



Hello and Welcome! This is an e-learning course on Basic Java Programming presented by the Java Technical Competency Development Team. There are 7 modules in this e-learning course.



Basic Java Programming

Introduction



Hello and Welcome! This is an e-learning course on Basic Java Programming presented by the Java Technical Competency Development Team. There are 7 modules in this e-learning course.

Course Objective

- To introduce Java Architecture & appreciate basic syntax in Java Language
- To apply Object Oriented Concepts using Java
- To illustrate how to make use of standard Java Class Library and create reusable classes.
- To learn what is required to package and deploy a Java application
- To introduce Exception Handling in Java
- To Introduce Java Applets
- To introduce User Interface Concepts & Event Handling using Swing



Copyright © 2005, Infosys
Technologies Ltd

3

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

By the end of the course, you will have the

- Knowledge of the Java Architecture, Ability to write Java programs using keywords, control statements and various other constructs
- Ability to use the classes declared in the class libraries, and to create custom libraries.
- Ability to write programs that handles exceptions.
- Ability to create Graphical User interfaces using the library classes and to handle events generated by them.



Basic Java Programming

Module – 1 : Object Oriented Concepts



We shall now move on to the module on Object Oriented Concepts.

Learning Outcomes

- To refresh Object Oriented Concepts



Copyright © 2005, Infosys
Technologies Ltd

5

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Java is an Object-Oriented Language. So before learning java let us have a look on the Object-Oriented Concepts.

Class and Object

What is a Class?

- A class is a *blueprint* or *prototype* that defines the variables and the methods (functions) common to all objects of a certain kind.

What is an Object?

- An object is a representative or specimen of a class. Software objects are often used to model real-world objects you find in everyday life.



Copyright © 2005, Infosys Technologies Ltd

6

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

What is an Object?

An object in the software world means a bundle of related variables and functions known as methods.

Software objects are often used to model real-world objects you find in everyday life.

The real world objects and software objects share two characteristics :

1. State : condition of an item.
2. Behavior : Observable effects of an operation or event including its results.

State can be of two types : Active state which reflects the behavior and Passive State refers to state which does not change

The behavior of an object forms the interface to the external world.

A class is a *blueprint* or *prototype* that defines the variables and the methods (or functions) common to all objects of a certain kind.

Constituents of a class are :

1. Member Variable : Variables declared within a class
2. Member Functions : Functions or methods declared within the class. These methods operate upon the member variables of the class.

So a class is a definition of a set of data and methods. When memory space for this data is actually allocated, we say that class is instantiated i.e an object is created.

Class defines how the object should be.

Object is an instance of a class. Each instance of a class will have its own data.

Features of OOP

• Abstraction:

- The process of extracting the essential information and hiding the irrelevant details

• Encapsulation:

- The process of binding code and data together in the form of a capsule

• Inheritance:

- The feature by which one class acquires the properties and functionalities of another class

• Polymorphism:

- The feature that allows the same interface to be used for a general set of actions



Copyright © 2005, Infosys Technologies Ltd

7

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Features of Object Oriented Programming

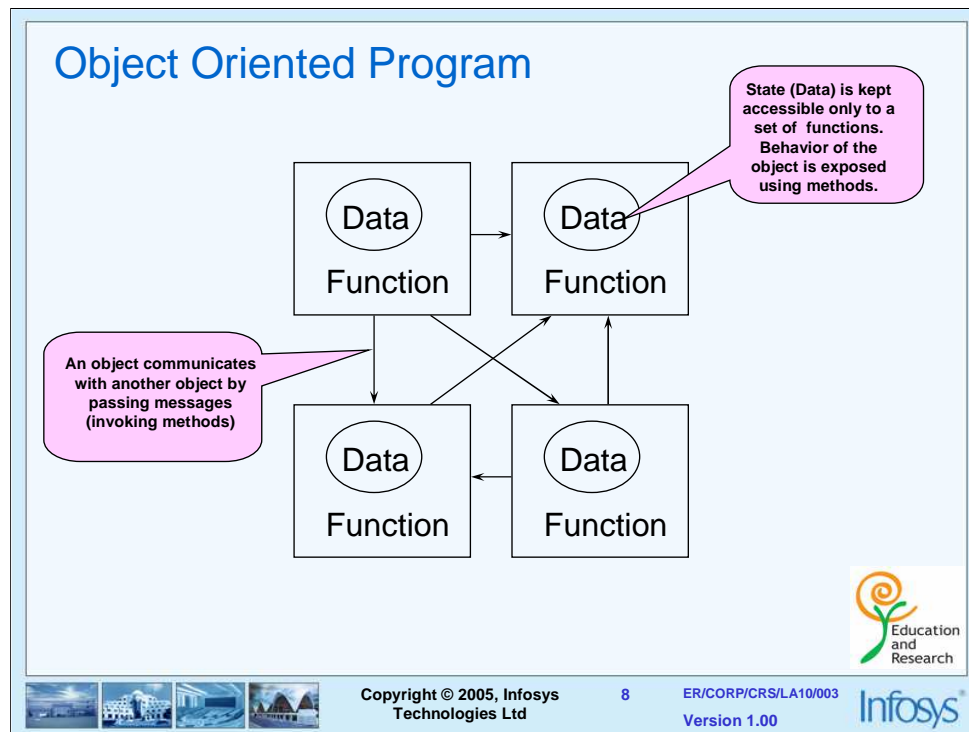
Abstraction is the process of exposing the relevant things and ignoring the irrelevant details. The easiest way to understand and appreciate this concept of handling complexity is by studying the example of Globe, a model/prototype of earth that is used by students to understand its geography. Globe provides only that information that is required and if too much of information is mentioned in it i.e. streets, lakes etc, it becomes too complex to comprehend. Hence Globe abstracts unwanted information and makes it easy to comprehend the complex earth.

Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. The data is not accessible to the outside world and only those functions that are wrapped in the class can access it. These functions provide the interface between the object's data and the program. The insulation of the data from the direct access by the program is called **data hiding**.

In OOP, code and data are merged into an object so that the user of an object can never peek inside the box. This is defined as encapsulation (i.e. Object is a capsule encapsulating data and behavior). All communication to it is through **messages** (i.e function calls which we use to communicate to the object). Messages define the interface to the object. Everything an object can do is represented by its message interface. Therefore, we need not know anything about what is in the object when we use it.

Inheritance is the process by which one class acquires the properties and functionalities of another class. This is important because it supports the concept of hierarchical classification.. Inheritance provides the idea of reusability of code and each sub class defines only those features that are unique to it.

Polymorphism is a feature that allows one interface to be used for a general class of actions. An operation may exhibit different behavior in different instances. The behavior depends on the types of data used in the operation. It plays an important role in allowing objects having different internal structures to share the same external interface. Polymorphism is extensively used in implementing inheritance.



This diagram represents a simple OO program

- The boxes represent the objects
- The circles inside the boxes signify the data hiding concept. It implies that the data is secure
- The arrows represent the messages that are sent across (or the methods invoked)

One object interacts with another object by invoking methods on that object which helps in achieving higher order of functionality.

One object invoking methods on another object is known as Message Passing. Also known as Method Invocation.

Object Oriented Programming vs Procedural Programming

Procedural Programming	Object Oriented Programming
Emphasis on algorithms, procedures	Emphasis on Data; binding of data structures with methods that operate on data
Real world is represented by logical entities and control flow. Tries to fit real life problem into procedural language	Real world is represented by objects mimicking external entities. Allows modeling of real life problem into objects with state and behavior
In a given module, data and procedures are separate	Data (State) is encapsulated effectively by methods (Behavior)
Program modules are linked through parameter passing mechanism	Program modules are integrated parts of overall program. Objects interact with each other by Message passing
Uses abstraction at procedure level	Uses abstraction at class and object level
Algorithmic decomposition tends to focus on the sequence of events	Object Oriented decomposition focuses on abstracted objects and their interaction
Passive and dumb data structures used by active methods	Active and intelligent data structures (object) encapsulates all passive procedures
Procedural languages: C, COBOL, PASCAL	OO Languages: C++, Small Talk, Java, C#



Copyright © 2005, Infosys Technologies Ltd

9

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Have a look on the Comparison between the Object Oriented programming and Procedural Programming.



Basic Java Programming

Module – 2 : Java Architecture



We shall now move on to the module on Java Architecture.

Learning Outcomes

- After completion of the module you will be able to Understand
 - The Java Architecture
 - How to write a simple program using Java
 - How to compile and run a program written using Java



Copyright © 2005, Infosys
Technologies Ltd

11

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

After completion of the module you will be able to Understand

- The Java Architecture
- How to write a simple program using Java
- How to compile and run a program written using Java

Introduction to Java

- A language developed by Sun Microsystems
- A general-purpose language
- High-level language
- Developed initially for consumer devices
- Popular platform to develop enterprise applications
 - Finds use in internet based applications



Copyright © 2005, Infosys
Technologies Ltd

12

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Java was conceived by a team of engineers in Sun Microsystems in 1991 as a part of a research project, which was led by James Gosling and Patrick Naughton. It took 18 months to develop the first working version. This language was initially called “Oak” but was renamed as “Java” in 1995. Modeled after C++, the Java language was designed to be small, simple, and portable across platforms and operating systems, both at the source and at the binary level, which means that the same Java program can run on any machine.

Features of Java

- Object-oriented
- Simpler language
 - Compared to earlier OO languages like C++, it is simple
 - Designed considering the pitfalls of earlier languages
- Robust
- Architecture Neutral / Portable
 - Example: Java code compiled on Windows can be run on Unix without recompilation
- Secure
 - Built-in security features like absence of pointers and confinement of the java program within its runtime environment
- Support for Multithreading at language level
- Designed to handle Distributed applications



Copyright © 2005, Infosys Technologies Ltd

13

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Features of Java

Java is an Object Oriented Language.

Object-Oriented design is a technique that helps the programmer visualize the program using real-life objects.

Java is a Simple Language.

The reason is, Certain complex features like operator overloading, multiple inheritance, pointers, explicit memory de allocation are not present in Java language.

Using Java we can write Robust Programs.

The Two main reasons for program failure are: (1) memory management mistakes (2) mishandled run time errors. Java has solution to these problems.

- (1) With Java, Memory management is very easy since the de allocation is done by the garbage collector automatically.
- (2) Java provides object oriented exception handling to manage run time errors.

Java is platform independent i.e Architecture Neutral / Portable (The idea behind it is:- Write once, Run anywhere)

Java is an Interpreter based language. With Java, the program needs to be compiled only once, and the code generated by the Java compiler can run on any platform. Java is platform independent at both the source and the binary level.

Java can be easily ported on any type of system and irrespective of the operating system being used. Java achieves this portability due to its feature of **Implementation Independency**.

Java is a **secured programming language** because it provides the user with a Virtual **Firewall** between the applications and the computer thus ensuring that the data in the user's system is protected by any possible Infectious contents. This is achieved by confining the Java program within the Java Runtime Environment.

Multithreading is the capability for a program to perform several tasks simultaneously within a program. A good example of multithreading would be one of the game software where the GUI display, sound effect, timer control and score updating are happening within the same process simultaneously. In network programming, a server can serve multiple clients at the same time via multithreading.

Java is designed for the **distributed environment** of the Internet. Java allows objects on two different computers to execute procedures remotely.

Platform independence

- Java is a language that is platform independent.
- A platform is the hardware and software environment in which a program runs
- Once compiled, code will run on any platform without recompiling or any kind of modification.
 - “Write Once Run Anywhere”
- This is made possible by making use of a **Java Virtual Machine** commonly known as JVM



Copyright © 2005, Infosys
Technologies Ltd

14

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Java is a language that is platform independent.

A platform is the hardware and software environment in which a program runs

Once compiled, code will run on any platform without recompiling or any kind of modification.

This is made possible by making use of a **Java Virtual Machine** commonly known as JVM

Java Virtual Machine (JVM)

- ❶ The source code of Java will be stored in a text file with extension .java
- ❷ The Java compiler compiles a .java file into byte code
- ❸ The byte code will be in a file with extension .class
- ❹ The .class file that is generated is the machine code of this processor.
 - Byte code is a binary language
- ❺ The byte code is interpreted by the JVM
- ❻ JVM varies from platform to platform, or, JVM is platform dependent
- ❼ JVM can be considered as a processor purely implemented with software.
- ❽ The interface that the JVM has to the .class file remains the same irrespective of the underlying platform.
 - This makes platform independence possible



Copyright © 2005, Infosys
Technologies Ltd

15

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The source code of Java will be stored in a text file with extension .java

The Java compiler compiles a .java file into byte code. Byte code is a binary language.

The byte code will be in a file with extension .class

The .class file that is generated is the machine code of this processor.

The byte code is interpreted by the JVM

JVM varies from platform to platform, or, JVM is platform dependent

JVM can be considered as a processor purely implemented with software.

The interface that the JVM has to the .class file remains the same irrespective of the underlying platform.

This makes platform independence possible

Java Virtual Machine (JVM) (Contd...)

- The JVM interprets the .class file to the machine language of the underlying platform.
- The underlying platform processes the commands given by the JVM



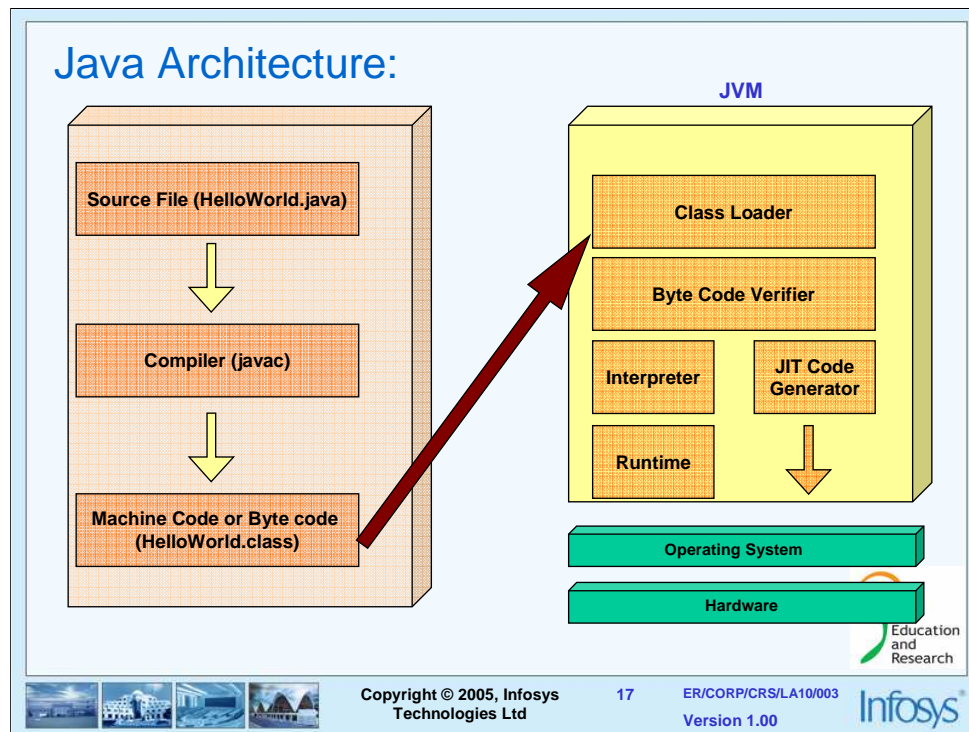
Copyright © 2005, Infosys
Technologies Ltd

16

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The JVM interprets the .class file to the machine language of the underlying platform.
The underlying platform processes the commands given by the JVM



The JVM Runtime Environment

The byte codes are stored in class files. At runtime, the bytecodes that make up a java software program are loaded, checked, and run in an interpreter. The interpreter has two functions: it executes bytecodes, and makes the appropriate calls to the underlying hardware.

The Java runtime environment runs code compiled for a JVM and performs four main tasks:

- Loads Code loader - Performed by the class loader
- Verifies Code verifier - Performed by the bytecode verifier
- Executes Code interpreter - Performed by the runtime interpreter
- Garbage Collection - De allocates memory not being used

Class Loader

The class loader loads all classes needed for the execution of a program. The class loader adds security by separating the namespaces for the classes of the local file system from those imported from network sources.


Once all the classes have been loaded, the memory layout of the executable file is determined. At this point specific memory addresses are assigned to symbolic references and the lookup table is created. Because memory layout occurs at runtime, the java technology interpreter adds protection against unauthorized access into the restricted areas of code.

Java software code passes several tests before actually running on your machine. The JVM puts the code through a bytecode verifier that tests the format of code fragments and checks code fragments for illegal code ie code that forges pointers, violates access rights on objects, or attempts to change object type.

The bytecode verifier makes four passes on the code in a program. It ensures that the code follows the JVM specifications and does not violate system integrity. If the verifier completes all four passes without returning an error message, then the following is ensured. The classes follow the class file format of the JVM specification

- There are no access restriction violations
- The code causes no operand stack overflows or underflows
- No illegal data conversions

Java Architecture- (Contd...)

 **Garbage Collection** : As the objects are dynamically allocated memory by using new operator, they must be released for later reallocation.

Java handles de-allocation automatically.

This technique is called Garbage Collection.

Finalize : Finalize method is used when certain objects are required to perform some action when they are destroyed.

By using this method, we can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector



Copyright © 2005, Infosys
Technologies Ltd

18

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

When no references to an object exist, that object is assumed to be no longer needed and the memory occupied by the object can be reclaimed. This is done by Garbage Collection. Garbage Collection occurs sporadically during the execution of the program.

when certain objects are required to perform some action when they are destroyed then those actions can be added by defining finalize() method. Java Run time calls that method whenever it is about to recycle an object of that class.

The GC runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. Right before an object is freed, the Java Run time calls the finalize() method on that object.

Installing and using Java

Before we begin, something on installation

- Java 2 SDK (v1.4 or higher)
- Can be downloaded freely from <http://java.sun.com>
- Also available in the intranet

Setting Environment variables for Java

Environment Variable:

- A variable that describes the operating environment of the process
- Common environment variables describe the home directory, command search path etc.



Copyright © 2005, Infosys
Technologies Ltd

19

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

For executing Java Programs Java 2 SDK Version 1.4 or higher Version needs to be installed.

Java 2 SDK Can be downloaded freely from <http://java.sun.com>

Then some Environmental variables have to be set.

Environmental variable is a variable that describes the operating environment of the process

Common environment variables describe the home directory, command search path etc.

