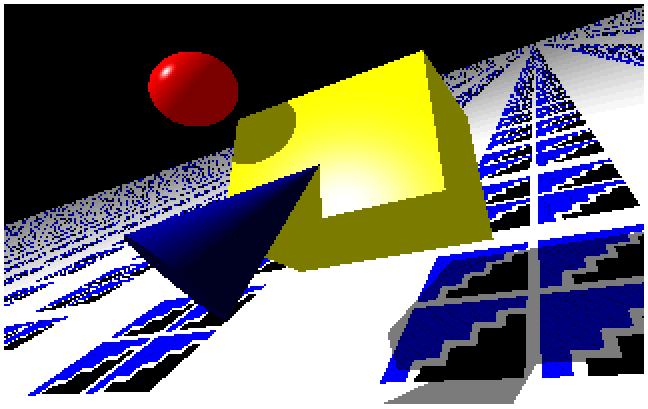
Space which is not occupied by any object in the design slate.



JAVelin

Graphical Object Oriented Development in JAVA

Getting Started

If you did not receive the printed "Getting Started" manual then we recommend that you make a hard copy of this tutorial for the purposes of having it by your computer to avoid the need to keep switching between it and Javelin. Alternatively this tutorial is sized so that it fits on the right side of your screen. It will remain on top of Javelin.

Tutorial or Quick Start?

The following tutorial takes you step by step through the major features of Javelin and includes a basic introduction to object oriented analysis and design and JAVA programming. For those who would rather learn by exploration, avoiding the tutorial, we recommend that you at least read the next few paragraphs and then read the sections in the online help named "Design Slate" and "Basic Class Operations". This will give you a basic understanding of how to drive the tool.

Developing our first JAVA application with Javelin

To explain how to use Javelin we'll take you step by step through the development process of the core classes of a trivial graphics application. Traditionally the phases of software development are called analysis, design, testing and implementation. In simple terms these describe an ordered and "proper" approach to fast, quality software development. You can read more about this in the on line help "Life Cycle" topic.

Analysis Phase

At some stage in the process of developing any program you must analyse what "things" your program has to represent. These things might be shapes, people, employees, accounts etc., If you're a good programmer you will tackle this analysis phase early. Many people write bad software because they try to avoid doing any thorough analysis and try to jump straight to the coding stage. This trap often catches many first time programmers so if you are new to programming you should take heed of this warning.

This tool makes the analysis phase much more palatable and results in more productive design and coding phases. The analysis process is inevitable and has to be performed by you. No computer in existence today can do this for you. However, in most cases this is a fairly simple task.

In our analysis we assume that the graphical user interface (GUI) classes are already in place. JAVA comes with a library called AWT which supplies a set of classes that provide windows, buttons, text controls etc., In our example we have already set up the code to produce the main window, the applet window, so the example is only concerned with drawing objects into that window.

The Example...

Take an imaginary, rather trivial, graphics application in which a user can have circles, boxes, or triangles painted on the screen. Before too many cups of coffee, or glasses of water for the non caffeinites, you should come to the (hopefully obvious) conclusion that your program will have classes in it which represent these "things" or objects. Traditionally you might draw on a piece of paper some little pictures representing your Box, Circle and Triangle classes. This piece of paper could be called a design slate. In this theme the main window in Javelin is called the design slate also. Paper design slates are hard to modify and can't do any nice "extras" for you. The Javelin, computer based, design slate changes all that.

As you stare at your design slate (Javelin or paper based) you realize that there is something in common with these shapes: they all must remember where they are currently located on the screen; and; they all must be able to paint themselves if told to do so. You soon realize that in an OO world the best way to do this is to have a super class representing what is common among these shapes (let's call it Shape) and three classes which extend the basic shape behaviour to represent the specific shapes (call them Circle, Box, Triangle).

Let's start at the very beginning, a very good place to start: Step 1!

