8. Java Development Tools

Introduction

This chapter will introduce the reader to various development tools that support the development of large scale Java systems (e.g. Eclipse and NetBeans). The importance of the API documentation and JavaDoc tool will also be demonstrated.

Objectives

By the end of this chapter you will be able to….

- Understand the roles of the JDK and JRE
- Investigate professional development environments (Eclipse and NetBeans)
- Understand the importance of the Javadoc tool and the value of embedding Javadoc comments within your code.
- Write Javadoc comments and generate automatic documentation for your programs.

This chapter consists of thirteen sections :-

1) Software Implementation
2) The JRE
3) Java Programs
4) The JDK
5) Eclipse
6) Eclipse architecture
7) Eclipse Features
8) NetBeans
9) Developing Graphical Interfaces Using NetBeans
10) Applying Layout Managers Using NetBeans
11) Adding Action Listeners
12) The Javadoc Tool
13) Summary

8.1 Software Implementation

Before a computer can complete useful tasks for us (e.g. check the spelling in our documents) software needs to be written and implemented on the machine it will run on. Software implementation involves the writing of program source code and preparation for execution on a particular machine. Of course before the software is written it needs to be designed and at some point it needs to be tested. There are many iterative lifecycles to support the process of design, implementation and testing that involve multiple implementation phases. Of particular concern here are the three long established approaches to getting source code to execute on a particular machine…

- compilation into machine-language object code
- direct execution of source code by ‘interpreter’ program
- compilation into intermediate object code which is then interpreted by run-time system

Implementing Java programs involves compiling the source code (Java) into intermediate object code which is then interpreted by a run-time system called the JRE. This approach has some advantages and disadvantages and it is worth comparing these three options in order to appreciate the implications for the Java developer.

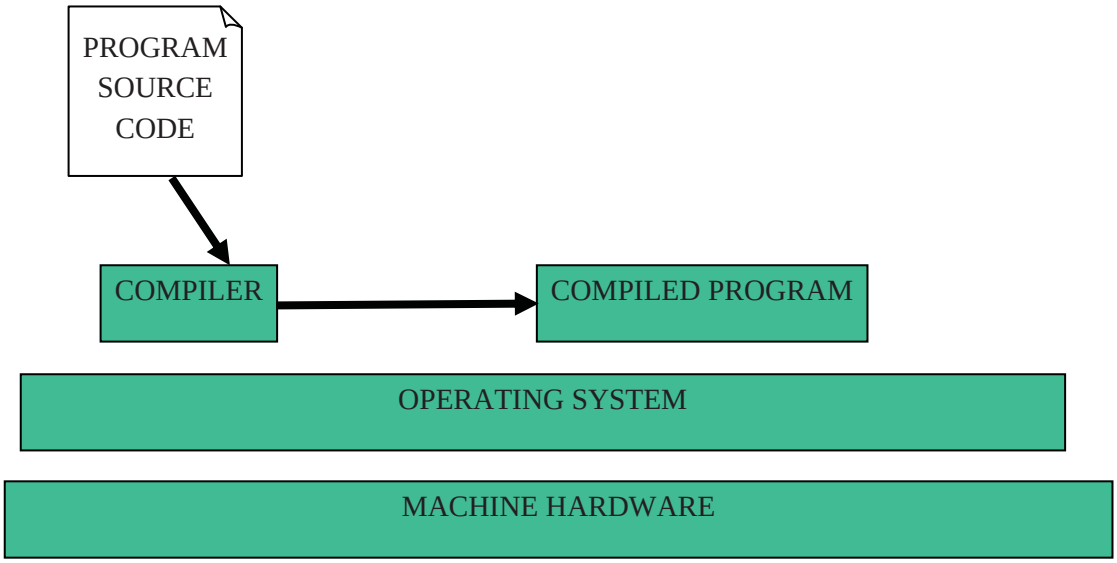
Compilation

The compiler translates the source code into machine code for the relevant hardware/OS combination.

Strictly speaking there are two stages: compilation of program units (usually files), followed by ‘linking’ when the complete executable program is put together including the separate program units and relevant library code etc.

The compiled program then runs as a ‘native’ application for that platform.

This is the oldest model, used by early languages like Fortran and Cobol, and many modern ones like C++. It allows fast execution speeds but requires re-compilation of the program each time the code is changed.