Environment variables used by JVM

• **JAVA_HOME:** Java Installation directory

- This environment variable is used to derive all other env. variables used by JVM
- In Windows: `set JAVA_HOME=C:\jdk1.4.3`
- In UNIX: `export JAVA_HOME=/var/usr/java`

• **CLASSPATH**

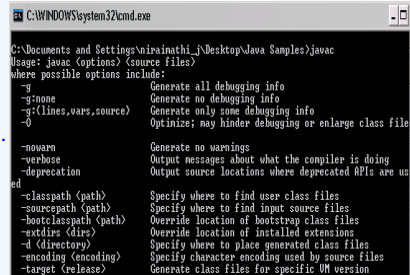
- In Windows:
 - `set PATH=%PATH%;%JAVA_HOME%\lib\tools.jar;`
- In UNIX:
 - `set PATH=$PATH:$JAVA_HOME/lib/tools.jar;`

• **PATH**

- Path variable is used by OS to locate executable files
- In Windows: `set PATH=%PATH%;%JAVA_HOME%\bin`
- In UNIX: `set PATH=$PATH:$JAVA_HOME/bin`

• This approach helps in managing multiple versions of Java – Changing JAVA_HOME will reflect on CLASSPATH and PATH as well

- Set these environment variables on the command prompt and type 'javac'
 - Displays all the options of using 'javac'



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\virainathi\Desktop\Java Samples>javac
Usage: javac [options] [source files]
where possible options include:
-g             Generate all debugging info
-g:none        Generate no debugging info
-g:lines,vars,source  Generate only some debugging info
-O             Optimize; may hinder debugging or enlarge class file
-nowarn        Generate no warnings
-verbose       Output messages about what the compiler is doing
-deprecation   Output source locations where deprecated APIs are used
-classpath <path> Specify where to find user class files
-sourcepath <path> Specify where to find input source files
-bootclasspath <path> Override location of bootstrap class files
-extdirs <dirs> Override location of installed extensions
-d <directory> Specify where to place generated class files
-encoding <encoding> Specify character encoding used by source files
-target <release> Generate class files for specific VM version
```



Copyright © 2005, Infosys Technologies Ltd

20

ER/CORP/CRS/LA10/003
Version 1.00

Infosys

Do the Environmental variable settings by typing the given commands in the Command Prompt and Type 'javac'. It will display all the options of using javac as shown in the diagram. If it says bad command or file name then check the path setting.

Source File Layout - Hello World

④ We will have the source code first

④ Type this into any text editor

```
public class HelloWorldApp {  
    public static void main(String[] args){  
        System.out.println("Hello World!");  
    }  
}
```

④ Save this as HelloWorldApp.java

④ **Important :**

- Take care!! cAsE of file name matters



Copyright © 2005, Infosys
Technologies Ltd

21

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Let us start writing the first Java Program.

A Java program file is an ordinary text file with “. java” extension. Java source files are created in a plain text editor, or in an editor that can save files in plain ASCII without any formatting characters. It contains one or more class definitions. In Java all code must reside inside a class. Even the main method has to be in a class.

Here in the example there is a class named HelloWorldApp which contains the main method, which is the starting point of execution of the program.

In this example main method prints “Hello World” text.

Type the above code in any editor e.g notepad and save it as HelloWorldApp.java

The name of the file must be the same as the name of the public class [i.e HelloWorldApp].

The extension must be .java

While saving the file using the text editor, make sure that the file is saved with an extension “.java” and NOT as “.java.txt”.

The source file layout

There are three *top-level* elements that may appear in a file. None of these elements is required. If they are present, they must appear in the following order:

To Compile

- 1. Open a command prompt
- 2. Set Environment variables (explained earlier)
- 3. Go to the directory in which you have saved your program.
- 4. Type **javac HelloWorldApp.java**
 - If it says bad command or file name then check the path setting
 - If it does not say anything, and you get the prompt, then the compilation was successful.



Copyright © 2005, Infosys
Technologies Ltd

22

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Compiling a Program

To compile the program, Open a command prompt , Go to the directory in which you have saved your program execute the compiler, **javac**, specifying the name of the source file on the command line.

For E.g: **javac HelloWorldApp.java**

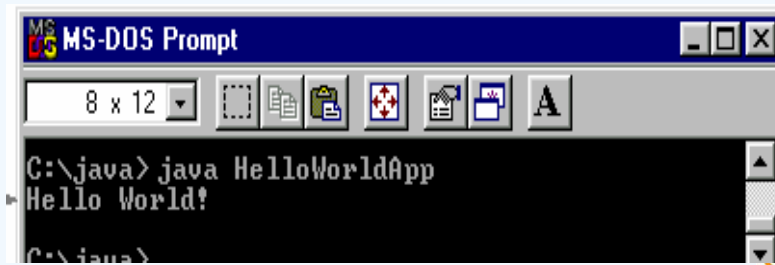
The javac compiler creates a file called **HelloWorldApp.class**. This **.class** file is nothing but the bytecode for the program. The bytecode is the intermediate representation of your program that contains instructions the Java interpreter will execute. Thus, the output of **javac** is not the code that can be directly executed.

To execute

- Type in the command prompt

`java HelloWorldApp`

- The result



```
MS-DOS Prompt
8 x 12
C:\java> java HelloWorldApp
Hello World!
C:\java>
```



Copyright © 2005, Infosys
Technologies Ltd

23

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

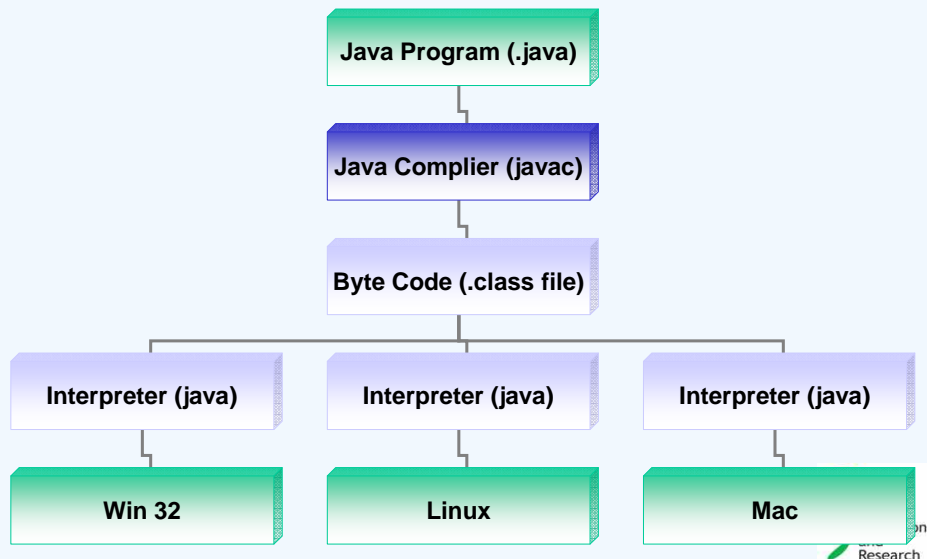
Executing a Program

To actually run the program, you must use the Java interpreter, called **java**. To do so, pass the class name at the command-line:

For Example: Type “java HelloWorldApp”

When the program is run, you get the output as shown in the Figure.

Compilation & Execution



Copyright © 2005, Infosys
Technologies Ltd

24

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®
Research

Have a look on the sequence of steps in compiling and executing a java program.

Best Practices

- One .java file must contain only one class declaration
- The name of the file must always be same as the name of the class
- Stand alone Java program must have a public static void main defined
 - it is the starting point of the program.
 - Not all classes require public static void main
- Code should be adequately commented
- Must follow indentation and coding standards



Copyright © 2005, Infosys
Technologies Ltd

25

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Let us see some best practices to be followed while writing Java Programs

One .java file must contain only one class declaration

The name of the file must always be same as the name of the class

Stand alone Java program must have a public static void main defined

Code should be adequately commented

Must follow indentation and coding standards



Basic Java Programming

Module – 3 : Basic Constructs



We shall now move on to the module on Basic Constructs of Java.

Learning Outcomes

After completion of the module you will be able to

- Understand the basic constructs in Java
- Know how to use the modifiers available in Java
- Work with Constructors
- Understand the concept of Method Overloading
- Program using the concept of inheritance.
- Understand the concept of inner classes.



Copyright © 2005, Infosys
Technologies Ltd

27

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

After completion of the module you will be able to

Understand the basic constructs in Java

Know how to use the modifiers available in Java

Work with Constructors

Understand the concept of Method Overloading

Program using the concept of inheritance and

Understand the concept of inner classes.

Java Keywords – (For Reference Only)

 The reserved keywords defined in the Java language

abstract	const	finally	implements	public	this
boolean	continue	for	instanceof	throw	transient
break	float	if	null	short	void
byte	default	import	int	super	volatile
case	do	false	return	switch	while
catch	double	interface	package	synchronized	
char	else	long	private	static	
class	extends	goto	protected	try	
true	final	new	native	throws	



Copyright © 2005, Infosys
Technologies Ltd

28

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Have a look on the keywords available in Java.

Data Types in Java

☛ Java is a Strongly typed language

- What is typecasting?
- Unlike C, at runtime type checking is strictly enforced
- Impossible to typecast incompatible types

☛ Two types of variables

- Primitive type
- Reference type



Copyright © 2005, Infosys
Technologies Ltd

29

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Java is a strongly typed language, which means that all variables must first be declared before they can be used.

The basic form of variable declaration is “Variable Type followed by variable name”. For example “ int grossSalary;”

Doing so tells your program that the variable named "grossSalary" exists and holds numerical data[i.e integer value]. A variable's data type determines the values it may contain, plus the operations that may be performed on it.

The data types are of two types. They are primitive data types and reference data types.

Primitive Data Types are predefined by the language and is named by a reserved keyword. Example: int, float etc.

Reference Data types are often referred as non-primitive data types. Example: Objects and Arrays. They are called as reference data types because, they are handled "by reference"--in other words, the address of the object or array is stored in a variable of reference data type and not the value directly. In contrast primitive types are handled "by value"--the actual primitive values are stored in variables of primitive data types .

Type Casting is the process of assigning a value of one type to the variable of another type. While assigning the value of var1 to var2, if both var1 and var2 are of different data types and if the data types of var1 and var2 are compatible and var2 data type is larger than var1 data type then java will perform the value conversion automatically else it has to be done manually. We will be discussing more on Type Casting in the later part of the presentation.

Primitive Data Types in Java

Integer data types

- byte (1 byte)
- short (2 bytes)
- int (4 bytes)
- long (8 bytes)

Floating Type

- float (4 bytes)
- double (8 bytes)

Textual

- char (2 bytes)

Logical

- boolean (1 byte) (true/false)

Notes:

- All numeric data types are signed
- The size of data types remain the same on all platforms (standardized)
- char data type in Java is 2 bytes because it uses UNICODE character set. By virtue of it, Java supports internationalization
- UNICODE is a character set which covers all known scripts and language in the world



Copyright © 2005, Infosys
Technologies Ltd

30

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

This shows the list of primitive data types available in Java.

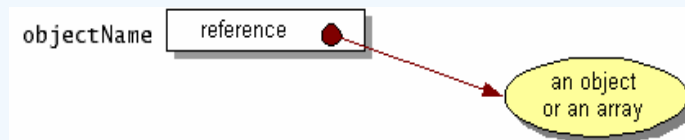
Totally there are 8 primitive data types.

Some important points to be noted are,

1. All numeric data types are signed
2. The size of data types remain the same on all platforms
3. char data type in Java is 2 bytes because it uses UNICODE character set which covers all known scripts and language in the world. By virtue of it, Java supports internationalization

Reference Types in Java

- ④ Objects, Arrays are accessed using *reference variables* in Java
- ④ A reference variable is similar to a pointer (stores memory address of an object)
- ④ Java does not support the explicit use of addresses like other languages
- ④ Java does not allow pointer manipulation or pointer arithmetic
- ④ A reference type cannot be cast to primitive type
- ④ A reference type can be assigned 'null' to show that it is not referring to any object
 - 'null' is a keyword in Java



Copyright © 2005, Infosys
Technologies Ltd

31

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

As discussed earlier, Reference Data types are non-primitive data types. Example: Objects and Arrays. They are called as reference data types because, they are handled "by reference"--in other words, the address of the object or array is stored in a variable and not the value directly.

Objects and Arrays are accessed using *reference variables* in Java

A reference variable is similar to a pointer in C

Java does not allow pointer manipulation or pointer arithmetic

A reference type can be assigned 'null' to show that it is not referring to any object

Variables in Java : Assignment & Declaration

- Using primitive data types is similar to other languages

```
int count;  
int max=100;
```

- In Java variables can be declared anywhere in the program

```
for (int count=0; count < max; count++) {  
    int z = count * 10;  
}
```

- In Java, if a local variable is used without initializing it, the compiler will show an error

- Variables must have a type

- Variables must have a name

- Assigning a value to the variable

```
count =10;
```

- Declaration and initialization may be combined in a single step

```
int count =10;
```

int count

name

type

BEST PRACTICE: Declare a variable in program only when required. Do not declare variables upfront like in C.



Copyright © 2005, Infosys
Technologies Ltd

32

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

In any Java Program, the variable is the elementary unit of storing data. A variable can be declared by using an identifier i.e the variable name and a data type. You can also initialize a variable when it is declared.

In Java, before you use any variable, it should be declared. At the same time, In Java, if a local variable is used without initializing it, the compiler will show an error. Other variables such as member variables etc are assigned with the default values automatically.

According to Java coding standards, the class name should always start with an upper case character. In case there is more than a word, each word should start with an uppercase letter and there should not be any _ in between the words. The name of a variable should be in lower case. In case there is more than a word, the first letter of each word should be in upper case, starting from the second word

The best practice to be followed in declaring the variable is, **Declare a variable in program only when required. Do not declare variables upfront like in C.**

Typecasting of Primitive Data Types

- ④ Automatic, non-explicit type changing is known as **Conversion**

- Variable of smaller capacity can be assigned to another variable of bigger capacity

```
int i = 10;  
double d;  
d = i;
```

- ④ Whenever a larger type is converted to a smaller type, we have to explicitly specify the **type cast operator**

```
double d = 10  
int i;  
i = (int) d;
```

Type cast operator

- This prevents **accidental loss** of data



Copyright © 2005, Infosys
Technologies Ltd

33

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Type Conversion and Casting

The process of assigning a value of variable of one type to a variable of another type is called casting. There are two types of type casting.

1. Automatic, non-explicit type changing: It is known as Implicit Conversion. While assigning the value of var1 to var2, if both var1 and var2 are of different data types and if the data types of var1 and var2 are compatible and var2 data type is larger than var1 data type then java will perform the value conversion automatically. The idea is Variable of smaller capacity can be assigned to another variable of bigger capacity. For example when the integer value of 10 is assigned to a double variable d, then java will automatically convert 10 to 10.0 and store it in d.
2. Explicit type changing: If we want to assign the value of var1 to var2 and if the data types of var1 and var2 are incompatible then java will not carry the type conversion automatically. The programmer has to explicitly do the conversion by specifying a type cast operator. The general form is (target-type) followed by value. Where the "target-type" specifies the desired type to convert the value to.

Widening and Narrowing

Widening conversions permitted in Java

- From a byte to a short, an int, a long, a float, or a double
- From a short to an int, a long, a float, or a double
- From an int to a long, a float, or a double
- From a long to a float, or a double
- From a float to a double

Char ----->|

Int -----> long -----> float -----> double

Byte -----> Short ->|

Narrowing Conversions allowed

- From a byte to a char
- From a short to a byte or a char
- From a char to a byte or a short
- From an int to a byte, a short, or a char
- From a long to a byte, a short, a char, or an int
- From a float to a byte, a short, a char, an int, or a long
- From a double to a byte, a short, a char, an int, a long, or a float



Copyright © 2005, Infosys
Technologies Ltd

34

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Type conversion that happens when Variable of smaller capacity assigned to variable of bigger capacity forms widening.

Type conversion that happens when Variable of larger capacity assigned to variable of smaller capacity forms Narrowing.

Narrowing is usually done through explicit cast.

Here are the permissible tasks that can be achieved through widening and narrowing.

Operators and Assignments

Arithmetic Operators:

OPERATOR	OPERATION
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Incrément (Unary)
+=	Addition assignment
- =	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement



Copyright © 2005, Infosys
Technologies Ltd

35

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Operators

Java provides a fully featured set of operators. Most of the operators are evaluated from left to right. For assignments associativity is from right-to-left. Operators are divided into groups such as, Arithmetic operators, Bitwise operators, Relational Operators, Boolean Logical Operators and the Ternary operator.

Arithmetic operators are used in mathematical expressions. Arithmetic operators defined in java are listed in the given table. The operands of arithmetic operators must be numeric type. Others except char type operands are not allowed. Char type operands are allowed because in java Char type is a sub set of int datatype.

Operators and Assignments (Contd..)

• The Bitwise operators:

OPERATOR	OPERATION
~	Bit wise Unary NOT
&	Bit wise And
	Bit wise OR
^	Bit wise exclusive OR
>>	Shift right (Performs a signed shift)
>>>	Shift right zero fill (Performs a unsigned shift)
<<	Shift left
&=	Bit wise AND assignment
=	Bit wise OR assignment
^=	Bit wise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment



Copyright © 2005, Infosys
Technologies Ltd

36

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Bitwise operators defined in java are applied to the bits with in an integer types such as long, int, short, char and byte. Bitwise **operators defined in java are listed in the given table**

Operators and Assignments (Contd..)

Relational Operators:

OPERATOR	OPERATION
==	Equal to
!=	Not Equal to
>	Greater than
<	Lesser than
>=	Greater than or equal to
<=	Less than or equal to



Copyright © 2005, Infosys
Technologies Ltd

37

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Relational Operators defined in java are listed in the given table

The outcome of relational operation is a Boolean value. The relational operators are most frequently used in the expressions that control the if statement and the various loop statements.

Operators and Assignments (Contd..)

Boolean Logical operators:

OPERATOR	OPERATION
&	Logical AND
	Logical OR
^	Logical XOR (exclusive OR)
	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else



Copyright © 2005, Infosys
Technologies Ltd

38

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Boolean Logical operators operate only on boolean operands. They combine two boolean values to form a resultant boolean value.

Boolean Logical operators defined in java are listed in the given table

Let us see the operations of Logical AND and Short Circuit AND operators

The logical AND checks the condition of the first operand and also the second operand. But the short-circuit AND checks the second operand only if the first operand returns true, else it returns false. The difference between the logical OR and short-circuit OR is similar to this.

Operators and Assignments (Contd..)


The ternary operator: ?

Example

```
int greatestNum=(num1>num2)?num1:num2;
```

Syntax

Expression1? Expression2:Expression3

-  Expression1 can be any expression that evaluates a boolean value. If Expression1 is true, then Expression2 is evaluated; otherwise, Expression3 is evaluated. Both Expression2 and Expression3 are required to return the same type, which can be void.



Copyright © 2005, Infosys
Technologies Ltd

39

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The ternary operator

Java includes a special ternary (i.e three way) operator that can replace certain types of if-then-else statements. It takes the form of

expression1?expression2:expression3;

Expression1 can be any expression that evaluates a boolean value. If expression1 is true, then expression2 is evaluated; otherwise, expression3 is evaluated. Both expression2 and expression3 are required to return the same type, which can even be void.

Access Modifiers – private and public

- Data members are always kept **private**
 - It is accessible only within the class
- The methods which expose the behavior of the object are kept **public**
 - However, we can have helper methods which are private
- Key features of object oriented programs
 - encapsulation (binding of code and data together)
 - State (data) is hidden and Behavior (methods) is exposed to external world
- Access modifiers (public, private etc) will be covered in more details in the later slides



Copyright © 2005, Infosys
Technologies Ltd

40

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

How a member can be accessed is determined by the access specifier, that modifies its declaration.

When a member of a class is modified by the public specifier, then that member can be accessed by any other code.

When a member of a class is specified as private, then that member can only be accessed by other members of its class.

When no access specifier is used, then by default the member of a class is public within its own package, but cannot be accessed outside of its package.

Data members are always kept **private i.e** It is accessible only within the class

The methods which expose the behavior of the object are kept **public**. However, we can have helper methods which are private

Access modifiers (public, private etc) will be covered in more detail in the later slides

Other Modifiers

Static

- For class variables
- For methods
- For a block of code

Final

- For class level/local variables
- For methods
- For class



Copyright © 2005, Infosys
Technologies Ltd

41

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Sometimes we want to define a class member that will be used independently of any object of that class. To create such a member, precede its declaration with keyword “static”. When a member is declared static, it can be accessed before any objects of its class are created and without reference to any object. Both variables and methods can be declared as static.

main() is declared as static because it must be called before any objects exist.

Instance variables declared as static are global variables. When objects of its class are declared, no copy of static variable is made. All instances of the class share the same static variable.

Methods declared as static have some restrictions:

They can only call other static methods

They must only access static data.

They can't refer to this or super in any way.

To do computation to initialize the static variables, we can declare a static block that gets executed exactly once, when the class is first loaded.

Static methods and variables can be used independently of any object via class names

A variable can be declared as final. Doing so prevents its contents from being modified. This means we must initialize a final variable when it is declared. Variables declared as final do not occupy memory on a per-instance basis. Thus, a final variable is essentially a constant.

To disallow a method from being overridden, specify final as a modifier at the start of its declaration. Methods declared as final can not be overridden.

To disallow a class from being inherited, precede final as the modifier in the class declaration. Declaring a class as final declares all of its methods as final too.

You can not declare an abstract class as final.

Writing a class in Java

```
public class Student {  
    private int rollNo;  
    private String name;
```

Data Members
(State)

```
    Student(){  
        //initialize data members  
    }  
    Student(String nameParam){  
        name = nameParam;  
    }  
    public int getrollNo (){  
        return rollNo;  
    }  
}
```

Constructor

Method (Behavior)

The main method may or may not be present depending on whether the class is a starter class



Copyright © 2005, Infosys
Technologies Ltd

42

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The class defines a new data type and this data type is used to create objects of that type. “class is an template for an object”.

Class is like a container of Data members and Methods. A sample class is given here.

Where, student is the name of the class. Rollnumber and name forms the data members and getrollnumber() forms the method that access the datamember. A simple class definition will take the form of

```
class classname  
{  
    variable1;  
    variable2;  
    Method1()  
{  
    // Body of Method1.  
}  
    Method2(parameters)  
{  
    //Body of Method2.  
}  
}
```

Usually, data members will be kept private and methods accessing the data members will be kept as public.

In student class two more methods are also defined Student() and Student(String nameParam).

These methods are called as constructors, which are called when the objects are created.

Creating Objects in Java

- The new operator creates the object and returns a reference to it
- Memory allocation of objects happens in the heap area
- Reference returned can be stored in reference variables
- E.g. :

```
Student obj1;
```

obj1 is reference variable

```
obj1 = new Student();
```

Or

```
Student obj2 = new Student("Jack");
```

new keyword creates an object and returns a reference to it



Copyright © 2005, Infosys Technologies Ltd

43

ER/CORP/CRS/LA10/003
Version 1.00

Infosys

The **new** keyword is used to create new objects in Java. Creating an object involves allocating memory to the object that is created.

If we just declare `Student obj1`, only a reference called `obj1` of type `Student` is created. No memory is allocated for the same. Its only when we use the new operator, memory is allocated, and the reference points to the memory that is allocated to the object.

How to use the new operator? There are two ways.

1. `Student obj1; obj1=new Student();` (or)
2. `Student obj1=new Student();`

When these statements are executed memory for the object is allocated, and the reference variable `obj1` points to the memory that is allocated to the object. i.e in the case of `Student obj1`; `obj1` will have the value null and once the new operator is applied on `obj1`, `obj1` will have the address allocated to the object.

Constructors

- A constructor is a special method that is used to initialize a newly created object
- Called just after the memory is allocated for the object
- Can be used to initialize the objects to required or default values at the time of object creation
- It is not mandatory for the coder to write a constructor for the class
- When writing a constructor, remember that:
 - it has the same name as the class
 - it does not return a value not even `void`
 - It may or may not have parameters (arguments)
- If no user defined constructor is provided for a class, compiler initializes member variables to its default values. Examples:
 - numeric data types are set to 0
 - char data types are set to null character(`'\0'`)
 - reference variables are set to null



Copyright © 2005, Infosys
Technologies Ltd

44

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

A constructor is a special method that is used to initialize a newly created object.
Called just after the memory is allocated for the object
Can be used to initialize the objects to required or default values at the time of object creation
It is not mandatory for the coder to write a constructor for the class
When writing a constructor, remember that:
 it has the same name as the class
 it does not return a value not even `void`
 It may or may not have parameters (arguments)
If no user defined constructor is provided for a class, compiler initializes member variables to its default values. Examples:
 numeric data types are set to 0
 char data types are set to null character(`'\0'`)
 reference variables are set to null

Constructors (Contd...)

