable will be stored in the model.

Methods

addNotify

```
public void addNotify()
```

Overrides:

addNotify in class JComponent

raise

```
public void raise()
```

Bring this movable to the front.

mousePressed

```
public void mousePressed(MouseEvent e)
```

This method is called when the user presses a mouse button. This movable is raised to the front and this method registers the point at which the button was pressed. This method is public as an implementation detail and should not be called by a programmer.

mouseReleased

```
public void mouseReleased(MouseEvent e)
```

This method is called when the user releases a mouse button. This method is public as an implementation detail and should not be called by a programmer.

mouseDragged

```
public void mouseDragged(MouseEvent e)
```

This method is public as an implementation detail and should not be called by a programmer.

- **mouseClicked**

```
public void mouseClicked(MouseEvent e)
```

This method is public as an implementation detail and should not be called by a programmer.

- **mouseEntered**

```
public void mouseEntered(MouseEvent e)
```

This method is public as an implementation detail and should not be called by a programmer.

- **mouseExited**

```
public void mouseExited(MouseEvent e)
```

This method is public as an implementation detail and should not be called by a programmer.

- **mouseMoved**

```
public void mouseMoved(MouseEvent e)
```

This method is public as an implementation detail and should not be called by a programmer.

- **doMove**

```
protected void doMove(int deltaX,
                      int deltaY)
```

This method is called to move the component to a new location.

Parameters:

deltaX - the amount to move in the X direction
deltaY - the amount to move in the Y direction

- **setLocation**

```
public void setLocation(Point newLocation)
```

Overrides:

setLocation in class Component

- **getPolygon**

```
public Polygon getPolygon()
```

Return the outline polygon

11.36 Class view.MyAction

```
java.lang.Object
   |
   +----com.sun.java.swing.AbstractAction
           |
           +----view.MyAction
```

---

public abstract class **MyAction**
extends AbstractAction

This is an abstraction of the action that results from a menu selection or a button click on a toolbar.

Variable Index

- **accelerator**
  The key the user key use to activate the action without using the menu.
- **label**
  The label that will be shown in a menu.
- **mnemonic**
  One character of the label in the menu that is underlined.
- **resources**
- **tooltip**
  The tooltip that will be shown when the mouse is over a button in the toolbar.

Constructor Index

- **MyAction**(String)

Method Index

- ()
- **getAccelerator**()
  Return the accelerator of the action.
- **getIcon**()
  Return the icon of the action.
- **getLabel**()
  Return the label of the action.
- **getMnemonic**()
  Return the mnemonic of the action.

- **getName**()
  Return the name of the action.
- **getResource**(String)
  Load the resource (image, ...) with name s from the resource file or return null if it was not found.
- **getResourceString**(String)
  Load string with the name s from the resource file or return null if it was not found.
- **getTooltip**()
  Return the tooltip of the action.
- **loadInfo**()
  Load the information for this action from the resource file.
- **setLocale**(Locale)
  Change to a different locale.

Variables

label

```
private String label
```

The label that will be shown in a menu.

tooltip

```
private String tooltip
```

The tooltip that will be shown when the mouse is over a button in the toolbar.

mnemonic

```
private char mnemonic
```

One character of the label in the menu that is underlined. If the user presses this character on the keyboard while the menu if open the action is activated.

accelerator

```
private char accelerator
```

The key the user key use to activate the action without using the menu. The value in this variable is combined with the Control key to create the actual accelerator.

resources

```
private static ResourceBundle resources
```

Constructors

MyAction

```
public MyAction(String name)
```

Methods

- **loadInfo**

```
public void loadInfo()
```

Load the information for this action from the resource file.

- **getName**

```
public String getName()
```

Return the name of the action.

- **getLabel**

```
public String getLabel()
```

Return the label of the action.

- **getTooltip**

```
public String getTooltip()
```

Return the tooltip of the action.

- **getMnemonic**

```
public char getMnemonic()
```

Return the mnemonic of the action.

- **getAccelerator**

```
public char getAccelerator()
```

Return the accelerator of the action.

- **getIcon**

```
public Icon getIcon()
```

Return the icon of the action.

- **getResourceString**

```
private String getResourceString(String s)
```

Load string with the name s from the resource file or return null if it was not found.

getResource

```
private URL getResource(String key)
```

Load the resource (image, ...) with name s from the resource file or return null if it was not found.

setLocale

```
public static void setLocale(Locale locale)
```

Change to a different locale. The caller is responsible to call loadInfo() for each action that should be changed to the new locale.

```
static void ()
```