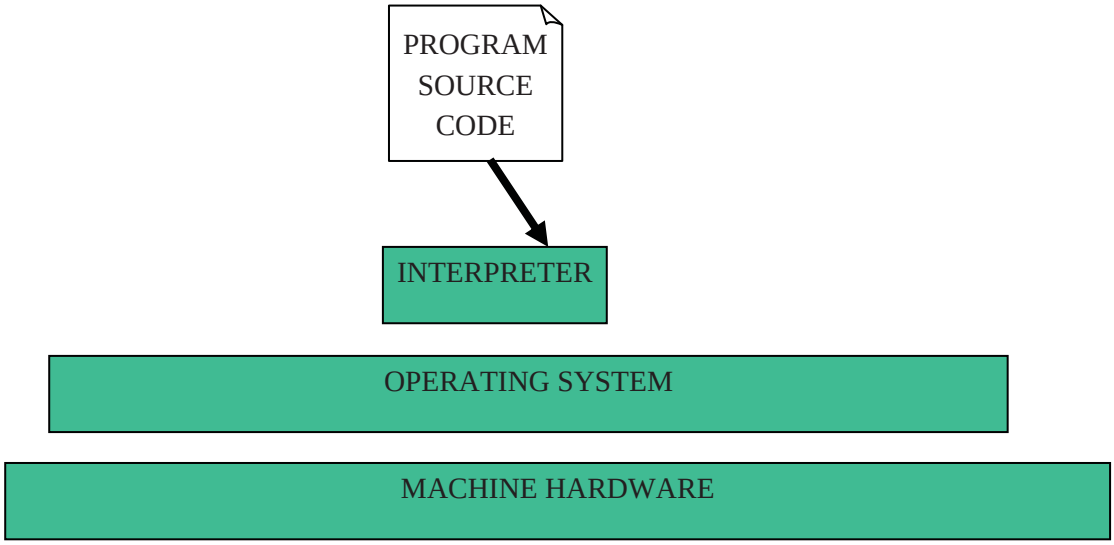
Interpretation

Here the source code is not translated into machine code. Instead an interpreter reads the source code and performs the actions it specifies.

We can say that the interpreter is like a ‘virtual machine’ whose machine language is the source code language.

No re-compilation is required after changing the code, but the interpretation process inflicts a significant impact on execution speed.

Scripting languages tend to work in this way.

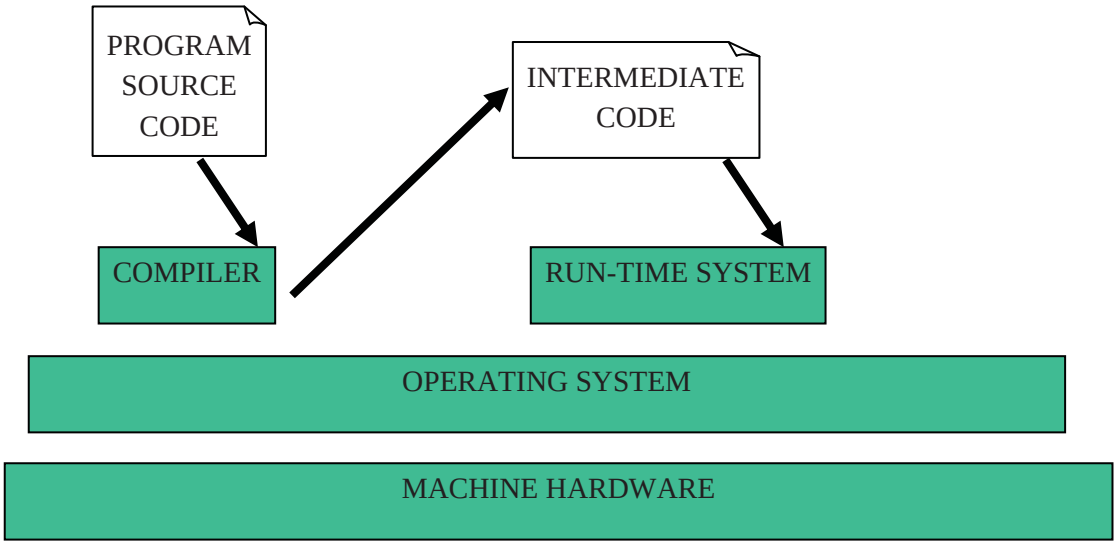


Intermediate Code

This model is a hybrid between the previous two.

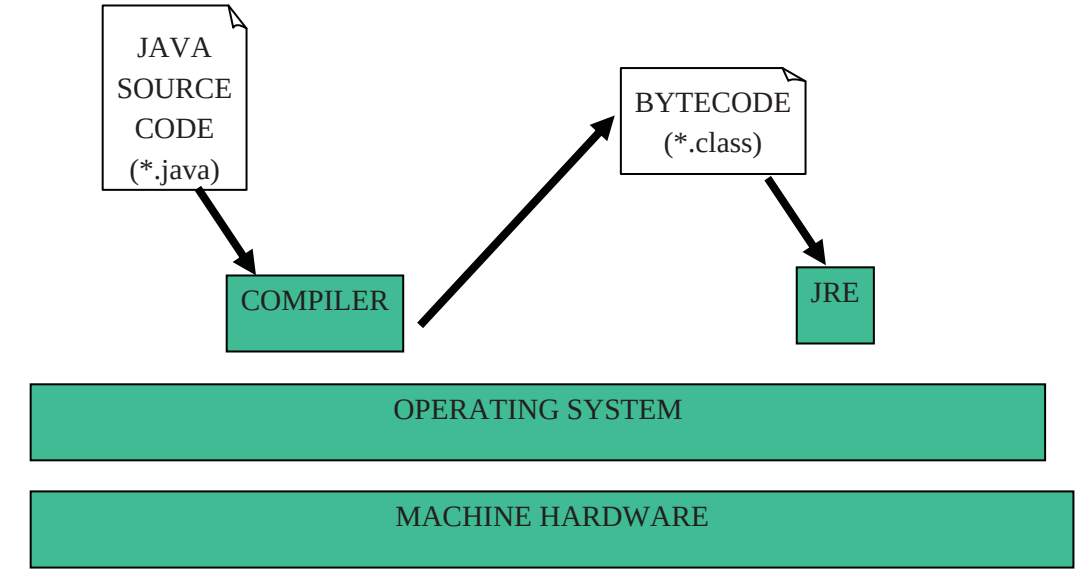
Compilation takes place to convert the source code into a more efficient intermediate representation which can be executed by a 'run-time system' (again a sort of 'virtual machine') more quickly that direct interpretation of the source code. However, the use of an intermediate code which is then executed by run-time system software allows the compilation process to be independent of the OS/hardware platform, i.e. the same intermediate code should run on different platforms so long as an appropriate run-time system is available for each platform.