```
public class Student {  
    private int rollNo;  
    private String name;  
  
    Student(){  
        rollNo=1;  
        name="abi";  
    }  
    Student(String nameParam){  
        name = nameParam;  
    }  
    public int getrollNo (){  
        return rollNo;  
    }  
}
```



Copyright © 2005, Infosys
Technologies Ltd

45

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

In this example **Student()** and **Student(String nameParam)** are constructors. When they will be invoked?

At the time of object creation, soon after the memory for the objects is allocated.

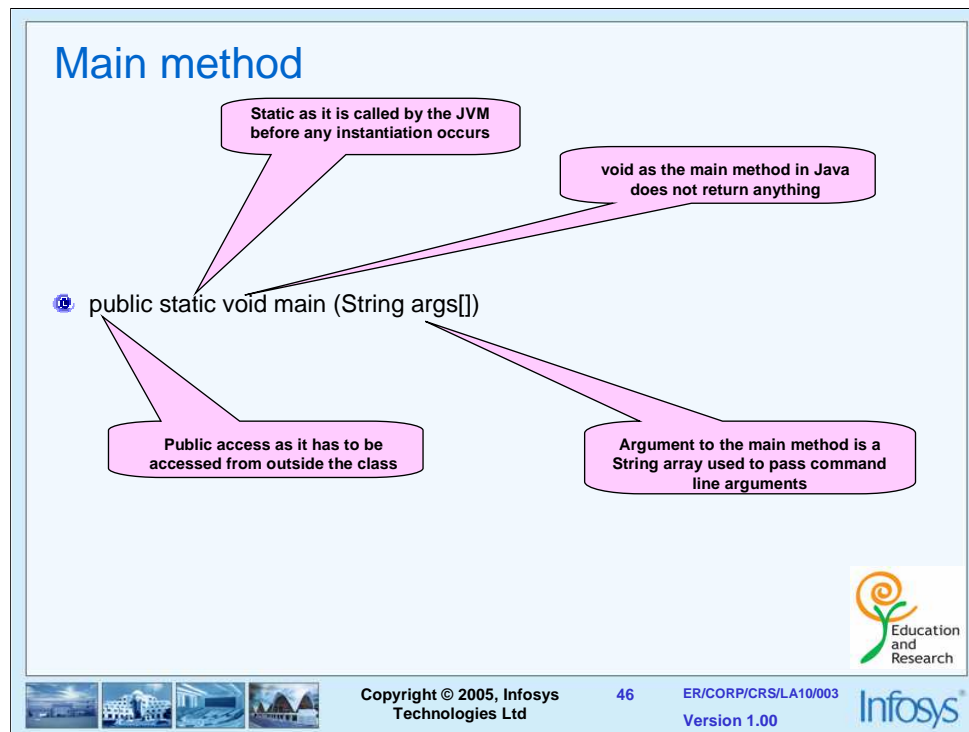
Here there are two constructors.

Consider the Statement, `Student obj1=new Student();`

What will happen when the statement gets executed? a new object obj1 gets created and `Student()` constructor gets invoked.

Consider the Statement, `Student obj1=new Student("abi");`

What will happen when the statement gets executed? a new object obj1 gets created and `Student(String nameParam)` constructor gets invoked.



Let us now analyse the Main method's signature.

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

It is the starting point of execution of the program.

It uses public access modifier, so that it can be accessed by Java Interpreter.

It is defined with static keyword so that it can be accessed by Java Interpreter before creating object for that class.

Void specifies that the return type of main method is nothing.

String args[] here args form the array variable that holds the command line arguments. i.e inputs given at the time of execution along with java command.

For example "java CmpNum 28 29".

Where java is the command to execute the program

CmpNum is the name of the .java file

28 & 29 are command line inputs.

They are stored as args[0]="28" and args[1]="29"

Point to be noted here is all the inputs are stored as strings.

Scope of Variables in Java

- **SCOPE** : Scope determines what objects are visible to other parts of the program.
- A block defines a scope
- Each time we start a new block, we create a new scope.
- Defining a variable within a scope mean localizing that variable and protecting it from unauthorized access / modification
- **Two major scopes in Java – CLASS and METHOD**
- Variables defined inside a method including parameters are visible only within the method.
- Instance variables have the scope of class in which they are defined.
- Scope can be nested



Copyright © 2005, Infosys
Technologies Ltd

47

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Scope determines what objects are visible to other parts of the program.

A block defines a scope

Each time we start a new block, we create a new scope.

Defining a variable within a scope mean localizing that variable and protecting it from unauthorized access / modification

There are two major scopes in Java – CLASS and METHOD

Variables defined inside a method including parameters are visible only within the method.

Instance variables have the scope of class in which they are defined.

Scope can be nested.

Control Structures in Java

Similar to the “C” Programming Language

Conditionals

- if-else, else if statements
- switch case statements

Loops

- for
- while
- do-while



Copyright © 2005, Infosys
Technologies Ltd

48

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Java has all the control structures available in C.

In Java, the if condition works in a slightly different manner from C.

In C, any non-zero is treated as a true condition and hence an if like “if(x=9)” would be evaluated as true.

But such a thing is not possible in java. Java would expect a strictly Boolean value true or false in an if statement.

This is also true for while, do-while etc..

Arrays

- An array is a data structure which defines an ordered collection of a fixed number of homogeneous data elements
- The size of an array is fixed and cannot increase to accommodate more elements
- In Java, arrays are objects and can be of primitive data types or reference types
- All elements in the array must be of the same data type



Copyright © 2005, Infosys
Technologies Ltd

49

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

An array is a data structure which defines an ordered collection of a fixed number of homogeneous data elements

The size of an array is fixed and cannot increase to accommodate more elements

In Java, arrays are objects and can be a collection of primitive data types or reference types

All elements in the array must be of the same data type

Arrays[Contd...]

Declaring Arrays Variables

```
<elementType>[] <arrayName>;
```

or

```
<elementType> <arrayName>[];
```

where <elementType> can be any primitive data type or reference type

Example:

```
int IntArray[];
```

```
Pizza[] mediumPizza, largePizza;
```



Copyright © 2005, Infosys
Technologies Ltd

50

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Declaration of Arrays:

Arrays can be declared in two ways as shown here.

```
<elementType>[] <arrayName>;
```

or

```
<elementType> <arrayName>[];
```

where <elementType> can be any primitive data type or reference type

For Example:

int IntArray[]; declares IntArray which is an integer Array

Pizza[] mediumPizza, largePizza; declares mediumPizza and LargePizza arrays which can hold the elements of type Pizza.

Arrays[Contd...]

Constructing an Array

```
<arrayName> = new <elementType>[<noOfElements>];
```

Example:

```
IntArray = new int[10];  
mediumPizza = new Pizza[5];  
largePizza = new Pizza[2];
```

Declaration and Construction combined

```
int IntArray = new int[10];  
Pizza mediumPizza = new Pizza[5];
```



Copyright © 2005, Infosys
Technologies Ltd

51

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

As you are aware Arrays are reference data types. So when we declare array only reference will be created. Actual Array object will be created, when you use new keyword. Use the given syntax for creating Array object.

We can also combine declaration and construction of array in a single step.

Arrays[Contd...]

Initializing an Array

```
<elementType>[] <arrayName> = {<arrayInitializerCode>;
```

Example:

```
int IntArray[] = {1, 2, 3, 4};  
char charArray[] = {'a', 'b', 'c'};  
Object obj[] = {new Pizza(), new Pizza()};  
String pets[] = {"cats", "dogs"};
```



Copyright © 2005, Infosys
Technologies Ltd

52

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Array initialization can be done as shown here.

this keyword

- **this** keyword can be used inside any method to refer to the current object.
- Syntax is
 - this.member variable
- this is always a reference to the object on which the method was invoked.
- E.g:

```
class Rectangle{  
    int length;  
    int width;  
    Rectangle(int length,int width){  
        this.length=length;  
        this.width=width;  
    }  
    ...  
}
```



Copyright © 2005, Infosys
Technologies Ltd

53

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Sometimes, we can have local variables, including formal parameters to methods, which overlap with the names of the class's instance variables.

When a local variable has the same name as an instance variable, the local variable hides the instance variable.

To get an access to instance variables, keyword *this* is used.

For example consider class `Rectangle`, with instance variables `int length` and `int width` and with a constructor `Rectangle(int length,int width)`. The constructor has to assign the values in its formal parameters to the instance variables `length` and `width`. If it uses `length` and `width` it will refer to the formal parameters only. So it will do the assignments with the help of *this* operator as,

```
this.length=length;  
this.width=width;
```

Method overloading (1 of 2)

- More than one method within the same class having the **same name but differing in method signature** are termed overloaded methods
- Calls to overloaded methods will be resolved during **compile time**
 - Also known as **static polymorphism**
- Argument list could differ via:
 - No of parameters
 - Data type of parameters
 - Sequence of parameters

• Ex.

```
void add (int a, int b)
void add (int a, float b)
void add (int a, float b)
void add (int a, int b, float c)

void add (int a, float b)
int add (int a, float b)
```

Overloaded methods

Not overloading. Compiler error.



Copyright © 2005, Infosys Technologies Ltd

54

ER/CORP/CRS/LA10/003
Version 1.00

Infosys

Overloading is used while creating several methods that perform closely related functions under different conditions **within a class**. The compiler will treat each of these methods as different method.

More than one method within the same class having the **same name but differing in method signature** are termed overloaded methods

Calls to overloaded methods will be resolved during **compile time**

It is Also known as **static polymorphism**

Argument list could differ via:

No of parameters

Data type of parameters

Sequence of parameters

The method return type has no effect in method overloading.

A class having two or more constructors is a special case of method overloading which is constructor overloading.

Inheritance

- When “is a” relationship exists between two classes, we use inheritance
- The parent class is termed **super** class and the inherited class is the **sub** class
- The keyword **extends** is used by the sub class to inherit the features of super class

```
class Person{  
    /*attributes and functionality of Person  
    defined*/  
}  
class Student extends Person{  
    /*inherits the attributes and functionalities  
    of Person and in addition add its own  
    specialties*/  
}
```

Person is more generalised

Student “is a” Person and is more specialized

- A subclass **cannot access the private members** of its super class.
- Inheritance leads to **reusability** of code



Copyright © 2005, Infosys Technologies Ltd

55

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

When “is a” relationship exists between two classes, we use inheritance

The parent class is termed **super** class and the inherited class is the **sub** class

The keyword **extends** is used by the sub class to inherit the features of super class

Inheritance leads to **reusability** of code

Using Inheritance, we can create a general class that defines the traits common to a set of related items. This class can then be inherited by other specific classes, each adding the things that are unique to it.

A sub class is the specialized version of a super class.

It inherits all of the instance variables and methods defined by the super class and adds its own, unique elements.

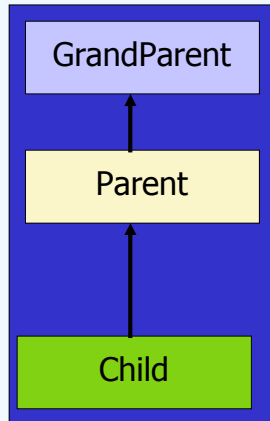
Although a sub class includes all of the members of its super class it can not access those members of the super class that have been declared as private.

A major advantage of inheritance is that once we have created a super class that defines the attributes common to a set of objects, it can be used to create any number of more specific subclasses.

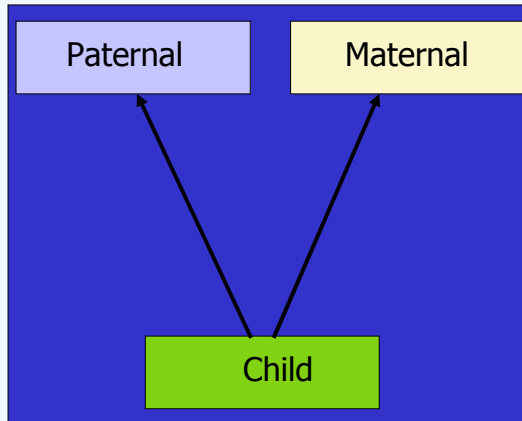
A reference variable of a super class can be assigned to a reference to any sub class derived from that super class

Inheritance

© **Multi-level** inheritance is allowed in Java but not **multiple** inheritance



Allowed in Java



Not Allowed in Java

Education
and
Research



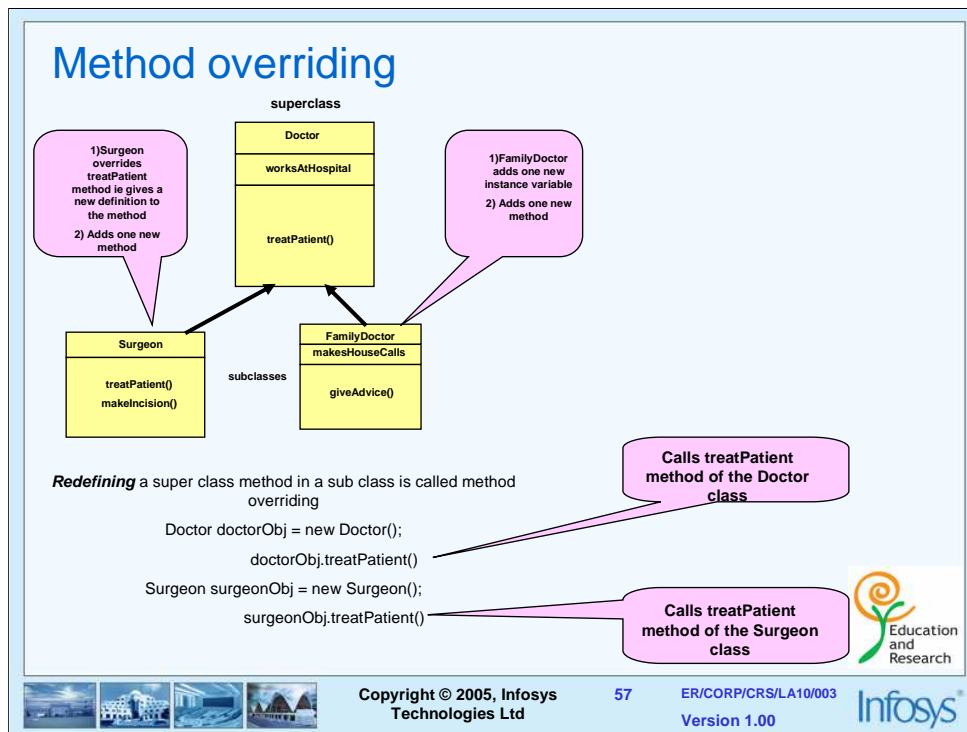
Copyright © 2005, Infosys
Technologies Ltd

56

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Multiple inheritance is not allowed in java.



Overriding

Redefining a super class method in a sub class is called method overriding

Some rules in overriding are

The method signature ie. method name, parameter list and return type have to match exactly

The overridden method can widen the accessibility but not narrow it, ie if it is private in the base class, the child class can make it public but not vice versa

Dynamic Binding

- ❏ Can we do this?
 - ***Doctor obj = new Surgeon();***
- ❏ A reference to a super class can refer to a sub class object
- ❏ Now when we say ***obj.treatPatient()***, which version of the method is called?
 - It calls Surgeon's version of the treatPatient method as the reference is pointing to a Surgeon object
- ❏ If a base class reference is used to call a method, the method to be invoked is decided by the JVM, depending on the object the reference is pointing to
 - For example, even though obj is a reference to Doctor, it calls the method of Surgeon, as it points to a Surgeon object
- ❏ This is decided during run-time and termed ***dynamic*** or ***run-time polymorphism***



Copyright © 2005, Infosys
Technologies Ltd

58

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Can we do this?

Doctor obj = new Surgeon();

A reference to a super class can refer to a sub class object

Now when we say ***obj.treatPatient()***, which version of the method is called?

It calls Surgeon's version of the treatPatient method as the reference is pointing to a Surgeon object

If a base class reference is used to call a method, the method to be invoked is decided by the JVM, depending on the object the reference is pointing to

For example, even though obj is a reference to Doctor, it calls the method of Surgeon, as it points to a Surgeon object

This is decided during run-time and termed ***dynamic*** or ***run-time polymorphism***

So we can say method overriding as an example for runtime polymorphism and method overloading as an example for compile time polymorphism.

super

- What if the treatPatient method in the Surgeon class wants to do the functionality defined in Doctor class and then perform its own specific functionality?
- The treatPatient method in the Surgeon class could be written as:

```
treatPatient(){  
  
    super.treatPatient();  
  
    //add code specific to Surgeon  
  
}
```

This calls the super class version of treatPatient() and then comes back to do the sub-class specific stuff



Copyright © 2005, Infosys Technologies Ltd

59

ER/CORP/CRS/LA10/003
Version 1.00

Infosys

What if the treatPatient method in the Surgeon class wants to do the functionality defined in Doctor class and then perform its own specific functionality?

The treatPatient method in the Surgeon class could be written as shown here. Here **super.treatPatient()** **calls the super class version of treatPatient() and then comes back to do the sub-class specific stuff**

Super has two general forms

1. Which calls super class constructor

Syntax is `super(arg-list);`

`Super()` must always be the first statement executed inside a sub class constructor.

When a sub class calls `super()`, it is calling the constructor of its immediate super class.

2. This second form is most applicable to situations in which member names of a sub class hide members by the same name in the super class.

Syntax is `super. member`

Here member can be either a method or an instance variable.

Polymorphism

STATIC

- ④ **Function Overloading**
 - within same class more than one method having same name but differing in signature
- ④ **Resolved during *compilation time***
- ④ **Return type is not part of method signature**

DYNAMIC

- ④ **Function Overriding**
 - keeping the signature and return type same, method in the base class is redefined in the derived class
- ④ **Resolved during *run time***
- ④ **Which method to be invoked is decided by the object that the reference points to and not by the type of the reference**



Copyright © 2005, Infosys
Technologies Ltd

60

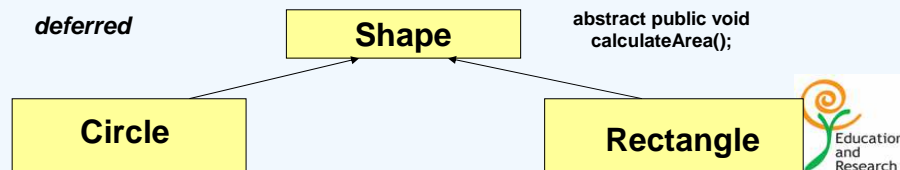
ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Have a look on the comparison between static polymorphism and dynamic or runtime polymorphism.

Abstract

- Consider a scenario where you consider a generalized class Shape which has two subclasses Circle and Rectangle
- Circle and Rectangle have their own way to calculate area
- So, Shape class cannot decide how to calculate the area
 - it only provides the guideline to the child classes that such a method should be implemented in them
- calculateArea() method has been declared abstract in the super class ie the method signature has been provided but the **implementation has been deferred**



Copyright © 2005, Infosys
Technologies Ltd

61

ER/CORP/CRS/LA10/003
Version 1.00



Consider a scenario where you consider a generalized class Shape which has two subclasses Circle and Rectangle

Circle and Rectangle have their own way to calculate area

So, Shape class cannot decide how to calculate the area

it only provides the guideline to the child classes that such a method should be implemented in them

calculateArea() method has been declared abstract in the super class ie the method signature has been provided but the **implementation has been deferred**

Abstract (Contd...)

- The **abstract** keyword can be used for method and class
- An **abstract method** signifies it has no body (implementation), only declaration of the method followed by a semicolon

```
abstract public double calculateArea();
```
- If a class has one or more abstract methods declared inside it, the class must be declared abstract

```
abstract class Shape{  
    //class definition  
}
```
- An abstract class cannot be instantiated ie objects cannot be created from an abstract class
- Reference of an abstract class can point to objects of its sub-classes thereby achieving run-time polymorphism



Copyright © 2005, Infosys
Technologies Ltd

62

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The **abstract** keyword can be used for method and class

An **abstract method** signifies it has no body (i.e implementation), only declaration of the method followed by a semicolon

If a class has one or more abstract methods declared inside it, the class must be declared abstract

An abstract class cannot be instantiated ie objects cannot be created from an abstract class

Reference of an abstract class can point to objects of its sub-classes thereby achieving run-time polymorphism

Abstract (Contd...)