11.37 Class view.OperationDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----com.sun.java.swing.JDialog
                                           |
                                           +----view.OperationDialog
```

---

public class **OperationDialog**
extends JDialog
implements ActionListener, DocumentListener, ModelChangeListener

This is a dialog box that allows the user to edit an operation.

Variable Index

- **abstractOperation**
  A checkbox that show whether the operation is abstract or not.
- **classMember**
  A checkbox that shows whether the attribute is a normal member or a class member (static)
- **closeButton**
  The close button, when the user click on it, the dialog disappears
- **model**
  The model of which this is a view
- **name**
  A text-field that shows the name of the operation
- **returnType**
  A text-field that shows the return type of the operation
- **tm**
- **visibility**
  A combobox that shows the visibility of the attribute

Constructor Index

- **OperationDialog**(Frame, OperationModel)
  Constructor; creates the dialog box for OperationModel model

Method Index

- **actionPerformed**(ActionEvent)
  This method is called when the user has pushed a button or pressed the enter-key in a text field.
- **changedUpdate**(DocumentEvent)
  This method is called when the user has changed something in a text-field that is neither an insert nor a remove.
- **handleDocumentEvent**(DocumentEvent)
  This method is called when the value in the name, type or initial value text-fields has changed.
- **insertUpdate**(DocumentEvent)
  This method is called when the user has entered something in a text-field.
- **removeUpdate**(DocumentEvent)
  This method is called when the user has removed something from a text-field.
- **valueAdded**(ModelChangeEvent)
- **valueChanged**(ModelChangeEvent)
  This method is called when a value in the model has changed.
- **valueRemoved**(ModelChangeEvent)

Variables

model

```
private OperationModel model
```

The model of which this is a view

name

```
private JTextField name
```

A text-field that shows the name of the operation

returnType

```
private JTextField returnType
```

A text-field that shows the return type of the operation

visibility

```
private JComboBox visibility
```

A combobox that shows the visibility of the attribute

- **classMember**

```
private JCheckBox classMember
```

A checkbox that shows whether the attribute is a normal member or a class member (static)

- **abstractOperation**

```
private JCheckBox abstractOperation
```

A checkbox that show whether the operation is abstract or not.

- **closeButton**

```
private JButton closeButton
```

The close button, when the user click on it, the dialog disappears

- **tm**

```
private ParameterTableModel tm
```

Constructors

- **OperationDialog**

```
public OperationDialog(Frame parent,
                       OperationModel model)
```

Constructor; creates the dialog box for OperationModel model

Parameters:

parent - the window this dialog belongs to.
model - the model of which this is a view.

Methods

- **actionPerformed**

```
public void actionPerformed(ActionEvent e)
```

This method is called when the user has pushed a button or pressed the enter-key in a text field. This method is public as an implementation detail and should not be called by a programmer.

- **insertUpdate**

```
public void insertUpdate(DocumentEvent e)
```

This method is called when the user has entered something in a text-field. This method is public as an implementation detail and should not be called by a programmer.

- **removeUpdate**

```
public void removeUpdate(DocumentEvent e)
```

This method is called when the user has removed something from a text-field. This method is public as an implementation detail and should not be called by a programmer.

- **changedUpdate**

```
public void changedUpdate(DocumentEvent e)
```

This method is called when the user has changed something in a text-field that is neither an insert nor a remove. This method is public as an implementation detail and should not be called by a programmer.

- **handleDocumentEvent**

```
private void handleDocumentEvent(DocumentEvent e)
```

This method is called when the value in the name, type or initial value text-fields has changed. This method will call the setName or setReturnType methods of the model to propagate the change.

- **valueAdded**

```
public void valueAdded(ModelChangeEvent e)
```

- **valueRemoved**

```
public void valueRemoved(ModelChangeEvent e)
```

- **valueChanged**

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when a value in the model has changed. This method will update the values in the text-fields. This method is public as an implementation detail and should not be called by a programmer.

11.38 Class view.OperationView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----com.sun.java.swing.JLabel
                                   |
                                   +----view.OperationView
```

---

public class **OperationView**
extends JLabel
implements ModelChangeListener

This class shows one operation on the screen. The operation is shown as a trafic light icon showing the visibility, followed by the name, parameters and optional return type of the operation.

Variable Index

- **model**
  The model of which this class is a view

Constructor Index

- **OperationView**(OperationModel)
  Constructor; create an OperationView for OperationModel model

Method Index

- **deselected**()
  This method is called when the attribute is deselected.
- **getModel**()
  Return the model of which this class is a view.
- **selected**()
  This method is called when the attribute is selected.
- **valueAdded**(ModelChangeEvent)
  This method is called when a value was added to the model.

- **valueChanged**(ModelChangeEvent)
  This method is called when a value in the model has changed.
- **valueRemoved**(ModelChangeEvent)
  This method is called when a value was removed from the model.

Variables

model

```
private OperationModel model
```

The model of which this class is a view

Constructors

OperationView

```
public OperationView(OperationModel model)
```

Constructor; create an OperationView for OperationModel model

Methods

valueChanged

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when a value in the model has changed. This method will update the label and the icon. This method is public as an implementation detail and should not be called by a programmer.

valueAdded

```
public void valueAdded(ModelChangeEvent e)
```

This method is called when a value was added to the model. This method will update the label. This method is public as an implementation detail and should not be called by a programmer.

valueRemoved

```
public void valueRemoved(ModelChangeEvent e)
```

This method is called when a value was removed from the model. This method will update the label. This method is public as an implementation detail and should not be called by a programmer.

selected

```
public void selected()
```

This method is called when the attribute is selected.

deselected

```
public void deselected()
```

This method is called when the attribute is deselected.

getModel

```
public OperationModel getModel()
```

Return the model of which this class is a view.