This approach is long-established (e.g. in Pascal from the early 1970s) and is how Java works.



8.2 The JRE

To run Java programs we must first generate intermediate code (called bytecode) using a compiler available as part of the Java Development Kit (JDK) – see section 8.4 below.



A version of the Java Runtime Environment (JRE), which incorporates a Java Virtual machine (VM), is required to execute the bytecode and the Java library packages. Thus a JRE must be present on any machine which is to run Java programs.

The Java bytecode is standard and platform independent and as JRE's have been created for most computing devices (including PC's, laptops, mobile devices, mobile phones, internet devices etc) this makes Java programs highly portable.

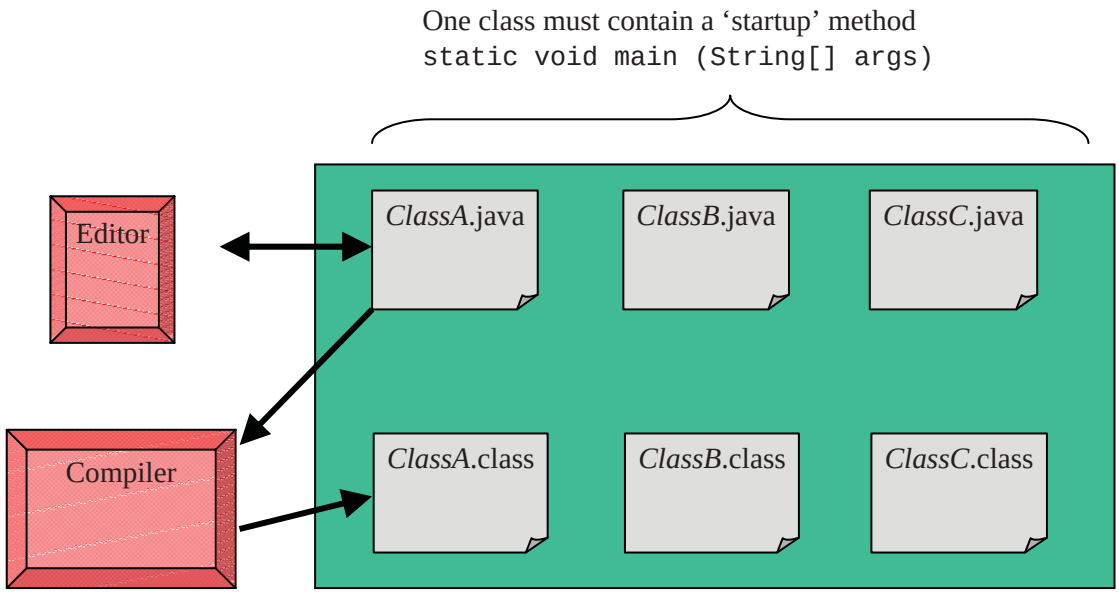
8.3 Java Programs

Whatever mode of execution is employed, programmers can work with a variety of tools to create source code. Options include the use of simple discrete tools (e.g. editor, compiler, interpreter) invoked manually as required or alternatively the use of an Integrated Development Environment (IDE) which incorporates these implementation tools behind a seamless interface. Still more sophisticated CASE (Computer Aided Software Engineering) tools which integrate the implementation process with other phases of the development cycle – such software could take UML class diagrams and generate classes and method stubs automatically saving some of the effort required to write the Java code.

When writing java programs each class (or interface) in a Java program has its own **name.java** file containing the source code.

These are processed by the compiler to produce **name.class** files containing the corresponding bytecode.

To actually run as an application, one of the classes must contain a main() method with the signature shown above.



8.4 The JDK

To develop Java programs you must first install the Java Development Kit (JDK). This was developed by Sun and is available freely from the internet via http://java.sun.com/. Prior to version 5.0 (or 1.5) this was known as the Java Software Development Kit (SDK).

A Java IDE's, e.g. Eclipse or NetBeans, sits on top of' the JDK to add the IDE features - these may include interface development tools, code debugging tools, testing tools and refactoring tools (more on these later). When using an IDE it is easy to forget that much of the functionality is in fact part of the JDK and when the IDE is asked to compile a program it is infact just passing on this request to the JDK sitting underneath.

We can use the JDK directly from the command line to compile and run Java programs though mostly it is easier to use the additional facilities offered by an IDE.

Somewhat confusingly the current version of Java is known both as 6.0, 1.6 and even 1.6.0. These supposedly have subtly different meanings – don't worry about it!

There are many tools in the JDK. A description of each of these is available from http://java.sun.com/javase/downloads/ and following the links for Documentation, APIs and JDK Programmer guides.