If you already have Javelin running then close it down and double click on the Javelin icon marked Step 1 to open the Step 1 example This should open up an example project which has the user interface classes already added for you. As we do not compile step 1 (that comes later in step 3) we have not shipped the source code for it. To prevent Javelin from warning you about missing underlying source code you should generate the code now by selecting Generate|All Code. This will generate the code for Step 1. Step 1 contains text which explains the various components of the objects on design slate. For more detailed explanations you should open up Quick Feature Summaries 1 and 2 from the start menu.

Intuitive User Interface - Smart Cursor

If you ever wonder how to perform a particular operation on an object you should be aware that most objects respond to the left and/or right mouse button. The left mouse button is used to initiate moves, drag/drop operations and selections; the right mouse button typically brings up an object's menu. Most of the actions can be performed directly on the objects themselves avoiding excessive use the menus or tool bar.

The cursor changes when it moves over objects: A cross on the left side of the cursor shows that a move is possible using the left mouse button. A menu on the right side of the cursor shows that a popup menu can be brought up by clicking the right mouse button. Have a play with the Step 1 example now.

Step 2 - The adventure continues

Open the Step 2 example which should contain a single class - the Graphic Window. We have removed the other classes for this example. We will now add some classes of our own.

1. Press the right mouse button on empty space and select Add Class.
2. A dialog box will appear asking you for the new class' details.
3. Type Circle as the class' name. You can learn about other options in the on line help system under "Class Details" but for now retain the default values.
4. Type in a description of how the class 'Circle' will represent a Circle drawn in our application. A good description here will stand you in good stead when it comes to the design phase.

5. The development phase field is automatically set to a picture of a light bulb to show you that you are at the Analysis (or ideas) Phase.
6. Press Ok to close the dialog.

Adjusting the View

Now your new class has been added but we need more space on the design slate. We will now demonstrate the flexible scaling features of Javelin.

1. Select View and choose Entire Slate.
2. The entire slate is now in view with a small section covered with our classes.
3. Drag a rectangle around the classes by pressing and holding the left mouse button down at the left and top of the group of classes and dragging to the right and bottom of the classes, leaving a reasonable clearance from the classes themselves.

Javelin will rescale the design slate so that the region we have covered with the rectangle will be magnified as much as possible. As you can see Javelin runs extremely fast even on low end PCs.

Moving A Class

To move a class just press and hold the left mouse button on its bottom section and drag it to the desired location. See the on line help topic Basic Class Operations for information on selecting and moving multiple classes. Add two more classes: Box and Triangle.

Shape is the super class which encapsulates the basic features of any shape. Add the class Shape by following the steps above. Layout your classes so that Shape sits above the other three classes which are located below it in a horizontal line. The basic qualities of a shape are that it has a position on the screen and that it can be told to paint itself on the screen at this position. For each of the specific shapes we want to "inherit" these basic qualities but refine them as necessary to cater for each specific shape's needs.

Representing inheritance

This inheritance relationship can be represented by drawing a line from your specific types of shapes (eg., Circle, Box, Triangle) up to your base class (Shape) in a similar way to which a family tree might be drawn. The class Circle is said to "derive" or "inherit" from the class Shape. Shape is called a base class and Circle is called the derived class.

On paper we would draw the line from the derived class up to the base class. To do this in Javelin is just as easy:

1. Press and hold the left mouse button on the top section of the derived class.
2. Drag to the base class and release. The Select Relationship type dialog box will appear.
3. Select inheritance and press Ok.

You're almost an expert now!

Associations Between Classes

NOTE: This section only applies to the professional version. If you are using the standard version please skip to the section marked "The Story So Far".

We will now add an association relationship. We know that each graphic window in our simple application is capable of displaying many shapes. It must somehow store a reference to each of these shapes so that it can draw them when required.

To create a reference relationship:

1. Drag from the top section of the GraphicWindow class to the class Shape. The Select Relationship Type dialog box will appear.
2. Select one-to-many.