11.39 Class view.OperationsView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----com.sun.java.swing.JPanel
                                   |
                                   +----view.OperationsView
```

---

public class **OperationsView**
extends JPanel
implements ModelChangeListener

This class show a list of operations; all the operations of a class.

Variable Index

- **box**
- **model**
  The class model that owns the operations that are shown by thic class
- **modelToViewMap**
  A hashtable that maps and attribute model to a view
- **mouseListener**
  if this is not null than all mouse events that occur on the attribute views will be send to this listener.

Constructor Index

- **OperationsView**(ClassModel, MouseListener)
  The OperationsView constructor; show a list of the operations of ClassModel model.

Method Index

- **addOperation**(OperationModel)
  Add a new operation to the list.
- **removeOperation**(OperationModel)
  Remove an operation from the list.

- **valueAdded**(ModelChangeEvent)
  This method is called when an operation was added to the model.
- **valueChanged**(ModelChangeEvent)
- **valueRemoved**(ModelChangeEvent)
  This method is called when an operation was removed from the model.

Variables

- **model**

```
private ClassModel model
```

  The class model that owns the operations that are shown by thic class

- **modelToViewMap**

```
private Hashtable modelToViewMap
```

  A hashtable that maps and attribute model to a view

- **box**

```
private Box box
```

- **mouseListener**

```
private MouseListener mouseListener
```

  if this is not null than all mouse events that occur on the attribute views will be send to this listener.

Constructors

- **OperationsView**

```
public OperationsView(ClassModel model,
                      MouseListener mouseListener)
```

  The OperationsView constructor; show a list of the operations of ClassModel model.

  Parameters:
  model - the classmodel that owns the operations that are to be shown by this class.
  mouseListener - if this is not null than all mouse events that occur on the operation views will be send to this listener.

Methods

- **addOperation**

```
private void addOperation(OperationModel oper)
```

Add a new operation to the list.

removeOperation

```
private void removeOperation(OperationModel oper)
```

Remove an operation from the list.

valueAdded

```
public void valueAdded(ModelChangeEvent e)
```

This method is called when an operation was added to the model. This method is public as an implementation detail and should not be called by a programmer.

valueRemoved

```
public void valueRemoved(ModelChangeEvent e)
```

This method is called when an operation was removed from the model. This method is public as an implementation detail and should not be called by a programmer.

valueChanged

```
public void valueChanged(ModelChangeEvent e)
```

11.40 Class view.PackageDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----com.sun.java.swing.JDialog
                                           |
                                           +----view.PackageDialog
```

---

public class **PackageDialog**
extends JDialog
implements ActionListener, DocumentListener, ModelChangeListener

This is a dialog box that allows the user to edit a PackageDialog

---

Variable Index

- **closeButton**
  The close button, when the user click on it, the dialog disappears
- **model**
  The package model can be edited by this dialog.
- **name**
  The text-field that shows the name of the class
- **stereotype**
  The text-field that shows the stereotype of the class
- **visibility**
  The combobox that shows the visibility of the class

Constructor Index

- **PackageDialog**(Frame, PackageModel)
  Constructor; creates the dialog box for PackageModel model.

Method Index

- **actionPerformed**(ActionEvent)
  This method is called when the user has pushed a button or pressed the enter-key in a text field.
- **changedUpdate**(DocumentEvent)
  This method is called when the user has changed something in a text-field that is neither an insert nor a remove.
- **handleDocumentEvent**(DocumentEvent)
  This method is called when the value in the name or stereotype text-fields has changed.
- **insertUpdate**(DocumentEvent)
  This method is called when the user has entered something in a text-field.
- **removeUpdate**(DocumentEvent)
  This method is called when the user has removed something from a text-field.
- **valueAdded**(ModelChangeEvent)
- **valueChanged**(ModelChangeEvent)
  This method is called when a value in the model has changed.
- **valueRemoved**(ModelChangeEvent)

Variables

model

```
private PackageModel model
```

The package model can be edited by this dialog.

name

```
private JTextField name
```

The text-field that shows the name of the class

stereotype

```
private JTextField stereotype
```

The text-field that shows the stereotype of the class

visibility

```
private JComboBox visibility
```

The combobox that shows the visibility of the class

closeButton

```
private JButton closeButton
```

The close button, when the user click on it, the dialog disappears

Constructors

PackageDialog

```
public PackageDialog(Frame frame,
                     PackageModel model)
```

Constructor; creates the dialog box for PackageModel model.