The two most important basic tools are:

javac – the java compiler

java – the java program launcher (that runs the VM)

To compile MyProg.java we type

javac MyProg.java

If successful this will create MyProg.class

To run Myprog (assuming it has a main() method) we type

java MyProg

Another, extremely useful tool, is javadoc - this uses comments in Java source code to generate automatic documentation for programs.

8.5 Eclipse

Moving to an ‘industrial strength’ IDE is an important stepping stone in your progress as a software developer, like riding a bicycle without stabilisers for the first time. With some practice you will soon find it offers lots of helpful and time-saving facilities that you will not want to work without again…

Eclipse is a flexible and extensible IDE platform. It was first developed by IBM but is now open source and can be downloaded from the Eclipse Foundation at www.eclipse.org.

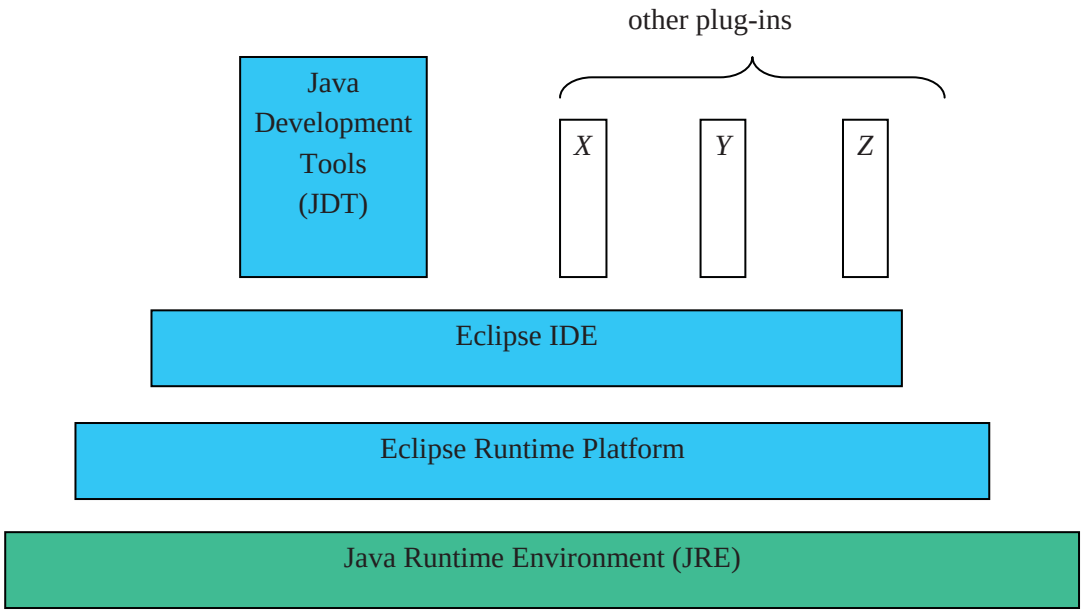
Eclipse was itself written in Java and like all Java programs it needs a JRE to work but using this it will therefore run on a large range of platforms. It comes with Java Development Tools (JDT) installed by default but also has extensions and additional tools to support the development of programs in other languages.

The Eclipse ‘runtime platform’ is sufficiently flexible that it can even be used as a framework for the development of applications unrelated to IDEs.

8.6 Eclipse Architecture

Because Eclipse relies on the JRE it is intrinsically multi-platform (Windows, Linux, Mac etc.)

The elements shown in blue below are part of the basic Eclipse download.



Lots of other plug-ins are available to support additional tools (style-checking, testing, GUI design etc.) and languages (including C/C++).

Download .zip file from www.eclipse.org

Like any IDE Eclipse requires an up-to-date JDK is installed as it will use this to compile programs and generate Java documentation.

8.7 Eclipse Features

Eclipse provides a range of features offering powerful support for a programmer wishing to develop large Java programs. These include :-

- code completion and help facilities,
- a code formatter which can automatically adjust a huge variety of code format aspects,
- powerful facilities to support program development including refactoring (see Chapter 10 section 4) and automatic testing facilities (see Chapter 10 section 5).

8.8 NetBeans

NetBeans is another very powerful IDE, originally developed and distributed by Sun Microsystems, NetBeans was made open source in 2000 and is extensively supported with plug-ins and video tutorials developed by the open source community.

NetBbeans can be downloaded for free from www.netbeans.org

NetBeans has many features that have been developed to aid novice programmers and save time. These include…

- A quick start guide, on line tutorials and online videos (covering among other topics an introduction to the IDE, the Javascript debugger, and PHP support).