This specifies that one graphic window can have many shapes. Many is shown with 3 "spokes" and one is shown with 1 spoke. The one or many is known as "cardinality". The graphic symbols at either end of the relationship are called "cardinality anchors". You will notice text drawn at either end of the relationship.

This text describes the role each class plays in reference to the other.

You can position cardinality anchors by moving them with the left mouse button. They are constrained to the class to which they are attached.

Changing Reference Options

We can now change the role name and also the cardinality if required. The role name should be such that the whole reference reads like a sentence. To change the cardinality options:

1. Press and hold the right mouse button while the mouse is over the cardinality anchor at the Shape class. A pop up menu will appear.
2. Select reference details. The reference options dialog box will appear.
3. Change the role name to "displays". The role names are often used when generating code for the relationship in the class declaration. You can read about code generation options for references in the on line help.
4. Change the role name at the Shape end of the reference to "is_displayed_on" by clicking on the cardinality anchor at the GraphicWindow end.

The Story so far...

What you have done is analysed your application's core class model. As you can see with Javelin you can enter your classes straight into the design slate. You can specify relationships between classes. You can modify classes at any time. You can add a complete description for each class describing what it is designed to represent and how it does it. This will help you or anyone else to understand the software.

Javelin is a live, dynamic, self documenting design system. So be clear and concise when adding information to the system. Use informative names like "Circle" instead of names like "Crcl". Enter meaningful comments to the system so that someone who is not acquainted with your design can pick it up quickly. In contrast to an analysis done on a bit of paper an analysis performed using Javelin is already starting to look a lot more manageable but that's only the beginning of what Javelin can do for you...

Design Phase

Now that you have analysed the classes that will be used in your application the next step is to move on to the next phase and further refine your work. This is the design phase where you specify the fields (data members in C++ terminology) and methods (member functions in C++ terminology) for each class. These are called the members or features of the class. Traditionally you might have scribbled the name of a method near the class on your paper based design but you barely have room to specify its parameters nor what it should do nor how it is to do it.

Javelin to the rescue:

To bring up the "Members Dialog Box" containing lists of the members of the Shape class:

1. Press and hold the right mouse button while the mouse pointer is located over the Shape class. A pop up menu will appear.
2. Selecting "Members" and a dialog box will appear containing a list of all the member data and member functions of a class.
   HINT: See the Javelin on line help under Class Members for explanations of the various symbols.

To add a new member:

1. Press either the upper New button for new data members or the lower New button for new methods.
2. After pressing new to add a member you will be prompted to enter its specifications: name, type, access and parameters (for methods only).

To modify an existing member:

1. Double click the left mouse button on a member to modify its specification.

Javelin allows quick access to a function's code, saving you time by avoiding searches through text files. Getting from the design slate to a function specification is only a few clicks away. Compare that with a design written on paper or worse still no design at all!

Adding some Members

One of the basic qualities of any Shape is that it can remember where it is located on the screen. Screen locations are usually represented by an X, Y coordinate pair. We thus add two data members to the Shape class: X and Y, both integers. Bring up the members dialog for the Shape class by following the steps above.

1. Press the upper New button to create a new data member.
2. For its Name field enter x. (by convention JAVA uses lower case first letter for names of data members and methods and upper case first letter for class names. If a name contains extra words like hitCount then the first letter of each extra word is also upper case)
3. For its Type field enter int.
   Note: By default data members have protected access.
4. Press Tab until the cursor is in the Description field and enter "X coordinate of shape's origin" (minus the quotes). At code generation time Javelin will enclose this text in a comment for you.
5. Click on the OK button. Repeat this process for the Y coordinate data member.

Objects that can paint themselves!

Another basic quality of a Shape is its ability to paint itself (real life shapes don't have this ability but they do in our application). The self paint ability will be accomplished by calls to a method called show.

The implementation of each show function will be different for each type of shape: the show method for class Circle draws a circle, the show method for class Box draws a box and so on. We will thus make the show function "abstract" in the class Shape.