```
abstract class Shape{  
    abstract public void calculateArea();  
    public void setColor(){  
        //code to color the shape  
    }  
}
```

Note:

- An abstract class may also have concrete (complete) methods
- For design purpose, a class can be declared abstract even if it does not contain any abstract methods



Copyright © 2005, Infosys
Technologies Ltd

63

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is an example for abstract class.

An abstract class may also have concrete (complete) methods

For design purpose, a class can be declared abstract even if it does not contain any abstract methods

abstract – Rules to follow (Contd...)

ⓐ The following cannot be marked with “abstract” modifier

- Constructors
- Static methods
- Private methods
- Methods marked with “final” modifier



Copyright © 2005, Infosys Technologies Ltd

64

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

To Conclude..

The abstract modifier can be applied to classes and methods.

Class when declared abstract

- cannot be instantiated.
- Abstract classes provide way to defer implementation to sub classes.

Method when declared Abstract

- No implementation for a method. Only the signature of the method is defined.
- Used to put some kind of compulsion on the class who inherits from this class. i.e., the class who inherits **MUST** provide the implementation of the method else the subclass will also become abstract
- A method can be made abstract to defer the implementation. i.e., when you design the class, you know that there should be a method, but you don't know the algorithm of that method.

A class **must** be declared abstract if any of the following conditions is true:

- The class has one or more abstract methods.
- The class inherits one or more abstract methods (from an abstract parent) for which it does not provide implementations.
- The class declares that it implements an interface but does not provide implementations for every method of that interface (we will soon come to interfaces)

From design perspective, a class may be declared abstract even if it has no abstract methods when it is felt that it is not required to create objects of that class

Constructors, Static methods, Private methods, Methods marked with “final”

modifier should not be defined abstract

Final

- “**final**” modifier has a meaning based on its usage
- For member data and local data in methods
 - Primitives: read-only (constant)
 - Objects: reference is read-only
 - **use all upper case letters by convention**
- For methods: **no overriding**
- For classes: **no inheritance**



Copyright © 2005, Infosys
Technologies Ltd

65

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The final modifier applies to classes, methods, and variables. The meaning of final varies from context to context, but the essential idea is the same.

- A final class may not be subclassed.
- If applied to a variable it means to say that the value is constant.
- A final object reference variable may not be changed but the data owned by the object that is referred to as final can be changed.
- A final method may not be overridden. This is done for security reasons and these methods are used for optimization.

Interfaces in Java

- Let us consider a design option: class Dog has been designed which is a sub-class of Animal
- Dog is also a Pet and it needs the behaviors of a pet
- So, should we place the functionalities of a Pet in class Dog?
- Then what happens when we design a class Cat which extends Animal and is also a Pet ? Should we repeat the Pet functionalities again in Cat class?
- Object oriented feature of reusability of code is not being followed



Copyright © 2005, Infosys
Technologies Ltd

66

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Let us consider a design option: class Dog has been designed which is a sub-class of Animal

Dog is also a Pet and it needs the behaviors of a pet

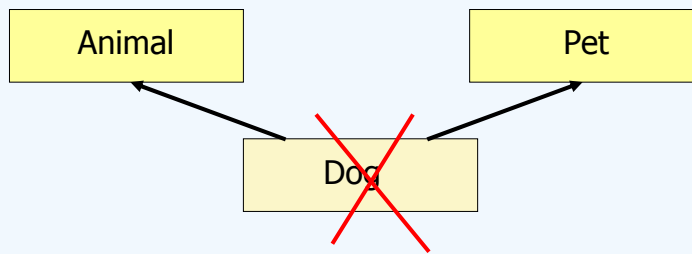
So, should we place the functionalities of a Pet in class Dog?

Then what happens when we design a class Cat which extends Animal and is also a Pet ? Should we repeat the Pet functionalities again in Cat class?

If we do so, Object oriented feature of reusability of code is not being followed

Interfaces in Java (Contd...)

- So what is the solution?
- What if we separate out the functionalities of Pet into a separate class?
- So class Dog now extends from 2 base classes Animal and Pet
- This too will not work as 'Multiple Inheritance' is not supported in Java



Copyright © 2005, Infosys
Technologies Ltd

67

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

So what is the solution?

What if we separate out the functionalities of Pet into a separate class?

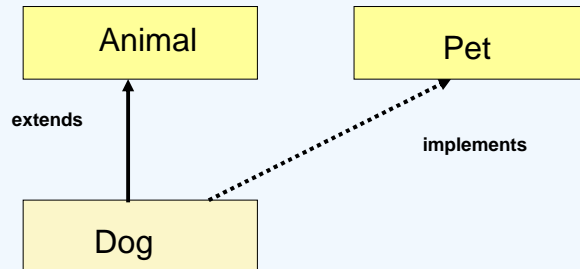
So class Dog now extends from 2 base classes Animal and Pet

This too will not work as 'Multiple Inheritance' is not supported in Java

In such case use of interface is the solution.

Interfaces in Java (Contd...)

- Interface can rescue us
- Interface is similar to an abstract class that contains only abstract methods
- The functionalities of Pet can be placed in an interface
- So class Dog now **extends** class Animal and **implements** Pet



Copyright © 2005, Infosys
Technologies Ltd

68

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Interface is similar to an abstract class that contains only abstract methods
The functionalities of Pet can be placed in an interface
So class Dog now **extends** class Animal and **implements** Pet

Interfaces in Java (Contd...)

- All methods in an interface are implicitly **public** and **abstract**
- The class which implements the interface needs to provide functionality for the methods declared in the interface
 - A class needs to be declared abstract if at least one of the methods in the interface is left undefined

```
interface Pet{
    void beFriejndly();
    void play();
}

class Dog extends Animal implements Pet{
    public void beFriendly(){
        //functionality
    }
    public void play(){
        //functionality
    }
    //other functions
}
```



Copyright © 2005, Infosys
Technologies Ltd

69

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

All methods in an interface are implicitly **public** and **abstract**

The class which implements the interface needs to provide functionality for the methods declared in the interface

A class needs to be declared abstract if at least one of the methods in the interface is left undefined

By including the **implements** clause in a class definition, we can Implement an interface.

Interfaces are defined in the same way as a class but with interface keyword instead of class keyword.

Here Pet is an interface.

Class Dog implements Pet interface using the keyword implements.

Interfaces in Java (Contd...)

- An interface may define data members and these are implicitly **public**, **final** and **static**
- An interface cannot have private or protected members
- An interface can extend from one or many interfaces
- A class can extend only one class but implement any number of interfaces

class Person extends LivingBeing implements Employee, Friend

interface RidableAnimal extends Animal, Vehicle



Copyright © 2005, Infosys
Technologies Ltd

70

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

An interface may define data members and these are implicitly **public**, **final** and **static**
An interface cannot have private or protected members
An interface can extend from one or many interfaces
A class can extend only one class but implement any number of interfaces
An interface can be declared as a member of a class or another interface. Such an interface is a member interface or nested interface.

Interfaces in Java (Contd...)

- Ⓢ An interface cannot be instantiated
- Ⓢ **An interface reference can point to objects of its implementing classes**
- Ⓢ The same interface can be implemented by classes from different inheritance trees
 - Example, Parrot 'is a' Bird but also a Pet
 - `class Parrot extends Bird implements Pet { }`
- Ⓢ interface Pet is being used by both Dog and Parrot



Copyright © 2005, Infosys
Technologies Ltd

71

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

An interface cannot be instantiated

An interface reference can point to objects of its implementing classes

The same interface can be implemented by classes from different inheritance trees

Abstract Class vs Interface – design choice

- Use an abstract class when a **template** needs to be defined for a group of sub-classes
 - ensure that no objects need to be created from it
- Use an interface when a **role** needs to be defined for other classes, regardless of the inheritance tree of these classes



Copyright © 2005, Infosys
Technologies Ltd

72

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Use an abstract class when a **template** needs to be defined for a group of sub-classes
ensure that no objects need to be created from it
Use an interface when a **role** needs to be defined for other classes, regardless of the
inheritance tree of these classes

Anonymous Inner Classes

- An inner class is a class defined within another class.
- An inner class is a non-static nested class. It has access to all of the variables and methods of its outer class.
- An inner class is fully within the scope of its enclosing class.
- **ANONYMOUS INNER CLASSES** : An anonymous inner class is a class that is not assigned any name.



Copyright © 2005, Infosys
Technologies Ltd

73

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

An inner class is a class defined within another class.

An inner class is a non-static nested class. It has access to all of the variables and methods of its outer class.

An inner class is fully within the scope of its enclosing class.

An anonymous inner class is a class that is not assigned any name.

Anonymous Inner Classes(Contd..)

```
class Outer {
    int outer_x = 100;
    void test() {
        Inner inner = new Inner();
        inner.display();
    }

    // this is an inner class
    class Inner {
        void display() {
            System.out.println("display : outer_x = " + outer_x);
        }
    }
}

class InnerClassDemo {
    public static void main(String args[]) {
        Outer outer = new Outer();
        outer.test();
    }
}
```



Copyright © 2005, Infosys
Technologies Ltd

74

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is an example for an inner class.

In the above program, an inner class named Inner is defined within the scope of class Outer. So, any code in class Inner can directly

Access the variable outer_x.

It is important to note that class Inner is known only within the scope of class Outer.

The Java compiler generates an error message if any code outside of class Outer attempts to instantiate class Inner.

Anonymous Inner Classes(Contd..)

```
//Anonymous Inner class Demo
import java.applet.*;
import java.awt.event.*;
/*
< applet code = "AnonymousInnerClassDemo" width=200 height=100>
</applet>
*/

public class AnonymousInnerClassDemo extends Applet {
    public void init() {
        addMouseListener(new MouseAdapter() {
            public void mousePressed( MouseEvent me) {
                showStatus("Mouse Pressed");
            }
        }
    }
}
```



Copyright © 2005, Infosys
Technologies Ltd

75

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is an example for Anonymous inner classes.

init() method calls addMouseListener() method. Its argument is an expression that defines and instantiates an anonymous inner class

The syntax new MouseAdapter() {...} indicates to the compiler that the code between the braces defines an anonymous inner class.

And that class extends MouseAdapter. This new class is not named but is automatically instantiated when this expression is executed

Java I/O Basics

- Java programs perform I/O through streams.
- A stream is an abstraction that either produces or consumes information
- Two types of streams:
 - Byte Stream
 - Character Stream
- ❖ In Byte stream at the top are two abstract classes : InputStream and OutputStream
- ❖ In Character Stream at the top are two abstract classes : Reader and Wrier
- ❖ All Java programs automatically import java.lang package
- ❖ In this package a class System is defined that encapsulates several aspects of the run-time environment



Copyright © 2005, Infosys
Technologies Ltd

76

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Java programs perform I/O through streams.

A stream is an abstraction that either produces or consumes information

Two types of streams:

Byte Stream

Character Stream

In Byte stream at the top are two abstract classes : InputStream and OutputStream

In Character Stream at the top are two abstract classes : Reader and Wrier

All Java programs automatically import java.lang package

In this package a class System is defined that encapsulates several aspects of the

run-time environment

System contains three predefined stream variables – in, out, err

These variables are declared as public and static within System. This means that they can be used by any other part of the program.

System.out refers to standard output stream which is an object of type PrintStream

System.in refers to standard input stream which is an object of type InputStream

System.err refers to standard error stream which is also an object of type PrintStream

All these are Byte Streams

Java.io. Package contains lot of I/O classes to support I/O operatiopns.



Basic Java Programming

Module – 4 : Java Class Libraries



We shall now move on to the module on Java Class Libraries.

Learning Outcomes

- After completion of the module you will be able to
 - Understand the concept of packages in Java
 - Create and use your own packages
 - Understand the standard packages available in Java
 - Work with Wrapper Classes
 - Work with Collections framework.



Copyright © 2005, Infosys
Technologies Ltd

78

ER/CORP/CRS/LA10/003
Version 1.00

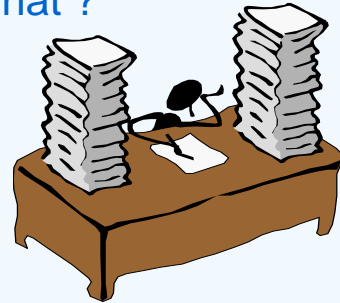
Infosys®

After completion of the module you will be able to

- Understand the concept of packages in Java
- Create and use your own packages
- Understand the standard packages available in Java
- Work with Wrapper Classes
- Work with Collections framework.

Packages in Java – Why ? What ?

- Just think of writing the code from the **scratch**, each time you create an application



- You'll end up spending your precious **time** and **energy** and finally land up with huge amounts of code with you!



Copyright © 2005, Infosys Technologies Ltd

79

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Let us start with Packages in Java. Before looking into the topic in detail let us see what is the need for a package?

Just think of writing the code from the **scratch**, each time you create an application.

You'll end up spending your precious **time** and **energy** and finally land up with huge amounts of code with you!

Packages in Java – Why ? What ? (Contd...)



- **Reusability** of code is one of the most important requirements in the software industry.
- Reusability saves **time, effort** and also ensures **consistency**.

- A class once developed can be reused by any number of programs wishing to incorporate the class in that particular program.



Copyright © 2005, Infosys
Technologies Ltd

80

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Reusability of code is one of the most important requirements in the software industry.
Reusability saves **time, effort** and also ensures **consistency**

A class once developed can be reused by any number of programs wishing to incorporate the class in that particular program.

Concept of Packages

- In Java, the code which can be reused by other programs is put into a **"Package"**.
- A **Package** is a collection of **classes, interfaces** and/or other **packages**.
- Packages are essentially a means of **organizing** classes together as **groups**.



Copyright © 2005, Infosys
Technologies Ltd

81

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

In Java, the code which can be reused by other programs is put into a **"Package"**.

A **Package** is a collection of **classes, interfaces** and/or other **packages**.

The package declaration, if any, must be at the beginning of the source file. You can precede it with white space and comments, but nothing else. Only one package declaration is permitted and it governs the entire source file.

A java program can contain any of the following four parts.

- A single package statement (which is optional)
- Any number of import statements (which is optional)
- Any number of classes and interfaces out of which only one class can be public

The **package** statement defines a namespace in which classes are stored. It is nothing but a directory, in which a class is defined. When the package name is omitted, it is put into the **default package**, which has no name (i.e.. The current directory)

Features of Packages

- ④ **Organize** your classes into smaller units and make it easy to locate and use the appropriate class file.
- ④ Avoid **naming conflicts**
 - Package names can be used to **identify** your classes
- ④ Packages allow you to **protect** your classes, data and methods in a larger way than on a class-to-class basis.



Copyright © 2005, Infosys
Technologies Ltd

82

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

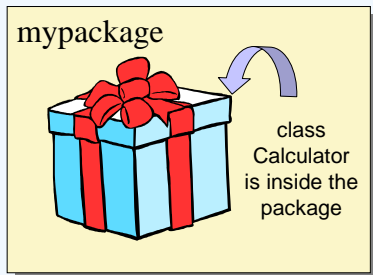
What are the features of packages?

- **Packages Organize** your classes into smaller units and make it easy to locate and use the appropriate class file – i.e., you can split up the classes logically.
- Avoid **naming conflicts** – since classes can be identified through Package name two classes in two different packages can have same name.
- Packages allow you to **protect** your classes, data and methods in a larger way than on a class-to-class basis.

Creating a Package

In Java Packages are created in the following manner :

```
package package_name ;
```



```
package mypackage ;  
  
public class Calculator  
{  
    public int add(int x, int y)  
    {  
        return( x + y ) ;  
    }  
}
```



Copyright © 2005, Infosys
Technologies Ltd

83

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Creating a Package

To create a package, you put a class or an interface in it. To do this, you put a package statement at the top of the source file in which the class or the interface is defined.

For example, the given code appears in the source file Calculator.java, puts the Calculator class in the mypackage package:

The Calculator class is a public member of the mypackage package.

You must include a package statement at the top of every source file that defines a class or an interface that is to be a member of the myPackage package.

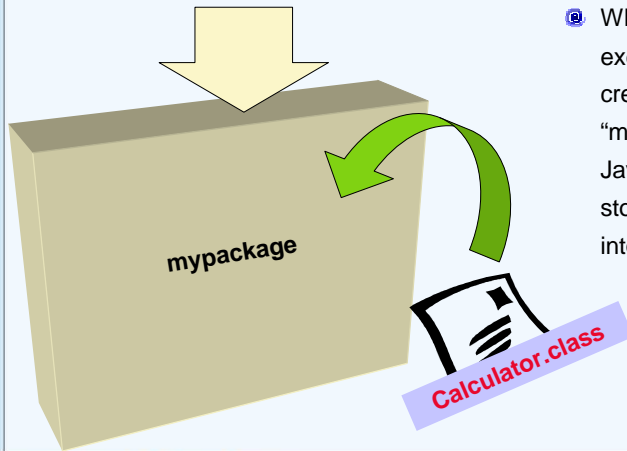
The scope of the package statement is the entire source file, so all classes and interfaces defined in Calculator.java are members of the myPackage package.

If you put multiple classes in a single source file, only one may be public, and it must share the name of the source files base name. Only public package members are accessible from outside the package. If you do not use a package statement, your class or interface ends up in the *default package*, which is a package that has no name. Generally speaking, the default package is only for small or temporary applications or when you are just beginning development. Otherwise, classes and interfaces belong in named packages.

Companies use their reversed Internet domain name in their package names, like, com.company.package. Some companies now choose to drop the first element com. Name collisions that occur within a single company need to be handled by convention within that company, perhaps by including the region or the project name after the company name, for example, com.infosys.bangalore.finacle.mobile

Compiling the Package

```
C:\JavaProgs>javac -d mypackage Calculator.java
```



- When the above command is executed, the compiler creates a folder called "mypackage" in our JavaProgs directory and stores the "Calculator.class" into this folder



Copyright © 2005, Infosys Technologies Ltd

84

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

When the above command is executed, the compiler creates a folder called "mypackage" in our JavaProgs directory and stores the "Calculator.class" into this folder

Importing a Package

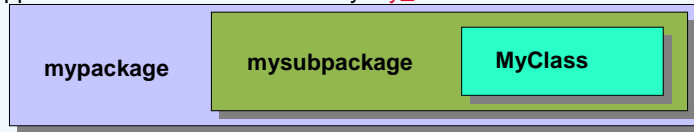
❏ How can a class which is not a part of a package reuse the classes in the package?

- Use this class as <package_name>.<class_name>
- Referring to a class always by its fully qualified name becomes cumbersome

❏ In Java, the Packages can be imported into a program in the following manner :

- `import <package_name>.<class_name>;`

❏ Suppose we wish to use a class say My_Class whose location is as follows :



- `import mypackage.mysubpackage.MyClass;`



Copyright © 2005, Infosys
Technologies Ltd

85

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

How can a class which is not a part of a package reuse the classes in the package?

Use this class as <package_name>.<class_name>

Referring to a class always by its fully qualified name becomes cumbersome

So we can import the class using the import statement and use the class name as such.

Importing a Package (Contd...)

- Two ways to import a package
- Importing a specific Package Member
 - `import mypackage.Calculator;`
- Importing an Entire Package
 - `import mypackage.*;`

BEST PRACTICE: Import only those classes that are required in your code, do not import the entire package

- What about naming conflicts if two imported packages both have a class with the same name?
 - Needs to be resolved using fully qualified name
- **Class Path setting**
 - Compiler and JVM must know from where to find the .class file
 - When working with packages, classpath setting has to be one level up the package hierarchy



Copyright © 2005, Infosys Technologies Ltd

86

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

There are two ways to import packages in java.

1. Importing only a single member of the package.
2. Importing all the members of a package.

1. Importing only a single member of the package.

To import only a single member of the package, we must put the import statement at the beginning of the java file just after the package statement if any. However, we can have comments preceding both the import statement, and the package statement. For example, you would import the Calculator class from the mypackage created in the previous section by including,

```
import myPackage.Calculator; statement.
```

Now you can refer to the Calculator class by its simple name:

2. Importing all the members of a package.

To import all the members of a package (ie, all the classes, interfaces etc), we use the import statement with the asterisk (*) wildcard character. For example import mypackage.*;

Now we can refer to any class or interface in the myPackage package by its short name:

Note: when we use *, only the classes and interfaces directly present in that package get imported. The sub-packages or the classes/interfaces present within the sub-package do not get imported.

The Java runtime automatically imports two entire packages by default:

The java.lang package and the current package by default (the classes in the current folder/directory)

Handling conflicts in class names !

There might be cases, where the name of the member of one package might be same as the name of the member belonging to another package, and both packages are imported. In such a case, we must refer to each member by its fully qualified name.

Unique package names are obtained universally by following the reverse of the domain name of the company



Eg. com.sun, com.microsoft

Classpath

It is a environmental variable, which contains the path for the default-working directory ie the present directory(.)

Importing + Creating a Package

- While **creating** a package, care should be taken that the statement for creating a package must be written before any other import statements

LEGAL 	ILLEGAL 
<pre>package mypackage ; import java.io;</pre>	<pre>import java.io; package mypackage ;</pre>

Education
and
Research



Copyright © 2005, Infosys
Technologies Ltd

87

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

While creating a package, which needs some other packages to be imported, the package statement should be the first statement of the program, followed by the import statement.

Source file layout -- revisited

```
package mypackage;
```

Package statement should be the first statement. It is optional

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

import statements are optional. There can be any no of import statements.