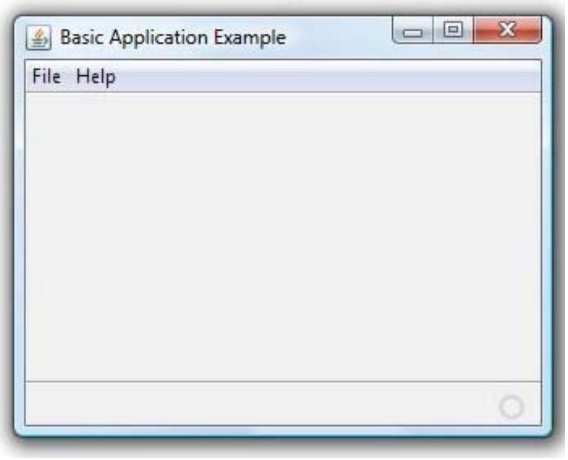
- Help features with automatic pop up windows highlighting relevant parts of the Java API,

- formatting and debugging facilities

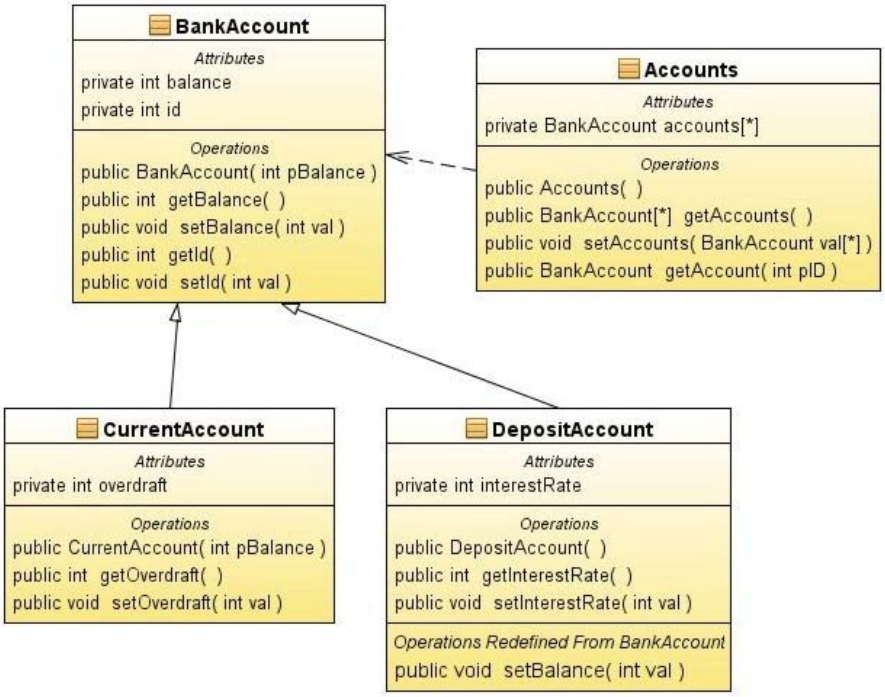
- code completion features that will auto insert for loops, try catch blocks and constructors (among others). These code completion facilities do far more than cut and paste sample blocks of text – for example when inserting a constructor for a subclass the code to call a super-constructor is automatically added and relevant parameters are defined for the sub-constructor and passed to the super-constructor.

- Project management features that monitor changes to a project that allow the project to be reverted to an earlier state.

Netbeans also includes templates for a range of Java applications including windows, enterprise, web and mobile applications. It even has templates for C++/PHP/ Ruby and database applications. Thus with just a few button clicks Netbeans will create a simple Java application with a main method that invokes a blank GUI as shown below….



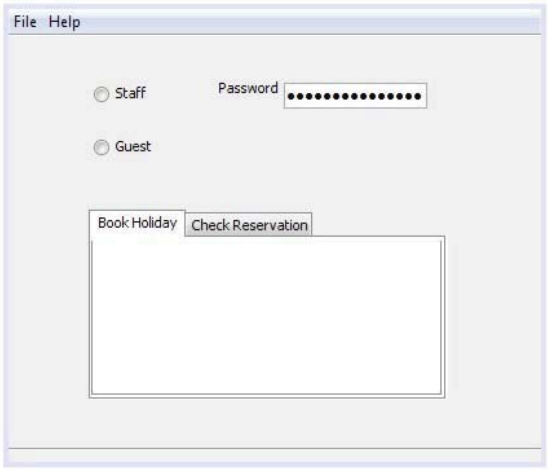
Additionally one of the many plug-ins available provides support for the creation of UML diagrams. While this tool does have its limitations it does not just support the creation of the diagrams but provides facilities that you may expect from a CASE tool such as automatic code generation (from a diagram) and reverse engineering (i.e. analysing current code to generate a UML diagram from the code).



Thus given the diagram above NetBeans would create four classes, three of which would automatically be placed in an inheritance hierarchy. Method stubs would be created for each of the specified methods and while the code inside each method would still need to be written the constructor for CurrentAccount would automatically invoke the constructor for Bank Account passing the parameter pBalance upwards.

8.9 Developing Graphical Interfaces Using NetBeans

NetBeans has a visual tool to help with the development of graphical user interfaces. This tools allows a window to be created and a range of objects to be dropped onto it including panels, tabbed panes, scrolled panes, buttons, radio buttons, check boxes, combo boxes, password fields, progress bars, and trees.



8.10 Applying Layout Managers Using NetBeans

When designing a graphical user interface it is ‘normal’ for objects to be placed on a window in precise positions. Interfaces are thus displayed exactly how they are designed. For this to work well an interface designer usually has some idea of the size of the display they are designing an interface for. Most PC games are designed to run on window of 14” – 21”. Most mobile games are designed to work on much smaller screens.