Parameters:

frame - the window this dialog belongs to.
model - the model of which this is a view

Methods

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

This method is called when the user has pushed a button or pressed the enter-key in a text field. This method is public as an implementation detail and should not be called by a programmer.

insertUpdate

```
public void insertUpdate(DocumentEvent e)
```

This method is called when the user has entered something in a text-field. This method is public as an implementation detail and should not be called by a programmer.

removeUpdate

```
public void removeUpdate(DocumentEvent e)
```

This method is called when the user has removed something from a text-field. This method is public as an implementation detail and should not be called by a programmer.

changedUpdate

```
public void changedUpdate(DocumentEvent e)
```

This method is called when the user has changed something in a text-field that is neither an insert nor a remove. This method is public as an implementation detail and should not be called by a programmer.

handleDocumentEvent

```
private void handleDocumentEvent(DocumentEvent e)
```

This method is called when the value in the name or stereotype text-fields has changed. This method will call the setName or setStereotype methods of the model to propagate the change.

valueChanged

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when a value in the model has changed. This method will update the values in the text-fields. This method is public as an implementation detail and should not be called by a programmer.

valueAdded

```
public void valueAdded(ModelChangeEvent e)
```

valueRemoved

```
public void valueRemoved(ModelChangeEvent e)
```

11.41 Class view.PackageLayout

```
java.lang.Object
   |
   +----view.PackageLayout
```

---

class **PackageLayout**
extends Object
implements LayoutManager

This is a layout manager that is used to layout the package views.

This class is not public and therefore cannot be used outside this package.

11.42 Class view.PackageView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----view.Movable
                                   |
                                   +----view.Resizable
                                           |
                                           +----view.ElementView
                                                   |
                                                   +----view.PackageView
```

---

public class **PackageView**
extends ElementView

This class shows one package element on the screen. At the top of the package there are the stereotype and name labels and the visibility icon. The visibility icon uses the colors of a traffic light to show public visibility (green), protected visibility (yellow) and private visibility (red). Below the labels and icon the content of the package is shown.

Variable Index

- **elementLayer**
  The layer in which the elements are shown
- **model**
  The package model that this class is a view of
- **modelToViewMap**
  This hashtable that maps element models to their views.
- **packagePopup**
  The menu that pops up when the user presses the right mouse button on the package view
- **relationLayer**
  The layer in which the relation are shown, this layer lies on top of the element layer.
- **topLevel**
  Is this a top level package (direct child of a PackageWindow).

Constructor Index

- **PackageView**(PackageModel, PackageView, boolean)
  Constructor; create a view of the package model.

Method Index

- **addAssociation**(AssociationModel)
  This method is called in response to an event to add a new association to the package.
- **addElement**(ElementModel)
  This method is called in response to an event to add a new element to the package.
- **addNewAssociation**(int, int)
  This method is called when the user wants to adds a new association to the package.
- **addNewClass**(int, int)
  This method is called when the user wants to adds a new class to the package.
- **addNewPackage**(int, int)
  This method is called when the user wants to adds a new package to the package.
- **doMove**(int, int)
  This method is called to move the component to a new location.
- **doResize**(int, int)
  This method is called to resize the component.
- **findClassView**(ClassModel)
- **findComponentAt**(int, int)
- **getElementLayer**()
  Return the layer that contains the elements.
- **getModel**()
  Return the model of this view.
- **getRelationLayer**()
  Return the layer that contains the relations.
- **mouseClicked**(MouseEvent)
  This method is called when the user clicks a mouse button.
- **mousePressed**(MouseEvent)
  This method is called when the user presses a mouse button.
- **removeAssociation**(AssociationModel)
  This method is called in response to an event to remove an association from the package.
- **removeElement**(ElementModel)
  This method is called in response to an event to remove an element from the package.
- **valueAdded**(ModelChangeEvent)
  This method is called when an element or association is added to the package model.
- **valueRemoved**(ModelChangeEvent)
  This method is called when an element or association is removed from the package model.

Variables

- **model**

```
private PackageModel model
```

The package model that this class is a view of

- **packagePopup**

```
private JPopupMenu packagePopup
```

The menu that pops up when the user presses the right mouse button on the package view

- **elementLayer**

```
private JPanel elementLayer
```

The layer in which the elements are shown

- **relationLayer**

```
private JPanel relationLayer
```

The layer in which the relation are shown, this layer lies on top of the element layer.

- **modelToViewMap**

```
private Hashtable modelToViewMap
```

This hashtable that maps element models to their views.

- **topLevel**

```
private boolean topLevel
```

Is this a top level package (direct child of a PackageWindow).

Constructors

- **PackageView**

```
public PackageView(PackageModel model,
                   PackageView parentPackage,
                   boolean topLevel)
```

Constructor; create a view of the package model.

Parameters:

model - the model that will be shown with this view
parentPackage - the package that owns this class
topLevel - whether this is a top level package; that is, a direct child of a PackageWindow.

Methods

- **getModel**

```
public PackageModel getModel()
```

Return the model of this view.

- **getElementLayer**

```
public JPanel getElementLayer()
```

Return the layer that contains the elements.

- **getRelationLayer**

```
public JPanel getRelationLayer()
```

Return the layer that contains the relations.

- **mousePressed**