```
class MyClass{  
    //attributes and functions  
}  
public class Demo{  
    //attributes and functions  
}
```

Any no of classes and interfaces but there can be only one public class in a given .java file



Copyright © 2005, Infosys
Technologies Ltd

88

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is an example that demonstrates the creation of packages.

Access Specifiers

☑ Java provides access protection at **2 levels**:

- Class/Interface level
- Member Level

☑ Four Access Control Modifiers in Java

- **public**
- **private**
- **protected**
- **package** [default access specifier]

☑ Top level class/interface can have only two access – public or default access



Copyright © 2005, Infosys
Technologies Ltd

89

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

A member access level is determined by the **access specifier**, that modifies its declaration.

Java's access specifiers are **public**, **private** and **protected**. Java also defines a default access level.

Java provides access protection at **2 levels**:

- Class/Interface level
- Member Level

Top level class/interface can have only two access – public or default access

Access Specifiers (2 of 3)

- **private** – only accessible within the class
- **default** – accessible only within the package
- **protected** – similar to default with the addition that available to all child classes
ie even if child class is in a different package
- **public** – accessible to all
- In order of increasing access level:

private default protected public



Copyright © 2005, Infosys
Technologies Ltd

90

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Let us discuss the access levels of the access specifiers.

When a member is declared with

private – it will be accessible only within the class

default – i.e when nothing is specified it will be accessible only within the package

When protected – its behavior will be similar to default with the addition that available to all child classes ie even if child class is in a different package

When public – accessible to all

The increasing order of access level is..

Private default protected and then public.

Access Specifiers (3 of 3)

Keyword	Applicable To	Who can Access
<i>private</i>	Data members and methods	All members within the same Class only
<i>(No keyword, usually we call it default)</i>	Data members, methods, classes and interfaces	All classes in the same package
<i>protected</i>	Data members and methods	All classes in the same package as well as all sub classes ie even sub classes residing in a different package
<i>public</i>	Data members, methods, classes and interfaces	Any class



Copyright © 2005, Infosys Technologies Ltd

91

ER/CORP/CRS/LA10/003
Version 1.00

Infosys

If a class data has to be accessed from outside, the access rights has to be checked at two levels both the class level and the data member level

So if class A is in a package Package1 and class B which is in a different package package2 is trying to access a method of class A, then class A must first have public rights to be accessible from another package and also the function in class A must be declared public.

Standard Java Packages

java.lang

- Contains classes that form the basis of the design of the programming language of Java.
- You don't need to **explicitly import** this package. It is always imported for you.
- The String class, System class, Thread class, Wrapper classes etc, belong to this package.

java.io

- Input/Output operations in Java is handled by the java.io package.

java.util

- Contains classes and interfaces that provide additional utility.
- Example : creating **lists**, **calendar**, **date** etc.



Copyright © 2005, Infosys
Technologies Ltd

92

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Java Provides some of the standard packages. They are,

java.lang

Contains classes that form the basis of the design of the programming language of Java.

You don't need to **explicitly import** this package. It is always imported for you.

The String class, System class, Thread class, Wrapper classes etc, belong to this package.

java.io

Input/Output operations in Java is handled by the java.io package.

java.util

Contains classes and interfaces that provide additional utility.

Example : creating **lists**, **calendar**, **date** etc.

Standard Java Packages (Contd...)

java.net

- This package provides classes and interfaces for **TCP/IP network** programming.

java.awt

- This package is useful to **create GUI** applications.

java.applet

- This package consists of **classes** that you need, to **write programs to add more features to a web page.**



Copyright © 2005, Infosys
Technologies Ltd

93

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

java.net

This package provides classes and interfaces for **TCP/IP network** programming.

java.awt

This package is useful to **create GUI** applications.

java.applet

This package consists of **classes** that you need, to **write applets.**

Introduction to java.lang package

- Most widely used Package
- To convert numbers from strings : `parseByte()`, `parseShort()`, `parseInt()` and `parseLong()` are used.
- Integer, Byte, Short, Character, Boolean, Long, Float and Double are the wrapper classes for the primitive data types.
- System class holds a collection of static methods and variables. The standard input, output and error output of Java run time are in the `in`, `out`, `err` variables.
- Throwable class supports Java Exception Handling
- Security Manager is an abstract class that our subclasses can implement to create a security manager.



Copyright © 2005, Infosys Technologies Ltd

94

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Let us have a detailed look on java.lang package.

This is the Most widely used Package

To convert numbers from strings : `parseByte()`, `parseShort()`, `parseInt()` and `parseLong()` are used.

Integer, Byte, Short, Character, Boolean, Long, Float and Double are the wrapper classes for the primitive data types.

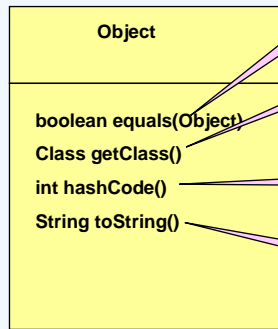
System class holds a collection of static methods and variables. The standard input, output and error output of Java run time are in the `in`, `out`, `err` variables.

Throwable class supports Java Exception Handling

Security Manager is an abstract class that our subclasses can implement to create a security manager.

Java.lang.Object class

- Part of java.lang package
- It is the superclass of all classes



Compares two object references

Returns the Class to which the object belongs

Represents a unique ID for the object

Represents a string message with name of the class that describe the object



Copyright © 2005, Infosys Technologies Ltd

95

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class. Every class is a descendant, direct or indirect, of the Object class. This class defines the basic state and behavior that all objects must have, such as the ability to compare oneself to another object, to convert to a string, to wait on a condition variable, to notify other objects that a condition variable has changed, and to return the class of the object.

java.lang.String class

- Present in **java.lang** package
- An object of the String class represents a fixed length, immutable sequence of characters
- Has overridden equals() method of the Object class that should be used to compare the actual string values
- Lot of other methods are available which are for the manipulation of characters of the string.
- You can refer to JavaDocs for the detailed list of methods



Copyright © 2005, Infosys
Technologies Ltd

96

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Java.lang.String

Present in **java.lang** package

An object of the String class represents a fixed length, immutable sequence of characters

Has overridden equals() method of the Object class that should be used to compare the actual string values

Lot of other methods are available which are for the manipulation of characters of the string.

You can refer to JavaDocs for the detailed list of methods

java.lang.StringBuffer class

- Present in **java.lang** package
- The prime difference between String and StringBuffer class is that the StringBuffer represents a string that can be **dynamically modified**.
- String Buffer's capacity could be dynamically increased even though its initial capacity is specified
- Whenever string manipulation like appending, inserting etc is required, this class should be used



Copyright © 2005, Infosys
Technologies Ltd

97

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

java.lang.StringBuffer class

Present in **java.lang** package

The prime difference between String and StringBuffer class is that the StringBuffer represents a string that can be **dynamically modified**.

String Buffer's capacity could be dynamically increased even though its initial capacity is specified

Whenever string manipulation like appending, inserting etc is required, this class should be used

You can refer to JavaDocs for the detailed list of methods

Wrapper Classes (1 of 2)

- There are many generic methods that take in Objects and not primitive data types as parameters.
- We need some mechanism to convert the primitive data type to Objects to use these generic methods
- The wrapper classes in **java.lang** package help us to do this
- To create a Wrapper class object

```
int primitiveInt = 500;  
Integer wrapperInt = new Integer(primitiveInt);  
int value = wrapperInt.intValue(); //gives back the primitive data  
type int
```

- Wrapper Class -> True Object Oriented implementation of the primitive data types



Copyright © 2005, Infosys
Technologies Ltd

98

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

There are many generic methods that take in Objects and not primitive data types as parameters.

We need some mechanism to convert the primitive data type to Objects to use these generic methods

The wrapper classes in **java.lang** package help us to do this

Wrapper Class -> True Object Oriented implementation of the primitive data types

Here is an example for Wrapper Classes.

Wrapper Classes (2 of 2)

Data Type	Wrapper Class	Data Type	Wrapper Class
boolean	Boolean	byte	Byte
char	Character	short	Short
long	Long	int	Integer
float	Float	double	Double



Copyright © 2005, Infosys
Technologies Ltd

99

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The table shows the primitive types & the wrapper classes present in the **java.lang** package.

Wrapper class is constructed by passing the value to be wrapped into the appropriate constructor.

java.util package

Java.util contains a wide assortment of classes and interfaces that support a broad range of functionality.

Important classes of java.util package are :

- GregorianCalendar
- ArrayList
- Collections
- HashMap
- TreeMap
- TreeSet
- Vector
- Random



Copyright © 2005, Infosys
Technologies Ltd

100

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

java.util package

Java.util contains a wide assortment of classes and interfaces that support a broad range of functionality.

Important classes of java.util package are :

GregorianCalendar
ArrayList
Collections
HashMap
TreeMap
TreeSet
Vector
Random

Collections Framework

• When do we use the Collection classes?

- When flexibility is required in terms of growing and shrinking in size
- When sequence matters
- When no duplicates are to be allowed
- When a value is to be referred to using a key



Copyright © 2005, Infosys
Technologies Ltd

101

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The Java 2 platform includes a new *collections framework*. A *collection* is an object that represents a group of objects. A collections framework is a unified architecture for representing and manipulating collections, allowing them to be manipulated independently of the details of their representation.

When do we use the Collection classes?

- When flexibility is required in terms of growing and shrinking in size
- When sequence matters
- When no duplicates are to be allowed
- When a value is to be referred to using a key

Collections Framework

- All collection classes in the Java API implement any one of the three interfaces
 - List, Set, Map
- **List** – A collection of objects *where the element present at a particular index* is known
- **Set** – A collection that *doesn't allow duplicates*
- **Map** – A collection that provides a *key-value* capability



Copyright © 2005, Infosys
Technologies Ltd

102

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

There are six *collection interfaces*. The most basic interface is Collection. Three interfaces that extend Collection are Set, List, and SortedSet. The other two collection interfaces, Map and SortedMap, do not extend Collection, as they represent mappings rather than true collections. However, these interfaces contain *collection-view* operations, which allow them to be manipulated as collections.

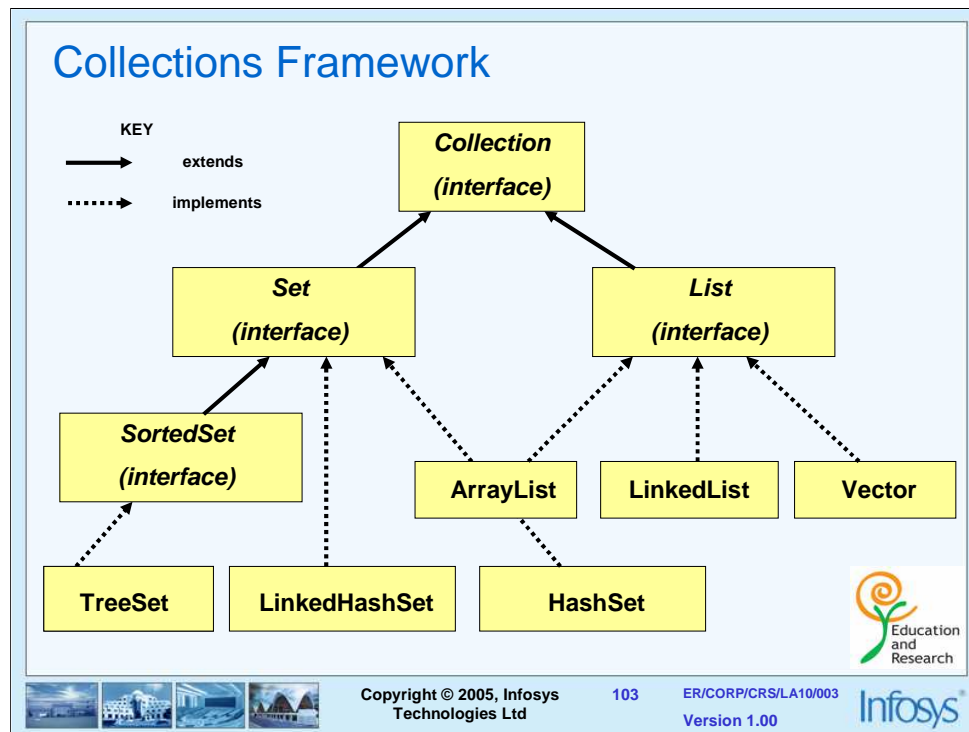
All collection classes in the Java API implement any one of the three interfaces

List, Set, Map

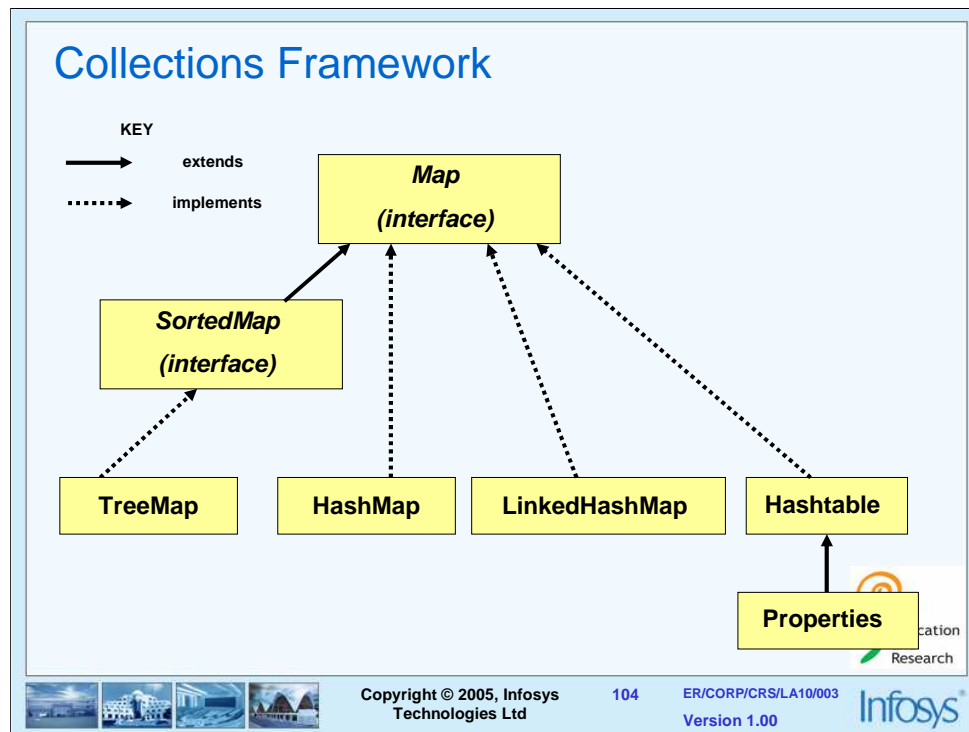
List for A collection of objects *where the element present at a particular index* is known

Set for A collection that *doesn't allow duplicates*

Map for A collection that provides a *key-value* capability



This shows the hierarchy from Collection Interface.



As discussed Map and SortedMap, do not extend Collection, as they represent mappings rather than true collections. This shows the hierarchy from Map Interface.

Collections Framework

• The **Collection** interface provides important methods which are implemented by the implementing classes in their own way

- add()
- remove()
- isEmpty()
- size()
- contains()



Copyright © 2005, Infosys
Technologies Ltd

105

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The collection interface is the foundation upon which the collections framework is built.

The **Collection** interface provides important methods such as

add()
remove()
isEmpty()
size()
contains()

which are implemented by the implementing classes in their own way

Collections Framework

• The **Map** interface similarly provides important methods which are implemented by the implementing classes in their own way

- clear()
- get()
- containsKey()
- isEmpty()
- size()
- put()
- remove()



Copyright © 2005, Infosys
Technologies Ltd

106

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The **Map** interface similarly provides methods such as

clear()
get()
containsKey()
isEmpty()
size()
put()
remove()

which are implemented by the implementing classes in their own way

Collections Framework

- Enumeration and Iterator interface provides capability to iterate through a collection in a standard manner independent of implementation
 - ArrayList or LinkedList could both be traversed in a similar way
- New implementations should preferably use Iterator
- Enumeration has methods like
 - **hasMoreElements ()** to check for the existence of elements in the collection
 - **nextElement ()** to retrieve the next element
- Iterator has similar methods like the Enumeration and an additional method to optionally remove the last element returned by the Iterator
 - **hasNext()**
 - **next()**
 - **remove()**



Copyright © 2005, Infosys
Technologies Ltd

107

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Enumeration and Iterator interface provides capability to iterate through a collection in a standard manner independent of implementation.

Enumeration has methods like

hasMoreElements () to check for the existence of elements in the collection
and

nextElement () to retrieve the next element

Iterator has similar methods like the Enumeration and an additional method **remove()** to optionally remove the last element returned by the Iterator

Collections Framework

- StringTokenizer class is an implementation of Enumeration interface
- It helps to split a string into tokens based on a delimiter character
 - **hasMoreTokens()** returns a boolean based on whether more tokens are to be returned
 - **nextToken()** returns the next token



Copyright © 2005, Infosys
Technologies Ltd

108

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

StringTokenizer class is an implementation of Enumeration interface

It helps to split a string into tokens based on a delimiter character

It has methods,

hasMoreTokens() which returns a boolean, based on whether more tokens are to be returned

nextToken() which returns the next token

Vectors

```
import java.util.*;
class VectorDemo{
    public static Vector v;
    public static void printVector(Vector v){
        Enumeration vEnum=v.elements();
        while(vEnum.hasMoreElements()){
            System.out.print(vEnum.nextElement()+" ");
        }
    }
    public static void main(String[] args){
        v=new Vector(3);
        v.add ("a");
        v.add ("b");
        v.add ("c");
        printVector(v);
    }
}
```



Copyright © 2005, Infosys
Technologies Ltd

109

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is a simple example that shows how to work with the Collection Class Vector which is a part of collections framework.



Basic Java Programming

Module – 5 : Exception Handling



We shall now move on to the module on Exception Handling

Learning Outcomes

- After completion of the module you will be able to
 - Understand the concepts of Exception handling in Java
 - Understand the constructs available for Exception Handling
 - Know how to perform Exception Handling
 - Know how to create user defined exceptions.



Copyright © 2005, Infosys
Technologies Ltd

111

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

After completion of the module you will be able to

- Understand the concepts of Exception handling in Java
- Understand the constructs available for Exception Handling
- Know how to perform Exception Handling
- Know how to create user defined exceptions.

Exception Handling in Java

Consider the following Code

```
class ExceptionDemo
{
    public static void main(String argv[])
    {
        int n=0;
        int e=16/n;
        System.out.println(e);
    }
}
```

- ❶ What happens when the above code is executed ?
- ❷ Exception is thrown (an object is thrown)
- ❸ How to recover from this ? (handle it !)



Copyright © 2005, Infosys Technologies Ltd

112

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

An **Exception** is a run-time error. **Exception Handler** is a set of instructions that handles an exception.

Exceptions can occur when

- you try to open a non-existing file.
- There is a network connection problem.
- Operands being manipulated are out of prescribed ranges
- Class file missing which was supposed to be loaded.

Analyse the given Code

What happens when the above code is executed ? [Pause the Presentation and Try Yourself]

An Exception is thrown

How to recover from this ? (by handling it !)

The default handler, provided by the Java run time system handles the exceptions which are not handled by the programmer. Such exceptions are referred to as **Uncaught Exceptions**. The default handler handles the Uncaught Exceptions by displaying a string describing the exception, prints the stack trace from the point at which the exception occurred and terminates the program.

The Exception in the above code is example for Uncaught Exception.

Exception Handling in Java (Contd...)

- An **Exception** is a run-time error
- It is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
- **Exception Handler** is a set of instructions that handles an exception
- The Java programming language provides a mechanism to help programs report and handle errors



Copyright © 2005, Infosys
Technologies Ltd

113

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The Java programming language provides a mechanism known as exceptions to help programs report and handle errors. When an error occurs, the program throws an exception. What does this mean? It means that the normal flow of the program is interrupted and that the runtime environment attempts to find an *exception handler* which is a block of code that can handle a particular type of error. The exception handler can attempt to recover from the error or, if it determines that the error is unrecoverable, provide a gentle exit from the program.

Exception Handling in Java (Contd...)

- When an error occurs, the program throws an exception
- The **exception object** that is thrown contains information about the exception, including its type and the state of the program when the error occurred.

E.g:

An exception generated by the system is given below:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at ExceptionDemo.main(ExceptionDemo.java:5)

ExceptionDemo	:	The class name
main	:	The method name
ExceptionDemo.java	:	The filename
java:5	:	Line number
ArithmeticException	:	It is a subclass of Exception, which describes the type of the error happened.

- The runtime environment attempts to find the Exception Handler
- The exception handler can attempt to recover from the error or, if it determines that the error is unrecoverable, provide a gentle exit from the program.
- Helpful in separating the execution code from the error handler



Copyright © 2005, Infosys Technologies Ltd

114

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

A Java exception is an object that describes an exceptional (that is, error) condition, that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and **thrown** in the method that caused the error. That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is **caught** and processed.

Exceptions can be generated by the Java run-time system, or they can be manually generated by your code.

Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment. A new exception object is thrown by the Java run-time system when it detects an error for example the attempt to divide by zero. This exception is constructed by the Java run-time system itself. Thus if an exception is thrown, the Java run-time system requires it to be caught in order to run the program.

The exception generated by the system for the action Division by Zero in the sample program discussed is:

Exception in thread "main" java.lang.ArithmeticException: / by zero
at ExceptionDemo.main(ExceptionDemo.java:5)

Where,

ExceptionDemo is	The class name	
main	is	The method name
ExceptionDemo.java	is	The filename
java:5	is	the Line number

Exception Handling in Java (Contd...)

- Java exception handling is managed via **try**, **catch**, **throw**, **throws**, and **finally**.
- Program statements that you want to monitor for exceptions are contained within a **try** block. If an exception occurs within the try block, it is thrown. Our code can **catch** these exceptions that are automatically thrown by the Java run-time system.
- To manually **throw** an exception we use the keyword **throw**. Any exception that is thrown out of a method must be specified by a **throws** clause. Any code that absolutely must be executed before a method returns is put in a **finally** block.



Copyright © 2005, Infosys
Technologies Ltd

115

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Java exception handling is managed via **try**, **catch**, **throw**, **throws**, and **finally**.

Program statements that you want to monitor for exceptions are contained within a **try** block. If an exception occurs within the try block, it is thrown. Our code can **catch** these exceptions that are automatically thrown by the Java run-time system.

To manually **throw** an exception we use the keyword **throw**. Any exception that is thrown out of a method must be specified by a **throws** clause. Any code that absolutely must be executed before a method returns is put in a **finally** block.

Exception Handling in Java (Contd...)

- The *try* statement identifies a block of statements within which an exception might be thrown.
- The *catch* statement must be associated with a try statement and identifies a block of statements that can handle a particular type of exception. The statements are executed if an exception of a particular type occurs within the try block.
- The *finally* statement must be associated with a try statement and identifies a block of statements that are executed regardless of whether or not an error occurs within the try block.



Copyright © 2005, Infosys
Technologies Ltd

116

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The *try* statement identifies a block of statements within which an exception might be thrown.

The *catch* statement must be associated with a try statement and identifies a block of statements that can handle a particular type of exception. The statements are executed if an exception of a particular type occurs within the try block.

The *finally* statement must be associated with a try statement and identifies a block of statements that are executed regardless of whether or not an error occurs within the try block.

Exception Handling in Java (Contd...)

Here's the general form of these statements:

```
try {  
    statement(s)  
} catch (exceptiontype name) {  
    statement(s)  
} finally {  
    statement(s)  
}
```



Copyright © 2005, Infosys
Technologies Ltd

117

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here's the general form of these statements

Exception Handling in Java (Contd...)

<u>Program 1</u>	<u>Program 2</u>
<pre>class Hello { public static void main (String args[]) { /* Now let's say hello */ System.out.print("Hello"); System.out.println(args[0]); } }</pre>	<pre>class ExceptionalHello { public static void main (String args[]) { /* Now let's say hello */ try { System.out.println("Hello" + args[0]); } catch (Exception e) { System.out.println("Hello whoever you are"); } } }</pre>



Copyright © 2005, Infosys
Technologies Ltd

118

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Consider Program 1

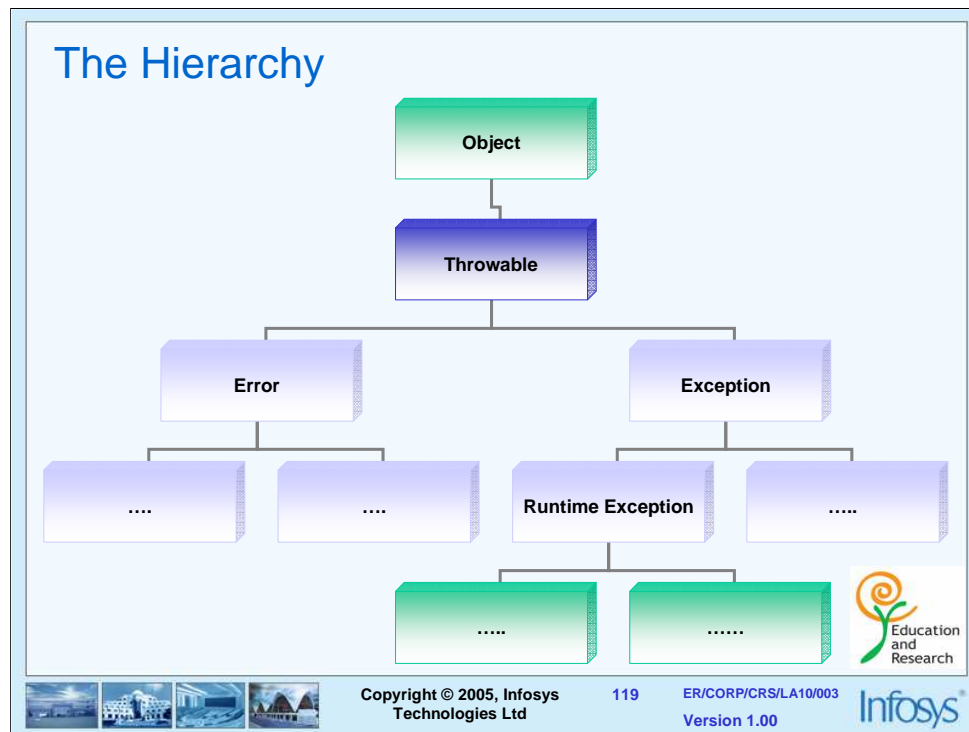
what will happen when you run the program without giving it any command line arguments? The runtime system generates an exception, Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException at Hello.main (at Line Number 7)

What happened was that since we didn't give Hello any command line arguments there wasn't anything in args[0]. Therefore Java kicked back this not too friendly error message about an "ArrayIndexOutOfBoundsException".

Usually we fix this problem by testing the length of the array before we tried to access its first element. This worked well in this simple case, but this is far from the only such potential problem. If you were to check for every possible error condition in each line of code, you would find your code becoming bloated with more error checking than actual code. Moreover you then have to start checking for errors in the error conditions.

The goal of exception handling is to be able to define the regular flow of the program in part of the code without worrying about all the special cases. Then, in a separate block of code, you cover the exceptional cases. This produces more legible code since you don't need to interrupt the flow of the algorithm to check and respond to every possible strange condition. The runtime environment is responsible for moving from the regular program flow to the exception handlers when an exceptional condition arises.

In practice what you do is write blocks of code that may generate exceptions inside try blocks as shown in Program 2. You try the statements that generate the exceptions. Within your try block you are free to act as if nothing has or can go wrong. Then, within one or more catch blocks, you write the program logic that deals with all the special cases.



This diagram describes the Exception hierarchy.

At the very top level, we have the object class, which is the base class of all the classes available in java. There is a class called **Throwable**, which inherits from the Object class.

We have two branches from the Throwable class. One called the **Error**, and the other called **Exception**.

The **Error** class defines exceptions or problems that are not expected to be caught under normal circumstances by our program. Eg, memory error, hardware error, JVM error etc.

On the other end, we have the **Exception** class, which represent exceptions which can be handled by our program, and our program can recover from these exceptions using the try-catch-finally block. **RuntimeException** is on subclass of the Exception class. Exceptions of this type represent exceptions that occur at run time, and which cannot be trapped at compile time. A very good example for the same is, **DivideByZero** exception or **NullPointerException**.

Checked and Unchecked Exceptions

- Checked Exceptions : The exceptions defined by java.lang that must be included in a method's throws list if that method can generate one of these exceptions and does not handle it itself.
- Unchecked Exceptions : These exceptions need not be included in any method's throws list because compiler does not check to see if a method handles or throws these exceptions



Copyright © 2005, Infosys
Technologies Ltd

120

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Checked Exceptions : The exceptions defined by java.lang that must be included in a method's throws list if that method can generate one of these exceptions and does not handle it itself.

Examples of Checked Exceptions :

- 1 ClassNotFoundException
- 2 IllegalAccessException
- 3 NoSuchFieldException
- 4 InterruptedException etc.

Unchecked Exceptions : These exceptions need not be included in any method's throws list because compiler does not check to see if a method handles or throws these exceptions

Examples of UnChecked Eceptions

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ClassCastException
- NegativeArraySizeException etc.

Exceptions and Errors

- Exceptions are situations within the control of an application, that it should try to handle
- Errors indicate serious problems and abnormal conditions that most applications should not try to handle



Copyright © 2005, Infosys
Technologies Ltd

121

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Errors indicate serious problems and abnormal conditions that most applications should not try to handle

When a dynamic linking failure or some other "hard" failure in the virtual machine occurs, the virtual machine throws an Error. Typical Java programs should not catch Errors. In addition, it's unlikely that typical Java programs will ever throw Errors either.

Some Java Errors are,

- ClassFormatError
- InternalError
- LinkageError
- OutOfMemoryError
- StackOverflowError
- UnknownError

Exceptions are situations within the control of an application, that it should try to handle

And Some Java Exceptions are

- ArithmeticException
- ClassCastException
- IllegalStateException
- IndexOutOfBoundsException

Try – Catch - Finally

- To guard against and handle a run-time error, simply enclose the code that you suspect of an exception being thrown, inside a try block. Immediately following the try block, include a catch clause that specifies the exception type that is to be caught.
- A try block is always followed by a catch block



Copyright © 2005, Infosys
Technologies Ltd

122

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

try and catch

Once an exception is thrown, program control is passed out of the try block into the catch block. Once the catch statement has executed, program control continues with the next line of the program following the try/catch signature.

A try and catch statement work in tandem or together. A try block is always followed by a catch block, which handles or catches the exception. A catch block always monitors the preceding try block. The catch block is not executed if no exception is thrown

Try – Catch – Finally- (Contd..)

Multiple Exceptions

- If there are multiple exception classes that might arise in the **try** block, then several **catch** blocks are allowed to handle them separately, each handling different exception type. After one catch statement executes, the others are bypassed, and execution continues after the **try/catch** block.



Copyright © 2005, Infosys
Technologies Ltd

123

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

If there are multiple exception classes that might arise in the **try** block, then several **catch** blocks are allowed to handle them separately, each handling different exception type. After one catch statement executes, the others are bypassed, and execution continues after the **try/catch** block.

Try – Catch – Finally- (Contd..)

```
public class MultipleCatchExample{
    public static void main(String argx[]) {
        try{
            int e=argx.length;
            int f=20/e;
            int g[]={1};
            g[20]=81;
        }
        catch(ArithmeticException ae) {
            System.out.println("Divided by zero"+ae);
        }
        catch(ArrayIndexOutOfBoundsException oe) {
            System.out.println("Array index out of bound"+oe);
        }
        System.out.println("After Try and Catch blocks");
    }
}
```



Copyright © 2005, Infosys
Technologies Ltd

124

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Pause the Presentation and Try the given code by Yourself.

So the output is:

Divided by zero java.lang.ArithmeticException: / by zero

After Try and Catch blocks

i.e

```
int g[]={1};
```

```
g[20]=81;
```

Statements in the Try block are skipped and the catch statement for
ArrayIndexOutOfBoundsException is bypassed.

Try – Catch – Finally- (Contd..)

Nesting the Try - Catch

- The try statements can be nested. Each time a **try** statement does not have a **catch** handler for a particular exception, the stack is unwound and the next **try** statement's **catch** handlers are inspected for a match. If no **catch** statement matches, then the java run-time system will handle the exception.



Copyright © 2005, Infosys
Technologies Ltd

125

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The try statements can be nested. Each time a **try** statement does not have a **catch** handler for a particular exception, the stack is unwound and the next **try** statement's **catch (ie parent try statement's catch)** handlers are inspected for a match. If no **catch** statement matches, then the java run-time system will handle the exception.

Try – Catch – Finally- (Contd..)

```
class NestedTryExample{
    public static void main(String argx[]){
        try{
            int a=argx.length;
            int b=20/a;
            System.out.println("a="+a);
            try {
                if(a==1){
                    a=a/(a-a);
                }
                if(a==2){
                    int c[]={1};
                    c[22]=99;
                }
            }
            catch(ArrayIndexOutOfBoundsException oe){
                System.out.println("Array index out of bounds");
            }
            catch(ArithmeticException ae){
                System.out.println("Divide by 0");
            }
        }
    }
}
```



Copyright © 2005, Infosys
Technologies Ltd

126

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Pause the presentation and try the given code and Analyse the result.

Finally Block

- The finally statement is associated with a try statement and identifies a block of statements that are executed regardless of whether or not an exception occurs within the try block.
- Defines the code that is executed always
- In the normal execution it is executed after the try block
- When an exception occurs, it is executed after the handler if any or before propagation as the case may be



Copyright © 2005, Infosys
Technologies Ltd

127

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The finally statement is associated with a try statement and identifies a block of statements that are executed regardless of whether or not an exception occurs within the try block.

Defines the code that is executed always

In the normal execution it is executed after the try block

When an exception occurs, it is executed after the handler if any or before propagation as the case may be

Finally Block-(Contd...)

- The circumstances that **prevent execution** of the code in a finally block are:
 - The death of a Thread (Threads will be discussed later)
 - Using of the System.exit() method.
 - Due to an exception arising in the finally block.
- An exception in the finally block, exactly behaves like any other exception.



Copyright © 2005, Infosys
Technologies Ltd

128

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The circumstances that **prevent execution** of the code in a finally block are:

- The death of a Thread
- Using of the System.exit() method.
- Due to an exception arising in the finally block.

An exception in the finally block, exactly behaves like any other exception.

Finally Block-(Contd...)

```
public class ExceptionDemoProgram{
    public static void main(String[] args){
        try{
            int i=Integer.parseInt(args[0]);
            int c=i/i;
        }
        catch(ArithmeticException e){
            System.out.println("ArithmeticException:"+e);
        }
        finally{
            System.out.println("inside finally");
        }
        System.out.println("End Statement");
    }
}
```



Copyright © 2005, Infosys
Technologies Ltd

129

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Pause the presentation and Try and analyze the given code with and with out command line argument values.

Throwing Exceptions

- Use the **throw** clause to throw an exception
- Exceptions in Java are compulsorily of type Throwable
- If you attempt to throw an object that is not throwable, the compiler refuses to compile your program
- Can also be used to re-throw an exception

```
public void read() throws IOException{  
    // Some code that causes IO Exception  
    throw new IOException();  
}
```

- Any exception that is thrown out of a method must be specified by a **throws** clause



Copyright © 2005, Infosys
Technologies Ltd

130

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Throw statement is used explicitly to throw an exception.

Its syntax is **throw ThrowableInstance;**

There is an important reason why the **throw** statement and the construction of the exception are normally combined. The exception builds information about the point at which it was created, and that information is shown in the stack trace when the exception is reported. It is convenient if the line reported as the origin of the exception is the same line as the throw statement, so it is a good idea to combine the two parts, and `throw new exception()` becomes the norm.

Throws

- The **throws** keyword is used along with the declaration of a method that can throw an exception.
- When defining a method you must include a throws clause to declare those exceptions that might be thrown but is not caught in the method.
- This tells other methods
“ If you call me, you must handle these exceptions that I throw”.



Copyright © 2005, Infosys
Technologies Ltd

131

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Throws

The **throws** clause in the method header tells the compiler that we know this exception may occur and if it does, the exception should be thrown to the caller of this method instead of crashing the program. The **throws** clause is placed after the parameter list and before the opening brace of the method. If more than one type of checked exception needs to be declared, separate the class names with commas.

A throws clause lists the types of exceptions that a method might throw. It is a general rule that any method that might throw an exception must declare the fact.

User defined exceptions

- We can create our own exception objects, which can be thrown using the throw keyword
- Create a class which extends the Exception class [this is our user-defined exception class, that we want to throw]
- Create an instance of the user-defined exception class.
- Use the **throw** keyword to throw the instance created.



Copyright © 2005, Infosys
Technologies Ltd

132

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

We can create our own exception objects, which can be thrown using the throw keyword. Such exceptions are User defined Exceptions.

To have user defined exceptions, create a class which extends the Exception class.
Create an instance of the user-defined exception class.
Use the **throw** keyword to throw the instance created.

User defined exceptions (Contd...)

```
class MyEx extends Exception{
    MyEx(String msg){
        super(msg);
    }
}
public class UserdefinedException{
    public static void main(String[] args){
        int i=args.length;
        try{
            if(i==0){
                throw new MyEx("Arguments Not Available");
            }
        }
        catch(MyEx e){
            System.out.println(e);
        }
    }
}
```



Copyright © 2005, Infosys
Technologies Ltd

133

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Try the code and analyse the results.



Basic Java Programming

Module – 6 : Applets



We shall now move on to the module on Java Applets.

Learning Outcomes

After completion of the module you will be able to Understand

- What are Applets
- The Life Cycle of Applets
- How to run Applets
- How to run Applets by passing parameters to it.



Copyright © 2005, Infosys
Technologies Ltd

135

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

After completion of the module you will be able to Understand

What are Applets

The Life Cycle of Applets

How to run Applets

How to run Applets by passing parameters to it.

Applets

- A Java class that can be **embedded within a HTML page** and downloaded and executed by a web browser
- An applet can be used to enhance the features of a Web Page
- The applet runs on **JVM embedded in the browser**



Copyright © 2005, Infosys
Technologies Ltd

136

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

An Applet is a small application that can be **embedded within a HTML page** and downloaded and executed by a web browser.

An applet can be used to enhance the features of a Web Page

The applet runs on **JVM embedded in the browser**

Applets

- Applet is a **container class** that is available in **java.applet** package
- We can create our own Applets by **extending the Applet class**



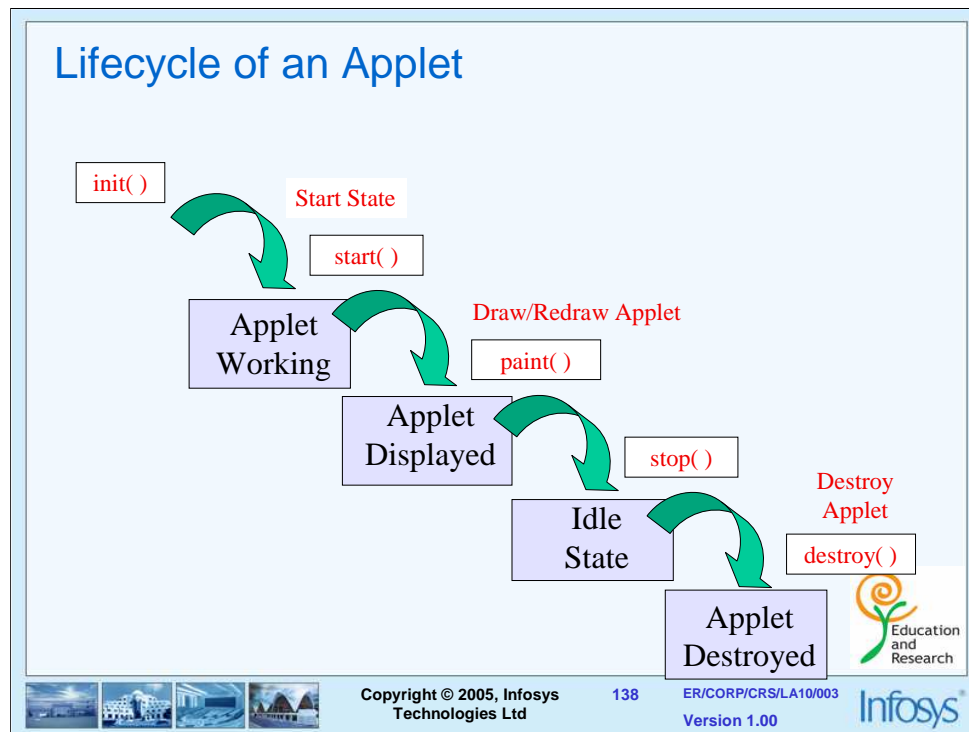
Copyright © 2005, Infosys
Technologies Ltd

137

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

While writing Applets we have to import applet package which contains Applet class. Every applet that you create must be a subclass of Applet class.



An applet is an window-based program.

The methods `init()`, `start()`, `stop()` and `destroy()` defined in Applet class and `paint()` method defined in AWT Component class forms the life cycle methods of an applet.

These methods are called in sequence by AWT.

When the applet begins first `init()` will be called then `start()` and then `paint()` will be called.

When the applet is terminated first `stop()` will be called then `destroy()` will be called.

Lifecycle of an Applet (Contd...)

- The browser calls the **init** method of the Applet, followed by the **start** method
- If the users leaves the web page, the browser will call the **stop** method of the Applet



Copyright © 2005, Infosys
Technologies Ltd

139

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The browser calls the **init** method of the Applet, followed by the **start** method
If the users leaves the web page, the browser will call the **stop** method of the Applet

Lifecycle of an Applet (Contd...)

- If the user comes back to the page the browser will call the **start** method of the Applet again
- The **destroy** method is called just before the applet is finally unloaded from memory
- **Paint** method is called whenever the applet's output must be redrawn.



Copyright © 2005, Infosys
Technologies Ltd

140

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

If the user comes back to the page the browser will call the **start** method of the Applet again

The **destroy** method is called just before the applet is finally unloaded from memory

Paint method is called whenever the applet's output must be redrawn.

Execution of an applet

- 1. Compile the applet program, say MyApplet.java, to get the .class file, MyApplet.class
- 2. Embed the <applet></applet> tags in a html file as follows

```
<applet code="MyApplet" width=200 height=200>
</applet>
```

 - **Code, width and height** are mandatory attributes for the applet tag
- 3. Use any **java compatible browser** to run the html file



Copyright © 2005, Infosys
Technologies Ltd

141

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

In order to execute the applet, Compile the applet program, say MyApplet.java, to get the .class file, MyApplet.class

Embed the applet tags in a html file as shown here.

Code, width and height are mandatory attributes for the applet tag

Use any **java compatible browser** to run the html file

Execution of an applet (Contd...)

- For testing an Applet, we can use the **appletviewer tool** which is in <javahome>\bin directory
- Type the applet tag alone in a file, say applet.txt, and type the following command
 - appletviewer applet.txt
- Instead of creating a separate file, the applet tag can be included as a comment in MyApplet.java file itself. The command will now be as follows
 - appletviewer MyApplet.java
- The appletviewer tool will open a window to display the Applet



Copyright © 2005, Infosys Technologies Ltd

142

ER/CORP/CRS/LA10/003
Version 1.00

Infosys

For testing an Applet, we can use the **appletviewer tool** which is in <javahome>\bin directory

Type the applet tag alone in a file, say applet.txt, and type the command “appletviewer applet.txt” in command prompt

Instead of creating a separate file, the applet tag can be included as a comment in MyApplet.java file itself. Then type the command “appletviewer MyApplet.java” in command prompt

The appletviewer tool will open a window to display the Applet

Other optional attributes of <applet> tag are:

1. CODEBASE which contains the URL directory, required if the .class file is in some other location from which the html file will be run
2. HSPACE which specifies horizontal spacing in pixels
3. VSPACE which specifies vertical spacing in pixels
4. ALT which specifies alternate text to be displayed in case applet is not loaded
5. PARAM to pass parameters in to an applet

Parameter passing to an applet

- Parameters are passed into an applet as **name-value pair** using the **param** tag within applet tag