However Java programs are expected to be platform independent – one of the strengths of Java is that code written in Java will work on a PC, laptop or small mobile device irrespective of the hardware or operating systems these devices use (as long as they all have a JRE). Thus the same interface should work on a large visual display unit or small mobile screen. For this reason when developing interfaces in Java it is usual to give Java some control over deciding exactly how / where to display objects. Thus, at run time, Java can reconfigure a window to fit whatever device is being used to run the program. This has benefits as the display will be reconfigured automatically to fit whatever device is being used to run the program but this flexibility comes at a cost. As some control is given to Java the interface designer cannot be certain exactly how the interface will look though they can give Java some ‘instructions’ on how an interface should be displayed.

Layout Managers are ‘objects’ that define how an interface should be displayed and it is normal to create a layout manager and assign it to a ‘container’ object such as a window frame.

There are a range of common layout managers - two of the most common being flow layout and grid layout. Using flow layout Java will arrange objects in a row and objects will automatically flow onto another row if the window is not big enough to hold each object on one row.

Flow Layout In Action

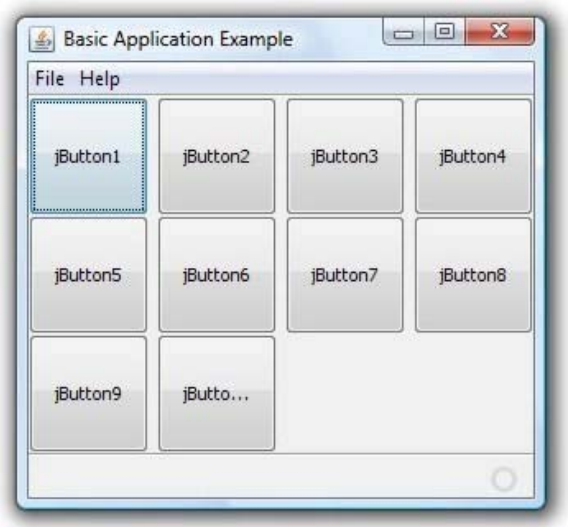
Two windows are shown below with ten buttons on each – in the second figure Java places some of the buttons on a second row as the window has been resized and the object will not fit on one line.

A flow manager can be created with various properties to define the justification of the object (centre, left or right) and to define the horizontal and vertical space between objects.

Grid Layout In Action

Another very common layout manager is gird layout. When applying objects in a grid a number of rows can be specified. Java will work out how many columns it needs to use in order to place the required number of objects in that number of rows. The objects will be resized to fit the window. Java will shrink or enlarge the objects to fit the window – even if this means that not all of the text can be shown.

The two figures below show the same 10 buttons being displayed on a grid with three rows.