With this refinement of the show method in place we can have a reference to any object which extends the class Shape and call it's show function and expect that the correct shape will be drawn. This is known as polymorphism. The caller of the show function needs not know how to draw any of the shapes. This is handled by the shapes themselves.

Add the show method now:

1. Click on the lower New button in the Members dialog box for Shape. Select General and press Ok. The function member dialog box will appear.
2. For its: Name enter show; Type enter void; Parameters enter Graphics g. Set the override to "must be overridden (abstract)".

The Graphics g parameter is a graphics object which is passed to an applet each time it needs redrawing. The Graphics class provides many methods for such things as painting shapes, text, changing colors etc.,

By default methods have public access which means any other object can access the method.

Now we will copy the show method in Shape. We do this from Shape's "members dialog box" by pressing the lower Copy button. Close this dialog and then open Circle's member dialog box and press the lower Paste button. Double click on the show method and change its overide type to can be overridden. Now Circle has overridden the Shape's abstract show method. In the next section we will add some code to that method.

Circles must have a radius so add a "radius" data member to the Circle class and make it of type int. Don't forget to describe the purpose of the data member.

Again we see that Javelin is a live, dynamic design system. Not only can you view class details, members specifications and method code you can modify them quickly and easily and the underlying source files are changed automatically. You can take a peek at the source files using note pad if you would like to see the code that has been automatically added. Make sure that you close note pad when you have finished. See Synchronizing Design and Code in the on line help for more information.

Implementation (coding) Phase

Now that you have your class model's analysis and design completed and have designed and specified your classes by defining their details and member specifications its time to start the implementation or coding phase.

Javelin automatically generates a significant amount of code for you. As you are probably aware much of the necessary JAVA code you need for setting up classes, member data and member functions is frustrating and repetitive - something you would rather have a computer do for you. The interesting part, once you have completed your design is the coding of the method bodies.

Javelin automatically generates source files (.java). If changes are made to class details, member specifications or method code then the files are regenerated when necessary. You should never have a need to do an explicitly code regeration.

Initially the function bodies are empty but in the implementation phase you enter code for your method bodies from within Javelin's in built editor or an editor that you specify in javelin.ini. (See Options|Editor) Javelin preserves the method body code that you have written when it regenerates.

Let's do some coding!

Open up the Circle class' members dialog box. Select the show method and press the "Source" button or alternatively hold down CTRL and double click on the show method. The inbuilt editor will appear with the empty body of the shape method. Between the curly brackets type:

```
g.fillOval(x, y, radius * 2, radius * 2);
```

Press Ok to save the method and this new code will be added to the underlying Circle.java source file.

You will recall that the show function has a parameter of type Graphics called 'g'. Using g.fillOval invokes the fillOval method on this object which causes a circle to be drawn using the current color.

The use of the Graphics class requires that we place an import line in the class source file so that the compiler knows where to find out about the Graphics class. This can be done within Javelin by pressing the right mouse button down on the Circle class and selecting Extras->before class... This will open the inbuilt editor. Type:

```
import java.awt.*;
```

and press Ok. This will add the line to the Class.java source file before the class definition. This line will need to be added to all classes that use classes from the AWT library.

Ready to compile

The Step 3 project is the completion and extension of this example tutorial. Open it up and have a look. You can compile it (assuming you have obtained and installed the JAVA JDK) by pressing the left most button in the tool bar and you can execute it by pressing the button second from the left. This code can be found in the step3 directory.

As an exercise experiment with the application by modifying the way objects are painted or you could even create a new type of shape!

If you only have the evaluation version you can not save changes to your projects. To purchase a real copy open the file named order.txt in the javelin directory to find out how affordable Javelin really is.

Conclusion

During this short introduction to the Javelin Graphical Object Oriented Development Tool you have seen the future of fast, quality software development and how Javelin is helping more and more JAVA programmers build their class models faster and more reliably than ever before. If you are only using the evaluation version we recommend that you purchase either the standard or professional edition so that you can continue on your path to rapid, quality software development. (See order.txt).