```
public void mousePressed(MouseEvent e)
```

This method is called when the user presses a mouse button. If the mouse button is the right mouse button, the popup menu is shown. If the mouse button is the left mouse button and the current mode is ADD_CLASS_MODE, ADD_PACKAGE_MODE, or ADD_ASSOCIATION_MODE a new class/package/association is created.

Overrides:
mousePressed in class ElementView

- **mouseClicked**

```
public void mouseClicked(MouseEvent e)
```

This method is called when the user clicks a mouse button. If the user double-clicks, then a dialog box is shown allowing the user to edit the package. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
mouseClicked in class Movable

- **doMove**

```
public void doMove(int deltaX,
                   int deltaY)
```

This method is called to move the component to a new location.

Overrides:
doMove in class Movable

doResize

```
protected void doResize(int x,
                        int y)
```

This method is called to resize the component.

Overrides:
doResize in class ElementView

valueAdded

```
public void valueAdded(ModelChangeEvent event)
```

This method is called when an element or association is added to the package model. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
valueAdded in class ElementView

valueRemoved

```
public void valueRemoved(ModelChangeEvent event)
```

This method is called when an element or association is removed from the package model. This method is public as an implementation detail and should not be called by a programmer.

Overrides:
valueRemoved in class ElementView

addElement

```
private void addElement(ElementModel e)
```

This method is called in response to an event to add a new element to the package.

removeElement

```
private void removeElement(ElementModel model)
```

This method is called in response to an event to remove an element from the package.

addAssociation

```
private void addAssociation(AssociationModel associationModel)
```

This method is called in response to an event to add a new association to the package.

- **removeAssociation**

```
private void removeAssociation(AssociationModel model)
```

This method is called in response to an event to remove an association from the package.

- **addNewClass**

```
private void addNewClass(int x,
                         int y)
```

This method is called when the user wants to adds a new class to the package.

- **addNewPackage**

```
private void addNewPackage(int x,
                           int y)
```

This method is called when the user wants to adds a new package to the package.

- **addNewAssociation**

```
private void addNewAssociation(int x,
                               int y)
```

This method is called when the user wants to adds a new association to the package.

- **findComponentAt**

```
public Component findComponentAt(int x,
                                 int y)
```

- **findClassView**

```
public ClassView findClassView(ClassModel model)
```

11.43 Class view.PackageWindow

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
                                   |
                                   +----com.sun.java.swing.JFrame
                                           |
                                           +----view.PackageWindow
```

---

public class **PackageWindow**
extends JFrame
implements ModelChangeListener

The package window is a window that shows one package and all it contents. It has a menubar and toolbar at the top, at ModelTree at the left and a PackageView at the right.

See Also:
ModelTree, PackageView, Mode, Selection

Variable Index

- **mode**
- **model**
- **modelTree**
- **selection**

Constructor Index

- **PackageWindow**(PackageModel)

Method Index

- **getMode**()
  Return the Mode object that keeps track of the current mode.
- **getModelTree**()
  Return the model tree

- **getSelection**()
  Return the selection of this window
- **valueAdded**(ModelChangeEvent)
- **valueChanged**(ModelChangeEvent)
  This method is called when the name, stereotype or owner of the package changes.
- **valueRemoved**(ModelChangeEvent)

Variables

model

```
private PackageModel model
```

selection

```
private Selection selection
```

mode

```
private Mode mode
```

modelTree

```
private ModelTree modelTree
```

Constructors

PackageWindow

```
public PackageWindow(PackageModel model)
```

Methods

getSelection

```
public Selection getSelection()
```

Return the selection of this window

getMode

```
public Mode getMode()
```

Return the Mode object that keeps track of the current mode.

getModelTree

```
public ModelTree getModelTree()
```

Return the model tree

valueAdded

```
public void valueAdded(ModelChangeEvent event)
```

valueRemoved

```
public void valueRemoved(ModelChangeEvent event)
```

valueChanged

```
public void valueChanged(ModelChangeEvent e)
```

This method is called when the name, stereotype or owner of the package changes.

11.44 Class view.ParameterTableModel

```
java.lang.Object
   |
   +----com.sun.java.swing.table.AbstractTableModel
           |
           +----view.ParameterTableModel
```

---

class **ParameterTableModel**
extends AbstractTableModel

This class is used to control the table that shows the parameters of an operation.

This class is not public and therefore cannot be used outside this package.