```
- <applet code="ParameterApplet" width=200 height=200>  
  <param name="name" value="value">  
</applet>
```

- The values are passed as **String** values
- getParameter (String parameterName)** returns the value



Copyright © 2005, Infosys
Technologies Ltd

143

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Parameters are passed into an applet as **name-value pair** using the **param** tag within applet tag as shown here.

The values are passed as **String** values

getParameter (String parameterName) in the applet code returns the value



Basic Java Programming

Module – 7 : javax.swing package



We shall now move on to the module on Swings.

Learning Outcomes

- After completion of the module you will be able to
 - Understand the need for Swing Programming
 - Design GUIs using Swing
 - Work with various components available in Swing
 - Work on Event Handling
- The code snippets given here are tested with J2SE5.



Copyright © 2005, Infosys
Technologies Ltd

145

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

After completion of the module you will be able to

- Understand the need for Swing Programming
- Design GUIs using Swing
- Work with various components available in Swing
- Work on Event Handling

SWING : Basic Concepts

- Swing is a set of classes that provides more powerful and flexible GUI components
- Swing is a response to the deficiencies in AWT.
- AWT translates its various visual components into their corresponding platform specific equivalents
- AWT components use native code resources, so termed as heavyweight.



Copyright © 2005, Infosys
Technologies Ltd

146

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Swing is a set of classes that provides more powerful and flexible GUI components

Swing is a response to the problems faced by AWT.

The problem with AWT is that, AWT components are heavyweight components. The reason is, AWT translates its various visual components into their corresponding platform specific equivalents. i.e AWT components use native code resources.

Java Swings provides solution to this problem. How the solution is provided? The answer is the Components of swings are written entirely in Java and the look and feel of the components are determined entirely by Swings and not by underlying platform

Features of Swing

☉ Lightweight Components

- Swing components are written entirely in Java and do not map directly to platform-specific peers.
- Look and Feel of each component is determined by Swing and not by underlying OS

☉ Supports Pluggable Look and Feel

- It is possible to separate the look and feel of a component from the logic.
- Possible to define a look and feel that is consistent across all platforms



Copyright © 2005, Infosys
Technologies Ltd

147

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

What are the features of Swings?

The Swing components are light weight components which means, Swing components are written entirely in Java and do not map directly to platform-specific peers. Look and Feel of each component is determined by Swing and not by underlying OS. Look and Feel of Swings are said to be pluggable. The reason is, It is possible to separate the look and feel of a component from the logic.

Features of Swing (Contd...)

- Swing is an extension for the AWT, not a replacement
- Has large set of built-in Components.
- There is an almost-equivalent Swing component for most AWT components .
- The Swing components are JavaBean compliant.



Copyright © 2005, Infosys
Technologies Ltd

148

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Swing is an extension for the AWT, not a replacement

Has large set of built-in Components.

There is an almost-equivalent Swing component for most AWT components .

The Swing components are JavaBean compliant

MVC and Swing

Visual Component is composite of three distinct aspects :

- The way that component looks when rendered on screen
- The way that component reacts to user
- The state information associated with the component

The architecture is **MVC – MODEL VIEW CONTROLLER**

- **Model** – state information associated with component
- **View** – how the component is displayed on the screen
- **Controller** – how the component reacts to the user

SWING uses a modified version of MVC that combines the view and controller into a single logical entity called UI delegate.

Therefore SWINGS approach is called either **Model-Delegate** architecture or **Separable Model** Architecture.



Copyright © 2005, Infosys Technologies Ltd

149

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Visual Component is composite of three distinct aspects :

- The way that component looks when rendered on screen
- The way that component reacts to user
- The state information associated with the component

The architecture behind it is **MVC – MODEL VIEW CONTROLLER** Architecture.

Where,

- Model** represents the state information associated with component
- View** represents how the component is displayed on the screen
- Controller** represents how the component reacts to the user

Let us consider the Check box component as example. Where,

The Model contains a field that indicates whether the box is checked or unchecked

The View represents display on the screen including any aspects of the view that are affected by the current state of the model

And when the user clicks a check box, **controller** reacts by changing the model to reflect's the user choice, which results in view being modified.

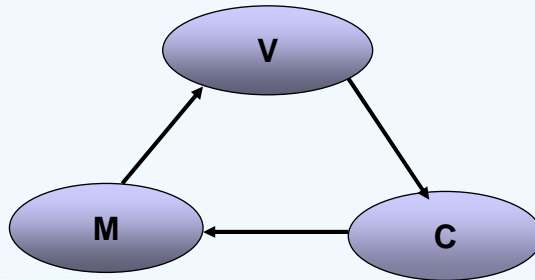
By separating a component into a model, a view and a controller, the specific implementation of each can be changed without affecting the other two.

SWING uses a modified version of MVC that combines the view and controller into a single logical entity called UI delegate.

Therefore SWINGS approach is called either **Model-Delegate** architecture or **Separable Model** Architecture.

MVC Architecture

- Swing architecture implements MVC
- MVC
 - **Model** - represents the data in the application
 - **View** - represents the presentation of data
 - **Controller** - communicates between the model and view



Copyright © 2005, Infosys
Technologies Ltd

150

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Swing Architecture implements MVC architecture where,

- Model** - represents the data in the application
- View** - represents the presentation of data and
- Controller** - communicates between the model and view

Working with Swings

```
import javax.swing.*;

public class JFrameExample1{

    public static void main(String[] args){

        JFrame.setDefaultLookAndFeelDecorated(true);

        JFrame frame = new JFrame("FrameDemo");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel jl1=new JLabel("Welcome to Swing Programming");

        frame.getContentPane().add(jl1);

        frame.pack();

        frame.setVisible(true);

    }

}
```



Copyright © 2005, Infosys
Technologies Ltd

151

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Let us start our discussion on Swings with the simple example. The given program will display a Frame titled “Frame Demo” with a Label displaying “Welcome to Swing Programming”.

Let us analyze the code in detail.

The statement `import javax.swing package`, imports `javax.swing package` which contains a set of "lightweight" components.

In Swing programming in order to design user interfaces with Frames, `JFrame Component` is used, which is an extended version of `java.awt.Frame` that adds support for the JFC/Swing component architecture.

`JFrame.setDefaultLookAndFeelDecorated(true);` Makes sure we have nice window decorations.

`JFrame frame = new JFrame("FrameDemo");` creates an invisible frame with the title “Frame Demo”.

`frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
This makes the program exit when the Close button is clicked on the frame.

`JLabel jl1=new JLabel("Welcome to Swing Programming");`

Executing a Simple Swing Application(Contd.)

- Swing programs are compiled and run in the same way as other Java applications
- Store the previous program under the name **JFrameExample1.java**
- To compile the program execute
 - `javac JFrameExample1.java`
- To run the program execute
 - `java JFrameExample1`



Copyright © 2005, Infosys
Technologies Ltd

152

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Swing programs are compiled and run in the same way as other Java applications

Store the previous program under the name **JFrameExample1.java**

To compile the program execute

```
javac JFrameExample1.java
```

To run the program execute

```
java JFrameExample1
```

This will display the output as shown here.

Best Practice.. For thread-safety problem.

```
public class JFrameExample{  
    public static void main(String[] args){  
        try{  
            SwingUtilities.invokeLater(new  
                Runnable(){  
                public void run(){  
                    JFrameExample jfex=new  
                        JFrameExample();  
                    jfex.makeGUI();  
                }  
            });  
        } catch(Exception e){  
        }  
    }  
}  
  
    public void makeGUI(){  
        JFrame.setDefaultLookAndFeelDecorated(true);  
        JFrame frame = new  
            JFrame("FrameDemo");  
        frame.setDefaultCloseOperation(  
            JFrame.EXIT_ON_CLOSE);  
        JLabel jl1=new JLabel("Welcome to Swing  
            Programming");  
        frame.getContentPane().add(jl1);  
        frame.pack();  
        frame.setVisible(true);  
    }  
}
```



Copyright © 2005, Infosys
Technologies Ltd

153

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Although the previous example works fine, it is not a good practice in writing Swing applications like that.

As per sun recommendations the best practice is to create GUI from a thread other than the event-dispatching thread.

For that we have to use `invokeLater(Runnable obj)` method or `invokeAndWait(Runnable obj)` method defined in `SwingUtilities` class. `Runnable` object will have a `run` method called by the event-dispatching thread. `Run` method is the place where we can have GUI creation.

The difference between `invokeLater(Runnable obj)` method or `invokeAndWait(Runnable obj)` method is that `invoke later` returns immediately and `invokeAndWait` waits until `obj.run()` returns.

The code given here is Thread-Safe as per sun recommendation.

Swing Components

The Swing Components are categorized as,

- Top-Level Containers
Applet, Dialog, Frame
- General-Purpose Containers
Panel, Scroll Pane, Split Pane, Tabbed pane, Tool bar.
- Special-Purpose Containers
Internal Frames, Layered Pane, Root Pane
- Basic Controls
Buttons, Combo box, List, Menu, Slider, Snipper, Text Field
- Uneditable Information Displays
Label, Progress bar, Tool tip.
- Interactive Displays of Highly Formatted Information
Color Chooser, File Chooser, Table, Text, Tree.



Copyright © 2005, Infosys
Technologies Ltd

154

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Sun categorizes the Swing components into,

Top-Level Containers

General-Purpose Containers

Special-Purpose Containers

Basic Controls

Uneditable Information Displays and

Interactive Displays of Highly Formatted Information

Components and Containers

- JComponent : It is the base class for all the Swing components except the Top level Containers.
- The JComponent class is an abstract class that extends the AWT Container class
- The visual components are known as the "J" classes and are named with JXxx *convention*. For Example: JLabel, JButton, etc.



Copyright © 2005, Infosys
Technologies Ltd

155

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

JComponent : It is the base class for all the Swing components except the Top level Containers.

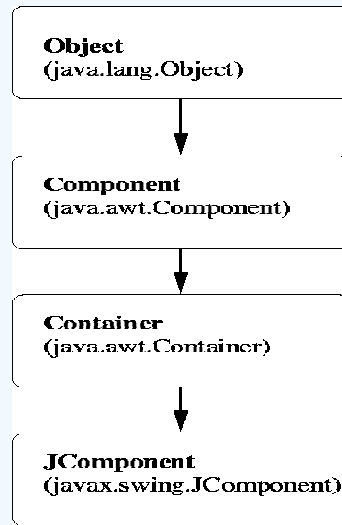
The JComponent class is an abstract class that extends the AWT Container class

The visual components are known as the "J" classes and are named with JXxx *convention*. For Example: JLabel, JButton, etc.

Common functionality provided by JComponents are,

- The visual class hierarchy
- Pluggable look-and-feel support in JComponent
- Keystroke handling in JComponent
- The Border property
- Scrolling support
- Accessibility support
- Internationalization support

JComponent Hierarchy



Copyright © 2005, Infosys
Technologies Ltd

156

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The JComponent class is the top-level class of the Swing component hierarchy. JComponent also inherits from the AWT Container and Component classes. Swing components can inherit various features and capabilities from the JComponent class.

Events

- Events correspond to :
 - Physical actions (E.g.: mouse button down, Key press/release)
 - Logical events (E.g.: gotfocus - receiving focus on a component)
- Event is an encapsulation of some input delivered asynchronously to the application
- The ***java.awt.event***, ***javax.swing.event*** packages defines classes to represent different type of events.



Copyright © 2005, Infosys
Technologies Ltd

157

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

An Action performed at an Instance is called an **Event**. When an event occurs, the underlying Operating System recognizes the event and notifies the respective application. Events are generated by user interaction with the GUI, like a button pressed and they may also occur without user interaction like, when a timer expires, a counter exceeds a value, or an operation is completed. The ***java.awt.event***, ***javax.swing.event*** packages defines classes to represent different type of events.

Event Handling Mechanisms

④ **Delegation event model** – The new approach

④ Concept :

- Source generates the events and sends them to one or more listeners
- Listener waits until it receives an event
- Once received, the listener processes the event and then returns

④ Event handling is totally separated from UI component

④ A UI is able to "delegate" the event handling procedure to a separate piece of code



Copyright © 2005, Infosys
Technologies Ltd

158

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Delegation event model

The modern approach to handling events is based on the delegation event model. Its concept is quite simple: a source generates an event and sends it to one or more Listeners. In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns. The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to “**delegate**” the processing of an event to a separate piece of code.

In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them.

Using the delegation event model

```
public class MyApplication{  
  
    ...  
    Button button = new Button("I'm a button!");  
    button.addActionListener(new MyHandler());  
}  
  
public class MyHandler implements ActionListener{  
    public void actionPerformed(ActionEvent e){  
        numClicks++;  
    }  
}
```

- ④ Create a Listener object which implements the listener interface.
- ④ Associate a Listener with the source

Method in the interface
(ActionListener)



Copyright © 2005, Infosys
Technologies Ltd

159

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

A sample code which shows the implementation of the delegated event model

Steps involved are,

1. Create a Listener object which implements the listener interface.
1. Associate a Listener with the source

Here ActionListener is the listener registered with button, which listens to the ActionEvent and performs the action specified in actionPerformed method when it is notified.

Commonly used Swing Component Classes

- JLabel
- JApplet
- JTextField
- JPasswordField
- JButton
- JCheckBox
- JRadioButton
- JComboBox
- JTextArea
- JScrollPane
- JMenuBar, JMenu and JMenuItem
- JPopupMenu
- JTabbedPane
- JTree



Copyright © 2005, Infosys
Technologies Ltd

160

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Let us have a look on the Swing Component such as,

JLabel

JTextField

JPasswordField

JButton

JCheckBox

JRadioButton

JComboBox

JTextArea

JScrollPane

JMenuBar, JMenu and JMenuItem

JPopupMenu

JTabbedPane

JTree

JLabel

- Swing Labels are instances of the JLabel class.
- JLabel provides a display area for text, an icon, or both.
- The JLabel class extends JComponent.
- The JLabel class does not respond to input events and cannot obtain the input focus.
- Constructors for creating labels are,
 - JLabel(Icon i)
 - JLabel(String text)
 - JLabel(String text, Icon i, int align)
 - JLabel(String text, int align) etc.
- `JLabel jl=new JLabel("Hello World",JLabel.CENTER);`
 - Creates a Swing Label.



Copyright © 2005, Infosys
Technologies Ltd

161

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Swing Labels are instances of the JLabel class.

JLabel provides a display area for text, an icon, or both.

The JLabel class extends JComponent.

The JLabel class does not respond to input events and cannot obtain the input focus.

Constructors for creating labels are,

`JLabel(Icon i)`

`JLabel(String text)`

`JLabel(String text, Icon i, int align)`

Where align for horizontal alignment.

`JLabel jl=new JLabel("Hello World",JLabel.CENTER);`

Creates a Swing Label.

JApplet

- JApplet is a Top-Level Container.
- JApplet extends Applet.
- Applets that use Swings must be a subclass of JApplet.



Copyright © 2005, Infosys
Technologies Ltd

162

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

JApplet is a Top-Level Container.

JApplet extends Applet.

Applets that use Swings must be a subclass of JApplet.

JApplet [Example]

```
import java.awt.*;
import javax.swing.*;

/*<applet code="JLabelDemo" height=500 width=500>
</applet>*/
public class JLabelDemo extends JApplet{
    public void init(){
        try{
            SwingUtilities.invokeLater(new Runnable(){
                public void run(){
                    makeGUI();
                }
            });
        }catch(Exception e){
        }
    }
    public void makeGUI(){
        JLabel jl=new JLabel("Hello World",JLabel.CENTER);
        add(jl);
    }
}
```

To execute the program. [Same as Java Applets]

- Store the program under JLabelDemo.java file name
- Then execute `javac JLabelDemo.java`
- Then execute `appletviewer JLabelDemo.java`

Output



Copyright © 2005, Infosys
Technologies Ltd

163

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is an example to work with JApplet. The execution steps are same as Java Applets.


JTextField and JPasswordField

JTextField

- This displays a Textbox that can hold a Single line editable Text.
- This extends JTextComponent which extends JComponent.
- `JTextField jtf=new JTextField(15);` creates a TextField of column size 15.

JPasswordField

- This displays a Textbox that can hold a Single line editable Text in encrypted form.
- This extends JTextField.
- `JPasswordField jpf=new JPasswordField(10);` creates a password Field of column size 10.
- `jpf.setEchoChar('.');` This specifies the character to echo in the Password field when the user enters a character

-  In both JTextField and JPasswordField an ActionEvent will be notified to the registered Listeners when the use presses "Enter" Key.



Copyright © 2005, Infosys
Technologies Ltd

164

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

JTextField

This displays a Textbox that can hold a Single line editable Text.

This extends JTextComponent which extends JComponent.

`JTextField jtf=new JTextField(15);` creates a TextField of column size 15.

JPasswordField

This displays a Textbox that can hold a Single line editable Text in encrypted form.

This extends JTextField.

`JPasswordField jpf=new JPasswordField(10);` creates a password Field of column size 10.

`jpf.setEchoChar('.');` This specifies the character to echo in the Password field when the user enters a character

In both JTextField and JPasswordField an ActionEvent will be notified to the registered Listeners when the use presses "Enter" Key.

JTextField and JPasswordField [Contd...]

```

public class JTextFieldDemo extends JApplet
    implements ActionListener{
    JTextField jtf=null;
    JPasswordField jpf=null;
    JLabel j11,j12;
    public void init(){
        try{
            SwingUtilities.invokeLater(new Runnable(){
                public void run(){
                    makeGUI();
                }
            });
        }catch(Exception e){
        }
    }

    public void makeGUI(){
        setLayout(new FlowLayout());
        j11=new JLabel("Enter the user name");
        j12=new JLabel("Enter the Password");
        jtf=new JTextField(10);
        jpf=new JPasswordField(10);
        jpf.setEchoChar('.');
        add(j11);
        add(jtf);
        add(j12);
        add(jpf);
        jtf.addActionListener(this);
        jpf.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e){
        ShowStatus(jtf.getText()+jpf.getPassword());
    }
}

```



Copyright © 2005, Infosys
Technologies Ltd

165

ER/CORP/CRS/LA10/003
Version 1.00



Here is a simple program that illustrates the functionality of TextField and Passwordfield. Whenever an ActionEvent occurs, it will be informed to the listeners attached with the respective Component and action specified in actionPerformed will be carried out.

JButton

- JButton extends AbstractButton which extends JComponent
- JButton ok = new JButton("OK"); creates a simple Button.
- An ActionEvent is sent when the button is clicked.
- setActionCommand defined in the AbstractButton specifies the Action command sent as part of the ActionEvent when the button is pressed.
- JButton provides the functionality of a push Button.
- JButton allows an icon, a string or both to be associated with JButton.



Copyright © 2005, Infosys
Technologies Ltd

166

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

JButton extends AbstractButton which extends JComponent

JButton ok = new JButton("OK"); creates a simple Button.

An ActionEvent is sent when the button is clicked.

setActionCommand defined in the AbstractButton specifies the Action command sent as part of the ActionEvent when the button is pressed.

JButton provides the functionality of a push Button.

JButton allows an icon, a string or both to be associated with JButton.

JButton (Contd...)

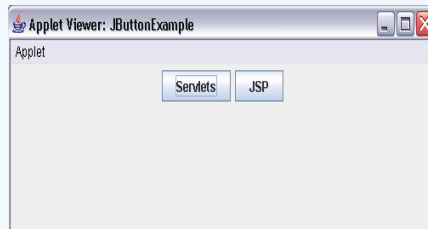
```
JButton jb1,jb2;
```

```
...
```

```
setLayout(new FlowLayout());  
jb1=new JButton("Servlets");  
jb2=new JButton("JSP");  
jb1.setActionCommand("SERVLETS");  
jb2.setActionCommand("JSP");  
jb1.addActionListener(this);  
jb2.addActionListener(this);  
add(jb1);  
add(jb2);
```

```
...
```

```
public void actionPerformed(ActionEvent e){  
    showStatus(e.getActionCommand());  
}
```



Copyright © 2005, Infosys
Technologies Ltd

167

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is a simple example that help you work with buttons.

JCheckBox

- The JCheckBox class extends the JToggleButton class which is for Two state buttons.
- JCheckBox jc1=new JCheckBox("Servlets",true); creates a CheckBox initially Selected.
- JCheckBox jc1=new JCheckBox("Servlets", false) or JCheckBox jc1=new JCheckBox("Servlets") creates a CheckBox initially not selected.
- The state of the CheckBox can be changed by setSelected(boolean state) method. For Checkbox to be selected, state must be true.
- When the CheckBox is selected or deselected, item event will be generated.



Copyright © 2005, Infosys
Technologies Ltd

168

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The JCheckBox class extends the JToggleButton class.

JCheckBox jc1=new JCheckBox("Servlets",true); creates a CheckBox initially Selected.

JCheckBox jc1=new JCheckBox("Servlets", false) or JCheckBox jc1=new JCheckBox("Servlets") creates a CheckBox initially not selected.

The state of the CheckBox can be changed by setSelected(boolean state) method. For Checkbox to be selected, state must be true.

When the CheckBox is selected or deselected, item event will be generated.

JCheckBox (Contd...)