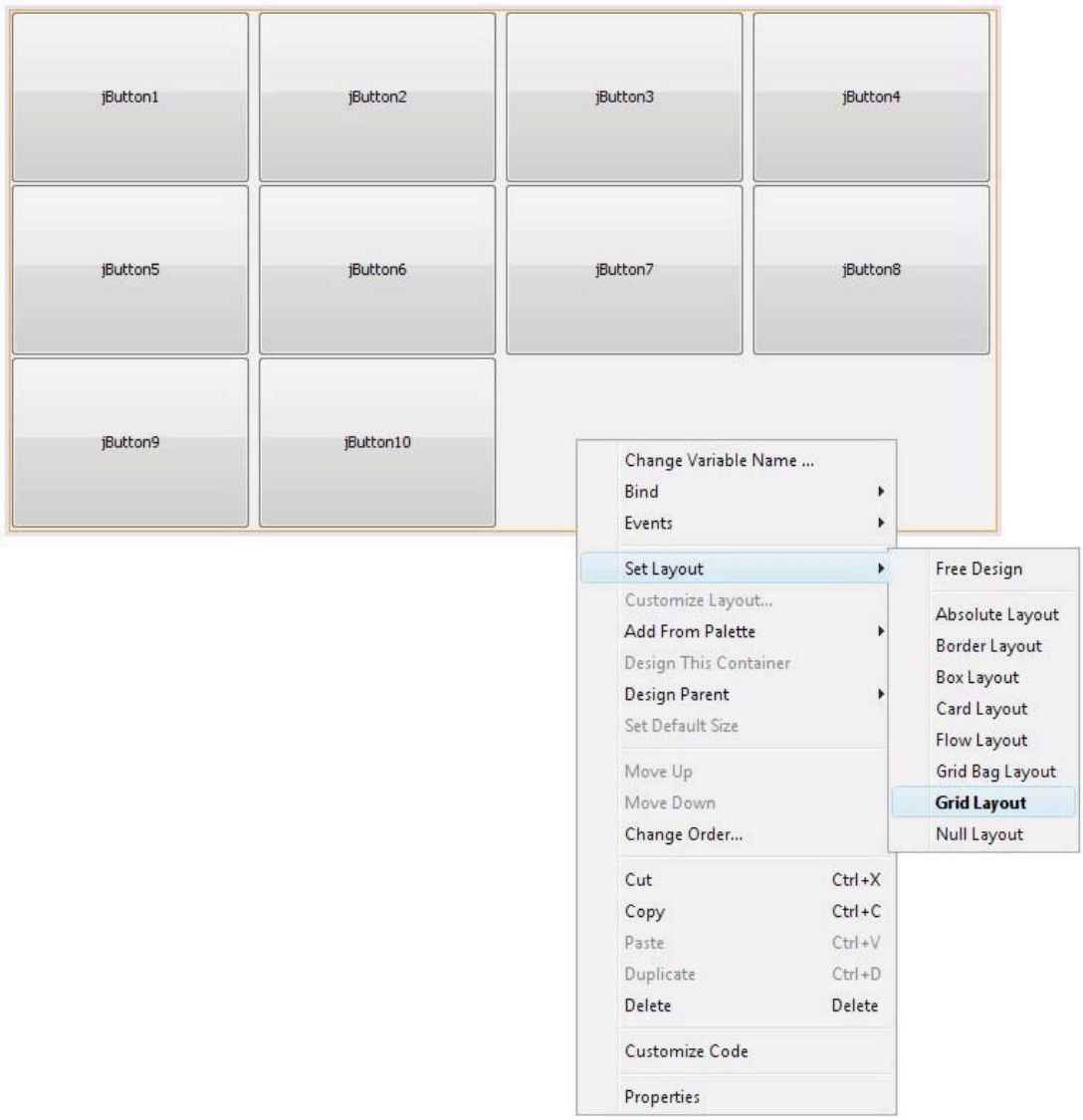


Note in the second of these the buttons have been resized to fit a different window – in doing so there is not enough space to display all of the text for button ten.

Layout managers can be used in layers and applied to any container object Thus we can create a grid where one element contains multiple objects arranged in a flow.

By using layout managers we can create displays that are flexible and can be displayed on a range of display equipment thus enabling our programs to be platform independent - however this does mean we lose some of the fine control in the design.

Creating a layout managers and setting their properties using NetBeans is very easy. Right clicking on a container object will bring up a context sensitive menu one option of which will be to apply a layout manager. Selecting this option will display the list of layout managers possible. Once a layout manager has been selected a window will appear to allow its properties to be specified.



8.11 Adding Action Listeners

The final step in creating a working interface is to create Action Listeners and apply them to object on the interface. Action listeners are objects that listen for user actions and respond accordingly – thus when a ‘calculate’ button is pushed a program will calculate and display some results.

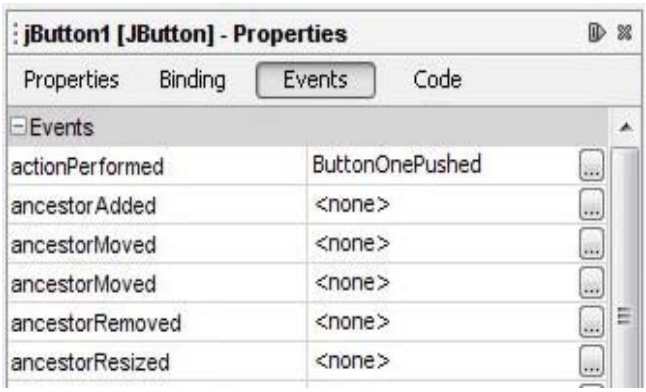
Action listeners can be created for all objects on an interface and for a range of actions for each object. Though many objects, such as labels, will be not be required to respond to a users interactions.

Creating an action listener in NetBeans is very easy.

When any object is selected, in design mode, a properties window is displayed. Selecting the events tab will allow a range of actions to be created for this object. Once one has been selected an ‘Action Listener’ stub will be written. Code will automatically be written to create an object of this class and assign this to the object selected in the design.

All that remains is for the programmer to write the code to define the response to the users action.

In the example below an event has been created to listen for button1 being pushed.




```
private void ButtonOnePushed(java.awt.event.ActionEvent evt) {
    System.out.println("button 1 pushed");

}
```