11.45 Class view.RelativeMovable

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----view.Movable
                                   |
                                   +----view.RelativeMovable
```

---

public class **RelativeMovable**
extends Movable
implements ComponentListener

The class Movable contains the functionality that allows the user to move a component on the screen to a new location. The user can use the mouse to drag a component (that is a subclass of Movable) and this class will allow it to move.

RelativeMovable inherits from Movable and allows a component on the screen to be moved relative to another component. For example, A is a Movable and B is a RelativeMovable that moves relative to A. The user can move B around and the RelativeMovable will keep track of the relative offset to A. If the user moves A around, B will move as well to keep the offset the same.

Variable Index

- **movable**
  The RelativeMovable moves relative to this Movable.
- **offsetX**
  The offset in the X direction
- **offsetY**
  The offset in the Y direction

Constructor Index

- **RelativeMovable**(Movable, ModelBase, String)
  The constructor the RelativeMovable

Method Index

- **addNotify**()
- **componentHidden**(ComponentEvent)
- **componentMoved**(ComponentEvent)
  This method is called when the Movable moves to a new location.
- **componentResized**(ComponentEvent)
  This method is called when the Movable changes size.
- **componentShown**(ComponentEvent)
- **doMove**(int, int)
  This method is called to move the component to a new location.
- **setLocation**(Point)
  Override the setLocation method of the superclass to update the offsetX and offsetY variables when the RelativeMovable moves.
- **updateLocation**()

Variables

movable

```
protected Movable movable
```

The RelativeMovable moves relative to this Movable.

offsetX

```
protected int offsetX
```

The offset in the X direction

offsetY

```
protected int offsetY
```

The offset in the Y direction

Constructors

RelativeMovable

```
public RelativeMovable(Movable movable,
                       ModelBase modelBase,
                       String locationName)
```

The constructor the RelativeMovable

Parameters:

movable - the RelativeMovable will move relative to this Movable.
modelBase - the model in which the location of the RelativeMovable will be stored
locationName - the name under which the location of the RelativeMovable will be stored in the model.

Methods

- **addNotify**

```
public void addNotify()
```

Overrides:
addNotify in class Movable

- **setLocation**

```
public void setLocation(Point newLocation)
```

Override the setLocation method of the superclass to update the offsetX and offsetY variables when the RelativeMovable moves.

Overrides:
setLocation in class Movable

- **doMove**

```
protected void doMove(int deltaX,
                      int deltaY)
```

This method is called to move the component to a new location.

Overrides:
doMove in class Movable

- **updateLocation**

```
protected void updateLocation()
```

- **componentResized**

```
public void componentResized(ComponentEvent e)
```

This method is called when the Movable changes size.

- **componentMoved**

```
public void componentMoved(ComponentEvent e)
```

This method is called when the Movable moves to a new location.

- **componentShown**

```
public void componentShown(ComponentEvent e)
```

- **componentHidden**

```
public void componentHidden(ComponentEvent e)
```

11.46 Class view.Resizable

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----view.Movable
                                   |
                                   +----view.Resizable
```

---

public class **Resizable**
extends Movable

The class Resizable contains the functionality that allows the user to resize a component. The user can use the mouse to drag a border of the component (that is a subclass of Resizable) and this class will allow it to change its size.

Variable Index

- **E_Cursor**
  The east resize cursor
- **mouseDownSide**
  The side of the component where the mouse was pressed down.
- **N_Cursor**
  The north resize cursor
- **NE_Cursor**
  The north east resize cursor
- **NW_Cursor**
  The north west resize cursor
- **S_Cursor**
  The south resize cursor
- **SE_Cursor**
  The south east resize cursor
- **SIDE_E**
  The east (right) side of the component
- **SIDE_N**
  The north (top) side of the component
- **SIDE_NE**
  The north east (top right) corner of the component
- **SIDE_NW**
  The north west (top left) corner of the component

- **SIDE_S**
  The south (bottom) side of the component
- **SIDE_SE**
  The south east (bottom right) corner of the component
- **SIDE_SW**
  The south west (bottom left) corner of the component
- **SIDE_W**
  The west (left) side of the component
- **sizeName**
  The name of the property under which the size of the component is stored in the model.
- **SW_Cursor**
  The south west resize cursor
- **W_Cursor**
  The west resize cursor

Constructor Index

- **Resizable**(ModelBase, String, String)

Method Index

- **doResize**(int, int)
  This method is called to resize the component.
- **getSide**(MouseEvent)
  Determin if the mouse pointer is currently over a corner or size of the component.
- **mouseDragged**(MouseEvent)
  This method is called when the user drags the mouse.
- **mouseMoved**(MouseEvent)
  This method is called when the user moves the mouse without pressing a mouse button.
- **mousePressed**(MouseEvent)
  This method is called when the user pressed a mouse button.
- **mouseReleased**(MouseEvent)
  This method is called when the user release the mouse button.

Variables

- **mouseDownSide**

```
protected int mouseDownSide
```

  The side of the component where the mouse was pressed down.

sizeName

```
protected String sizeName
```

The name of the property under which the size of the component is stored in the model.

SIDE_N

```
private static final int SIDE_N
```

The north (top) side of the component

SIDE_NE

```
private static final int SIDE_NE
```

The north east (top right) corner of the component

SIDE_E

```
private static final int SIDE_E
```

The east (right) side of the component

SIDE_SE

```
private static final int SIDE_SE
```

The south east (bottom right) corner of the component

SIDE_S

```
private static final int SIDE_S
```

The south (bottom) side of the component

SIDE_SW

```
private static final int SIDE_SW
```

The south west (bottom left) corner of the component

SIDE_W

```
private static final int SIDE_W
```

The west (left) side of the component

SIDE_NW

```
private static final int SIDE_NW
```

The north west (top left) corner of the component

- **NW_Cursor**

```
private static final Cursor NW_Cursor
```

The north west resize cursor

- **N_Cursor**

```
private static final Cursor N_Cursor
```

The north resize cursor

- **NE_Cursor**

```
private static final Cursor NE_Cursor
```

The north east resize cursor

- **E_Cursor**

```
private static final Cursor E_Cursor
```

The east resize cursor

- **SE_Cursor**

```
private static final Cursor SE_Cursor
```

The south east resize cursor

- **S_Cursor**

```
private static final Cursor S_Cursor
```

The south resize cursor

- **SW_Cursor**

```
private static final Cursor SW_Cursor
```

The south west resize cursor

- **W_Cursor**

```
private static final Cursor W_Cursor
```

The west resize cursor

Constructors

- **Resizable**

```
public Resizable(ModelBase modelBase,
                 String locationName,
                 String sizeName)
```

Methods

- **mousePressed**

```
public void mousePressed(MouseEvent e)
```

This method is called when the user pressed a mouse button. If the current mode is MOVE_MODE then the user may change the size of the component.

Overrides:
mousePressed in class Movable

- **mouseDragged**

```
public void mouseDragged(MouseEvent e)
```

This method is called when the user drags the mouse. If the user started the drag at a side of the component that call doResize() to resize the component.

Overrides:
mouseDragged in class Movable
See Also:
doResize

- **mouseReleased**

```
public void mouseReleased(MouseEvent e)
```

This method is called when the user release the mouse button.

Overrides:
mouseReleased in class Movable

- **mouseMoved**

```
public void mouseMoved(MouseEvent e)
```

This method is called when the user moves the mouse without pressing a mouse button. When the mouse pointer is positioned above a side or a corner of the component the mouse pointer changes shape to reflect the ability to resize the component.

Overrides:
mouseMoved in class Movable

- **doResize**

```
protected void doResize(int x,
                        int y)
```

This method is called to resize the component. The parameters x and y represent the increase or decrease in width/height.

getSide

```
protected int getSide(MouseEvent e)
```

Determin if the mouse pointer is currently over a corner or size of the component.

11.47 Class view.RolePanel

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----com.sun.java.swing.JComponent
                           |
                           +----com.sun.java.swing.JPanel
                                   |
                                   +----view.RolePanel
```

---

class **RolePanel**
extends JPanel
implements ActionListener, DocumentListener, ModelChangeListener

class RolePanel is used to display a AssociationRoleModel in a AssociationDialog.

See Also:
AssociationRoleModel, AssociationDialog

This class is not public and therefore cannot be used outside this package.

11.48 Class view.Selection

```
java.lang.Object
   |
   +----view.Selection
```

---

public class **Selection**
extends Object

This class keeps track of the selected components within a PackageWindow. There are two types of methods: normal methods that set, add to or remove from the selection and static methods that first find a PackageWindow, request the selection of that PackageWindow and call the normal methods of that selection.

Variable Index

- **selection**
  The selected components

Constructor Index

- **Selection**()

Method Index

- **add**(Component)
  Add component c to the selection, the previous selection is not deselected.
- **addToSelection**(Component)
  Add component c to the selection, the previous selection is not deselected.
- **clear**(Component)
  Clear the selection, deselect all components in the current selection.
- **clearSelection**(Component)
  Clear the selection, deselect all components in the current selection.
- **findWindow**(Component)
  Find the PackagWindow that is the ancestor of component c
- **remove**(Component)
  Remove component c from the selection, the previous selection (except component c) is not deselected.
- **removeFromSelection**(Component)
  Remove component c from the selection, the previous selection (except component c) is not deselected.

- **set**(Component)
  Set the selection to component c, the previous selection is deselected
- **setSelection**(Component)
  Set the selection to component c, the previous selection is deselected

Variables

selection

```
private Vector selection
```

The selected components

Constructors

Selection

```
public Selection()
```

Methods

findWindow

```
private static PackageWindow findWindow(Component c)
```

Find the PackagWindow that is the ancestor of component c

setSelection

```
public static void setSelection(Component c)
```

Set the selection to component c, the previous selection is deselected

set

```
public void set(Component c)
```

Set the selection to component c, the previous selection is deselected

addToSelection

```
public static void addToSelection(Component c)
```

Add component c to the selection, the previous selection is not deselected.

add

```
public void add(Component c)
```

Add component c to the selection, the previous selection is not deselected.

removeFromSelection

```
public static void removeFromSelection(Component c)
```

Remove component c from the selection, the previous selection (except component c) is not deselected.

remove

```
public void remove(Component c)
```

Remove component c from the selection, the previous selection (except component c) is not deselected.

clearSelection

```
public static void clearSelection(Component c)
```

Clear the selection, deselect all components in the current selection.

clear