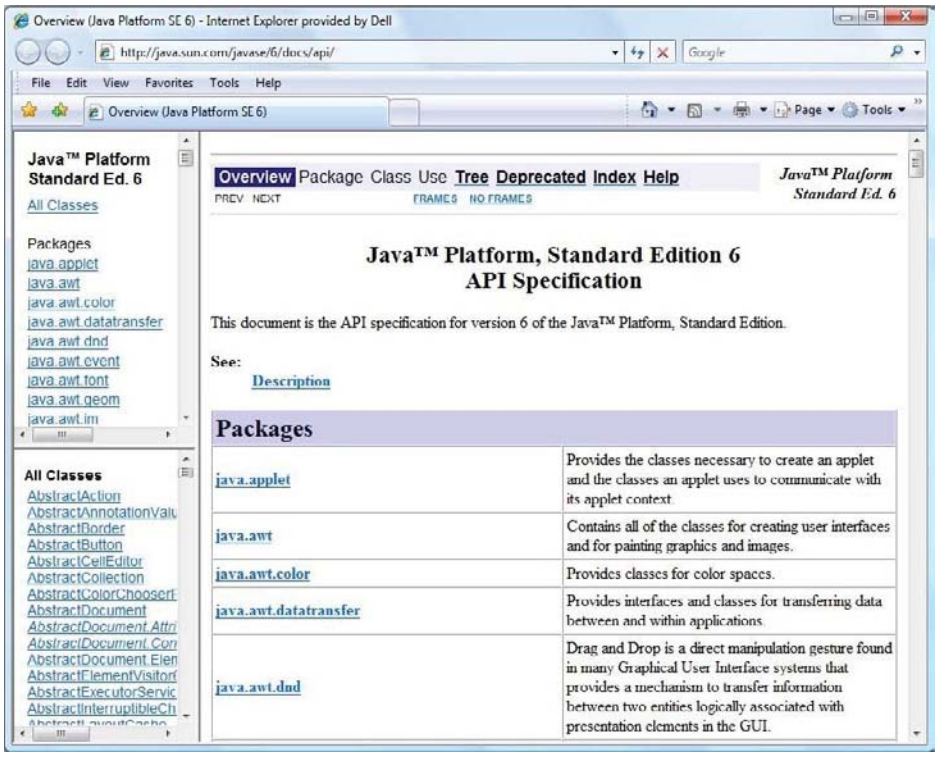
8.12 The Javadoc Tool

One particularly useful tool within the JDK is ‘javadoc’.

Programmers for many years have been burdened with the tedious and time consuming task of writing documentation. Some of this documentation is intended for users and explain what a program does and how to use it. Other documentation is intended for future programmers who will need to amend and adapt the program so that its functionality will change as the needs of an organisation change. These programmers need to know what the program does and how it is structured. They need to know :-

- what packages it contains
- what classes exist in each of these packages and what these classes do,
- what methods exist in each class and for each of these methods
    - what does the method do
    - what parameters does it require
    - what, if any, value is returned.

The javadoc tool won’t produce a user guide but it will provide a technical description of the program. The tool analyses *.java source files and produces documentation as a set of web pages (HTML files) in the same format as API documentation on the Sun website.



This website provides, as a set of indexed web pages, essential details of the Java language specification including all packages, classes, methods and method parameters and return values. This extremely useful information was not generated by hand but generated using the javadoc tool.

The same tool can produce the same style of documentation doe any Java program. However to do this it relies on properly formatted (and informative!) javadoc-style comments in source files, including tags such as @author, @param etc.

Because this documentation is generated automatically, at the push of a button, it saves programmers from a tedious, time consuming and error prone task. Furthermore the documentation can be updated whenever a program is changed.

This tool does require a programmer to add meaningful comments to code as it is developed and poor attention to commenting of source code will result in poor javadoc output! On the other hand, if the commenting is done properly then the reference documentation is produced 'for free'!

Javadoc comments should be added at the start of every class using the standard tags @author and @version to give an overview of that class in the following format..

```
/**********************************
 * A description of the class
 *
 * @author name of author
 * @version details of version (and date).
**********************************/
```

A similar comment should be provided for every method using the @param and @return to describe each parameter and to describe the value returned. The details of each parameter, starting with the name of the parameter, should be provided on separate lines as shown below.

```
/**********************************
 * A description of the method
 *
 * @param nameOf1stParam description of first parameter
 * @param nameOf2ndParam description of second parameter
 * @return description of value returned
**********************************/
```

Activity 1

The method below takes two integer numbers and adds them together. This value is returned by the method. Write a javadoc comment, using appropriate tags, to describe this method.

```
public int add(int pNumber1, int pNumber2) {
        return (pNumber1 + pNumber2);
}
```

Feedback 1

```
/***********************************
 * This method adds two integer numbers.
 *
 * @param pNumber1 first number to be added
 * @param pNumber2 second number to be added
 * @return sum of the two numbers
***********************************/
```

The javadoc tool cannot analyse the description of a method to determine if this description provides an accurate summary of that method. However the tool does do far more than cut and paste a programmers comments onto a web document. One thing this tool does is analyse method signatures and compare this with the tags in the comments to check for logical errors. If a method requires three parameters but the comment only describes two then this error will be detected and reported to the programmer.

> “warning - @param argument "pNumber3" is not a parameter name.”

By analysing the code the tool also provides additional information about the class, the methods that have been inherited and the methods that have been overridden.

The reference documentation produced is for programmers using the class(es) concerned, it does not include comments within methods intended for programmers editing the class source code in maintenance work.

The generated documentation can be placed in a subfolder and can be inspected by opening the file index.html in a web browser.

‘javadoc’ will only document classes and members which are public or protected because only these are available to other programmers. Items which are private or for which no access qualifier is given are omitted.

8.13 Summary

There are various ways in which source code in a programming language can be executed. Java uses an intermediate language called ‘bytecode’ and a main method is required.

We can go ‘back to basics’ creating Java programs using a text editor and the tools of the Java Development Kit but the use of a professional IDE, sitting on top of the JDK, can offer additional facilities to support programmers.

Specialist tools are available for aspects of development such as GUI design, diagramming and documentation but some IDEs go beyond the basics and provide some of these facilities in built into the IDE.

Eclipse and NetBeans are two open source powerful IDEs which support the development of Java programs as standard. These are both available as a free download.

These tools can initially seem daunting as they provides extensive support for professional development including code formatting and refactoring though online help does exist and both tools provide some level of automatic code generate, e.g. the main method, to make the job of the programmer easier.

The Javadoc tool is an extremely useful and timesaving device but it does require the programmer to inserting meaningful Javadoc style comments into the code (for all classes and all public methods).