```
public void clear(Component c)
```

Clear the selection, deselect all components in the current selection.

11.49 Interface view.SelectionListener

public interface **SelectionListener**
extends EventListener

All componentes that can be selected by the user should implement this interface. When the user selects the component the method selected() is called. When the user selects another component the method deselected() is called.

Method Index

- **deselected**()
  This method is called when the user has selected another component.
- **selected**()
  This method is called when the user has selected the component.

Methods

selected

```
public abstract void selected()
```

This method is called when the user has selected the component.

deselected

```
public abstract void deselected()
```

This method is called when the user has selected another component.

11.50 Class Hierarchy

- class java.lang.Object
  - class com.sun.java.swing.AbstractAction (implements com.sun.java.swing.Action, java.lang.Cloneable, java.io.Serializable)
    - class view.MyAction
  - class com.sun.java.swing.table.AbstractTableModel (implements com.sun.java.swing.table.TableModel, java.io.Serializable)
    - class view.ParameterTableModel
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class com.sun.java.swing.JComponent (implements java.io.Serializable)
        - class com.sun.java.swing.JLabel (implements com.sun.java.swing.SwingConstants, com.sun.java.accessibility.Accessible)
          - class view.AttributeView (implements ModelChangeListener)
          - class view.OperationView (implements ModelChangeListener)
          - class Renderer (implements com.sun.java.swing.ListCellRenderer)
        - class com.sun.java.swing.JPanel (implements com.sun.java.accessibility.Accessible)
          - class view.AttributesView (implements ModelChangeListener)
          - class view.ModelTree (implements ModelChangeListener)
          - class view.OperationsView (implements ModelChangeListener)
          - class view.RolePanel (implements java.awt.event.ActionListener, com.sun.java.swing.event.DocumentListener, ModelChangeListener)
        - class view.LineSegment (implements java.awt.event.ComponentListener, com.sun.java.swing.event.ChangeListener)
        - class view.Movable (implements java.awt.event.MouseListener, java.awt.event.MouseMotionListener)
          - class view.RelativeMovable (implements java.awt.event.ComponentListener)
            - class view.BoundedMovable
              - class view.Handle
                - class view.AssociationRoleView (implements ModelChangeListener)
          - class view.Resizable
            - class view.AssociationView (implements ModelChangeListener)
            - class view.ElementView (implements

ModelChangeListener, view.SelectionListener)
- class view.ClassView
- class view.PackageView

- class java.awt.Panel
  - class java.applet.Applet
    - class com.sun.java.swing.JApplet (implements com.sun.java.accessibility.Accessible, com.sun.java.swing.RootPaneContainer)
      - class SoccaEditor
- class java.awt.Window
  - class java.awt.Dialog
    - class com.sun.java.swing.JDialog (implements com.sun.java.swing.WindowConstants, com.sun.java.accessibility.Accessible, com.sun.java.swing.RootPaneContainer)
      - class view.AssociationDialog (implements java.awt.event.ActionListener, com.sun.java.swing.event.DocumentListener, ModelChangeListener)
      - class view.AttributeDialog (implements java.awt.event.ActionListener, com.sun.java.swing.event.DocumentListener, ModelChangeListener)
      - class view.ClassDialog (implements java.awt.event.ActionListener, com.sun.java.swing.event.DocumentListener, java.awt.event.MouseListener, ModelChangeListener)
      - class view.OperationDialog (implements java.awt.event.ActionListener, com.sun.java.swing.event.DocumentListener, ModelChangeListener)
      - class view.PackageDialog (implements java.awt.event.ActionListener, com.sun.java.swing.event.DocumentListener, ModelChangeListener)
  - class java.awt.Frame (implements java.awt.MenuContainer)
    - class com.sun.java.swing.JFrame (implements com.sun.java.swing.WindowConstants, com.sun.java.accessibility.Accessible, com.sun.java.swing.RootPaneContainer)
      - class view.MainWindow (implements java.awt.event.ActionListener)
      - class view.PackageWindow (implements ModelChangeListener)
- class view.Line (implements java.awt.event.MouseListener,

java.awt.event.MouseMotionListener, view.SelectionListener)

- class java.util.EventObject (implements java.io.Serializable)
  - class ModelChangeEvent
- class view.MenuFactory
- class view.Mode
- class model.ModelBase (implements java.io.Serializable)
  - class model.AssociationModel
  - class model.AssociationRoleModel
  - class model.ElementModel
    - class model.ClassModel
    - class model.PackageModel
  - class model.GeneralizationModel
  - class model.MemberModel
    - class model.AttributeModel
    - class model.OperationModel
  - class model.NoteModel
  - class model.UsesRelationModel
- interface ModelChangeListener (extends java.util.EventListener)
- class view.PackageLayout (implements java.awt.LayoutManager)
- class model.ParameterModel (implements java.io.Serializable)
- class view.Selection
- interface view.SelectionListener (extends java.util.EventListener)
- class Visibility (implements java.io.Serializable)