```
JCheckBox jc1,jc2;
...
jc1=new JCheckBox("Servlets",true);
jc2=new JCheckBox("JSP");
jc1.addItemListener(this);
jc2.addItemListener(this);
add(jc1);
add(jc2);
...

public void itemStateChanged(ItemEvent e){
    JCheckBox jc=(JCheckBox)e.getItem();
    if(e.getStateChange()==ItemEvent.SELECTED){
        showStatus(jc.getText()+" - Selected");
    }
    else{
        showStatus(jc.getText()+" - Cleared");
    }
}
```



Copyright © 2005, Infosys
Technologies Ltd

169

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is a simple example on Check boxes.

JRadioButton

- The JRadioButton class extends the JToggleButton class.
- The Difference between Radio button and checkbox is that only one button from a group will be selected.
- The ButtonGroup ensures that only one button is selected at any given time.
 - `JRadioButton jr1=new JRadioButton("Servlets",true);`
 - `JRadioButton jr2=new JRadioButton("JSP");`
 - Creates two Radio Buttons.
 - `ButtonGroup bg=new ButtonGroup();`
 - `bg.add(jr1);`
 - `bg.add(jr2);`
 - Creates a Button group and adds both the radio buttons in to the group so that only of the both will be selected at a time.
- When the Radio Button is selected or deselected, action event will be generated.



Copyright © 2005, Infosys
Technologies Ltd

170

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

The JRadioButton class extends the JToggleButton class.

The Difference between Radio button and checkbox is that only one button from a group will be selected.

The ButtonGroup ensures that only one button is selected at any given time.

```
JRadioButton jr1=new JRadioButton("Servlets",true);
```

```
JRadioButton jr2=new JRadioButton("JSP");
```

Creates two Radio Buttons.

```
ButtonGroup bg=new ButtonGroup();
```

```
bg.add(jr1);
```

```
bg.add(jr2);
```

Creates a Button group and adds both the radio buttons in to the group so that only of the both will be selected at a time.

When the Radio Button is selected or deselected, action event will be generated

JRadioButton(Contd...)

```
JRadioButton jr1,jr2;  
...  
jr1=new JRadioButton("Servlets",true);  
jr2=new JRadioButton("JSP");  
jr1.addActionListener(this);  
jr2.addActionListener(this);  
add(jr1);  
add(jr2);  
ButtonGroup bg=new ButtonGroup();  
bg.add(jr1);  
bg.add(jr2);  
  
public void actionPerformed(ActionEvent e){  
    showStatus(e.getActionCommand());  
}
```



Copyright © 2005, Infosys
Technologies Ltd

171

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is a simple example on Radio Buttons.

JComboBox

- JComboBox is a combination of text field and a drop down list which allows the user to select an entry from the list.
- JComboBox extends JComponent.
- JComboBox displays one entry at a time.
- JComboBox `jc1=new JComboBox();` creates a JComboBox.
- Items can be added to the comboBox by using “add” method.
- E.g:
 - `jc1.addItem("Servlets");`
 - `jc1.addItem("JSP");`
- When a new selection is made, item event will be generated



JComboBox [Contd...]

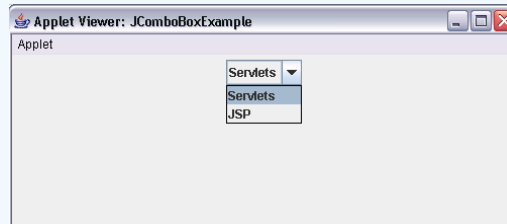
```
JComboBox jc1;
```

```
...
```

```
jc1=new JComboBox();  
jc1.addItem("Servlets");  
jc1.addItem("JSP");  
jc1.addItemListener(this);  
add(jc1);
```

```
...
```

```
public void itemStateChanged(ItemEvent e){  
    showStatus((String)e.getItem());  
}
```



Copyright © 2005, Infosys
Technologies Ltd

173

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is an example for JComboBox.

TextArea and JScrollPane

JTextArea

- It is for handling multi-line text.
- It extends JTextComponent class.
- JTextArea doesn't manage scrolling, but implements the swing Scrollable interface.
This allows it to be placed inside a JScrollPane if scrolling behavior is need, and used directly otherwise.

JScrollPane

- This extends JComponent.
- This provides a rectangular area into which a component can be placed.
- Horizontal and Vertical Scrollbars may be provided whenever necessary.



Copyright © 2005, Infosys
Technologies Ltd

174

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

JTextArea and JScrollPane [Contd...]

```
JTextArea textArea = new JTextArea("Fill your details here",7,25);
textArea.setFont(new Font("Serif", Font.ITALIC, 16));
textArea.setLineWrap(true);
textArea.setWrapStyleWord(true);
int v=ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS;
int h=ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS;
JScrollPane js=new JScrollPane(textArea,v,h);
add(js);
```



Copyright © 2005, Infosys
Technologies Ltd

175

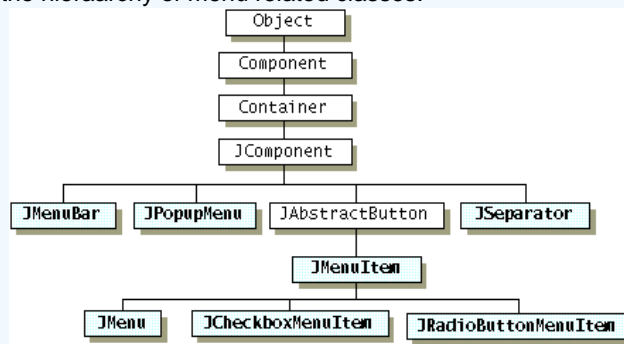
ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is an example where the textArea is placed inside a JScrollPane.

Working with Menus

- Menus- these are the components that are not part of any other components.
- They can be the part of MenuBar or can appear as popup menu.
- Here is the hierarchy of menu related classes.



Copyright © 2005, Infosys
Technologies Ltd

176

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Menus- these are the components that are not part of any other components.
They can be the part of MenuBar or can appear as popup menu.
Here is the hierarchy of menu related classes

JMenuBar, JMenu, JMenuItem

JMenuBar

- `JMenuBar menuBar=new JMenuBar();` creates a MenuBar named menuBar.
- `setJMenuBar(JMenuBar menuBar)` method in `JFrame` and `JApplet` class sets the menuBar to the respective `TopLevel` container.

JMenu

- `JMenu file=new JMenu("File");` creates a menu with the name file and Text "File" associates with it.
- `add(JMenuItem menuItem)` adds a Menu Item to the end of the Menu.
- `add(Component c)` adds a Component ex: `JMenu` [which forms the sub menu] to the end of this menu.
- Menus can be added to the MenuBar with the help of `add` method in `JMenuBar`

JMenuItem

- `JMenuItem menuItem=new JMenuItem("New");` creates a Menu Item.
- It is similar to a button . When selected action associated with the menu item will get executed.



Copyright © 2005, Infosys
Technologies Ltd

177

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

JMenuBar

`JMenuBar menuBar=new JMenuBar();` creates a MenuBar named menuBar.
`setJMenuBar(JMenuBar menuBar)` method in `JFrame` and `JApplet` class sets the menuBar to the respective `TopLevel` container.

JMenu

`JMenu file=new JMenu("File");` creates a menu with the name file and Text "File" associates with it.
`add(JMenuItem menuItem)` adds a Menu Item to the end of the Menu.
`add(Component c)` adds a Component ex: `JMenu` [which forms the sub menu] to the end of this menu.
Menus can be added to the MenuBar with the help of `add` method in `JMenuBar`

JMenuItem

`JMenuItem menuItem=new JMenuItem("New");` creates a Menu Item.
It is similar to a button . When selected action associated with the menu item will get executed.

JMenuBar,JMenu,JMenuItem [Contd...]

- Menus can be accessed through keyboard using mnemonics and accelerators.
- Mnemonics → this helps us to navigate the menu hierarchy using Keyboard.
- Accelerators → offers Keyboard shortcuts to activate MenuItems.
- **setMnemonic**(int mnemonic) in **AbstractButton** class helps in setting mnemonics to JMenus and JMenuItem.
 - E.g: `file.setMnemonic(KeyEvent.VK_F);`
 - This adds Mnemonic to Menu file.
- **setAccelerator**(KeyStroke keyStroke) Sets the key combination which invokes the menu item's action listeners without navigating the menu hierarchy.
 - E.g:
`menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_1,ActionEvent.ALT_MASK));`
 - This makes the menuItem activated when Alt and 1 keys are pressed.



Copyright © 2005, Infosys
Technologies Ltd

178

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Menus can be accessed through keyboard using mnemonics and accelerators.

Mnemonics → this helps us to navigate the menu hierarchy using Keyboard.

Accelerators → offers Keyboard shortcuts to activate MenuItems.

setMnemonic(int mnemonic) in **AbstractButton** class helps in setting mnemonics to JMenus and JMenuItem.

E.g: `file.setMnemonic(KeyEvent.VK_F);`

This adds Mnemonic to Menu file.

setAccelerator(KeyStroke keyStroke) Sets the key combination which invokes the menu item's action listeners without navigating the menu hierarchy.

E.g:

`menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_1,ActionEvent.ALT_MASK));`

This makes the menuItem activated when Alt and 1 keys are pressed.

JMenuBar,JMenu,JMenuItem [Contd...]

```

JMenuBar menuBar;
JMenu file,view,toolBar;
JMenuItem menuItem;

...
menuBar=new JMenuBar();
file=new JMenu("File");
file.setMnemonic(KeyEvent.VK_F);
menuItem=new JMenuItem("New",KeyEvent.VK_N);
menuItem.addActionListener(this);
file.add(menuItem);
menuItem=new JMenuItem("Open",KeyEvent.VK_O);
menuItem.addActionListener(this);
file.add(menuItem);
menuItem=new JMenuItem("Close",KeyEvent.VK_C);
menuItem.addActionListener(this);
file.add(menuItem);
menuBar.add(file);
view=new JMenu("View");
view.setMnemonic(KeyEvent.VK_V);
toolBar=new JMenu("ToolBar");
toolBar.setMnemonic(KeyEvent.VK_T);
menuItem=new JMenuItem("Standard",KeyEvent.VK_S);

menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_I,ActionEvent.ALT_MASK));
menuItem.addActionListener(this);
toolBar.add(menuItem);
menuItem=new JMenuItem("Formatting",KeyEvent.VK_F);
menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_2,ActionEvent.ALT_MASK));
menuItem.addActionListener(this);
toolBar.add(menuItem);
toolBar.addSeparator();
menuItem=new JMenuItem("Drawing",KeyEvent.VK_D);
menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_3,ActionEvent.ALT_MASK));
menuItem.addActionListener(this);
toolBar.add(menuItem);
menuItem=new JMenuItem("Picture",KeyEvent.VK_P);
menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_4,ActionEvent.ALT_MASK));
menuItem.addActionListener(this);
toolBar.add(menuItem);
view.add(toolBar);
menuBar.add(view);
setJMenuBar(menuBar);

...
public void actionPerformed(ActionEvent e){
    showStatus(e.getActionCommand());
}

```



Copyright © 2005, Infosys
Technologies Ltd

179

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is an example on Menus.

JMenuBar,JMenu,JMenuItem [Contd...]



When you run the above code you will get the menu displays as shown here.

JPopupMenu

- JPopupMenu is created to provide a menu anywhere desired.
- These menus are invisible until the user makes a platform-specific mouse action such as pressing the right mouse button over a pop-up enabled component.
- Popup menus are created in the same way as menu bar the difference is instead of JMenuBar, JPopupMenu is used.
- JPopupMenu menuBar=new JPopupMenu(); creates a Propup Menu.
- This can have JMenus, which in turn can have JMenus and JMenuItem.
- To associate a popup menu with a component we have to register a mouse listener to that component which will listen to the user request for popup menu.



Copyright © 2005, Infosys
Technologies Ltd

181

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

JPopupMenu is created to provide a menu anywhere desired.

These menus are invisible until the user makes a platform-specific mouse action such as pressing the right mouse button over a pop-up enabled component.

Popup menus are created in the same way as menu bar the difference is instead of JMenuBar, JPopupMenu is used.

JPopupMenu menuBar=new JPopupMenu(); creates a Propup Menu.

This can have JMenus, which in turn can have JMenus and JMenuItem.

To associate a popup menu with a component we have to register a mouse listener to that component which will listen to the user request for popup menu.

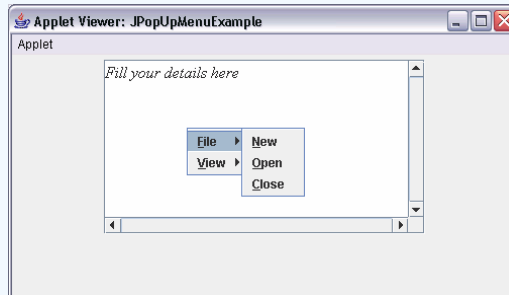
JPopupMenu [Contd..]

```
JPopupMenu menuBar=new JPopupMenu();
...

JTextArea textArea = new JTextArea("Fill your details
here",7,25);
textArea.setFont(new Font("Serif", Font.ITALIC, 16));
textArea.setLineWrap(true);
textArea.setWrapStyleWord(true);
int v=ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS;
int h=ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS;
JScrollPane js=new JScrollPane(textArea,v,h);
add(js);

textArea.addMouseListener(new MouseAdapter(){
    public void mousePressed(MouseEvent e){
        showPopup(e);
    }
    public void mouseReleased(MouseEvent e){
        showPopup(e);
    }
});
...

public void showPopup(MouseEvent e){
    if(e.isPopupTrigger()){
        menuBar.show(e.getComponent(),e.getX(),e.getY());
    }
}
```



Copyright © 2005, Infosys
Technologies Ltd

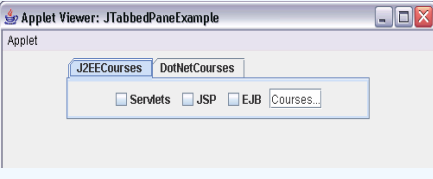
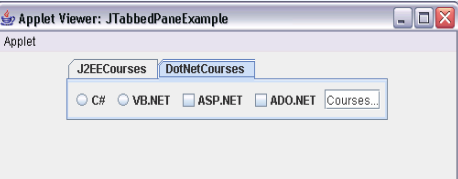
182

ER/CORP/CRS/LA10/003
Version 1.00


Infosys®


This shows how to have a Popup Menu in your application.

JTabbedPane

- This is a simple example for a tabbed pane.
- It can have n number of tabs each having its own components.
- Only one tab can be selected at a time.
- Each tab has its own components.
- **addTab**(String title, Component component) is used to add a Tab.






Copyright © 2005, Infosys Technologies Ltd

183

ER/CORP/CRS/LA10/003
Version 1.00



See the diagram. This is how the tabbed pane looks.

It can have n number of tabs each having its own components.

Only one tab can be selected at a time.

Each tab has its own components.

addTab(String title, Component component) is used to add a Tab.

JTabbedPane [Contd...]

```

JTabbedPane jtp=new JTabbedPane();
jtp.addTab("J2EECourses",new J2EE());
jtp.addTab("DotNetCourses",new DotNet());
add(jtp);

...
class J2EE extends JPanel implements ItemListener{
    JTextField jtl;
    JCheckBox jc1,jc2,jc3;
    public J2EE(){
        jc1=new JCheckBox("Servlets");
        jc2=new JCheckBox("JSP");
        jc3=new JCheckBox("EJB");
        jtl=new JTextField("Courses...");
        jc1.addItemListener(this);
        jc2.addItemListener(this);
        jc3.addItemListener(this);
        add(jc1);
        add(jc2);
        add(jc3);
        add(jtl);
    }

    public void itemStateChanged(ItemEvent e){
        String courses="U have selected the courses ";
        if(jc1.isSelected()){
            courses+=jc1.getText();
            courses+=" ";
        }
        if(jc2.isSelected()){
            courses+=jc2.getText();
            courses+=" ";
        }
        if(jc3.isSelected()){
            courses+=jc3.getText();
            courses+=" ";
        }
        jtl.setText(courses);
    }
}

class DotNet extends JPanel implements ActionListener,ItemListener{
    -
}

```



Copyright © 2005, Infosys
Technologies Ltd

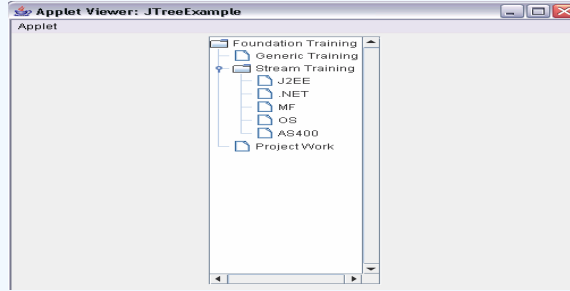
184

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is the code for the previously shown output.

JTree



- This is a simple example for a JTree.
- This presents the data in hierarchical form.
- Individual sub trees in the display can be expanded or collapsed.
- DefaultMutableTreeNode class is used to create a tree node.
- The node created from it will not have parents. To create hierarchy add method of DefaultMutableTreeNode can be used.
- `JTree jt1=new JTree(JTree root);` creates a Tree with the root node as root.



Copyright © 2005, Infosys
Technologies Ltd

185

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

This is a simple example for a JTree.

This presents the data in hierarchical form.

Individual sub trees in the display can be expanded or collapsed.

DefaultMutableTreeNode class is used to create a tree node.

The node created from it will not have parents. To create hierarchy add method of DefaultMutableTreeNode can be used.

`JTree jt1=new JTree(JTree root);` creates a Tree with the root node as root.

JTree [Contd...]

```

JTree jtl;
JScrollPane js1;
...
DefaultMutableTreeNode root=new
    DefaultMutableTreeNode("Foundation Training");
DefaultMutableTreeNode c1=new DefaultMutableTreeNode("Generic
    Training");
DefaultMutableTreeNode c2=new DefaultMutableTreeNode("Stream
    Training");
DefaultMutableTreeNode c3=new DefaultMutableTreeNode("Project
    work");

    root.add(c1);
    root.add(c2);
    root.add(c3);

DefaultMutableTreeNode c2c1=new DefaultMutableTreeNode("J2EE");
DefaultMutableTreeNode c2c2=new DefaultMutableTreeNode(".NET"); ...
DefaultMutableTreeNode c2c3=new DefaultMutableTreeNode("MF");
DefaultMutableTreeNode c2c4=new DefaultMutableTreeNode("OS");
DefaultMutableTreeNode c2c5=new DefaultMutableTreeNode("AS400");

    c2.add(c2c1);
    c2.add(c2c2);
    c2.add(c2c3);
    c2.add(c2c4);
    c2.add(c2c5);

jtl=new JTree(root);
jtl.addMouseListener(new MouseAdapter(){
    public void mouseClicked(MouseEvent me){
        TreePath
        tp=jtl.getPathForLocation(me.getX(),me.getY());
        if(tp!=null){
            showStatus(tp.toString());
        }
        else{
            showStatus("");
        }
    }
});
jtl.addTreeExpansionListener(this);
int v=ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS;
int h=ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS;
JScrollPane js=new JScrollPane(jtl,v,h);
add(js);

public void treeCollapsed(TreeExpansionEvent tee){
    display(tee);
}
public void treeExpanded(TreeExpansionEvent tee){
    display(tee);
}
public void display(TreeExpansionEvent tee){
    showStatus(""+tee.getPath());
}
}

```



Copyright © 2005, Infosys
Technologies Ltd

186 ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Here is a simple example for using JTree

Layout Managers

- Layout Managers controls the size and position (layout) of components inside a Container object.
- java.awt package provides a set of predefined Layout Managers that implements java.awt.LayoutManager interface.
- Every Swing container has a predefined layout manager as its default
- container.setLayout method is used to change the Layout Manager associated with the particular component.
- Predefined Layout Managers available are,
 - FlowLayout
 - BorderLayout
 - GridLayout
 - GridBagLayout
 - CardLayout
- Refer to JavaDocs to explore more on this.



Copyright © 2005, Infosys
Technologies Ltd

187

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

Layout Managers controls the size and position (layout) of components inside a Container object.

java.awt package provides a set of predefined Layout Managers that implements java.awt.LayoutManager interface.

Every Swing container has a predefined layout manager as its default

container.setLayout method is used to change the Layout Manager associated with the particular component.

Predefined Layout Managers available are,

FlowLayout
BorderLayout
GridLayout
GridBagLayout
CardLayout

Refer to JavaDocs to explore more on this.

Different Layouts

There are **5** different layouts available

FlowLayout

- The components are placed horizontally one after another and then move to the next line

GridLayout

- The components are placed in a grid (rows, columns)

BorderLayout

- 5 components can be added at the most
- The 5 borders are North, South, East, west and Center
- If not specified the default position is center in Border Layout



Copyright © 2005, Infosys
Technologies Ltd

188

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

There are **5** different layouts available

FlowLayout

The components are placed horizontally one after another and then move to the next line

GridLayout

The components are placed in a grid of (rows, columns)

BorderLayout

5 components can be added at the most in a border layout.

The 5 borders are North, South, East, west and Center

If not specified the default position is center in Border Layout

Different Layouts [Contd...]

CardLayout

- The CardLayout places components/containers **on top of each other** like a deck of cards
- Only one is visible at a time
- Every card is made visible using the method **show()**

GridBagLayout

- Most powerful and flexible
- It is a more advanced form of GridLayout where components can be placed horizontally and vertically
- Components can be of different sizes and they can span multiple cells in the grid



Copyright © 2005, Infosys
Technologies Ltd

189

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

CardLayout

The CardLayout places components/containers **on top of each other** like a deck of cards

Only one is visible at a time

Every card is made visible using the method **show()**

GridBagLayout

Most powerful and flexible

It is a more advanced form of GridLayout where components can be placed horizontally and vertically

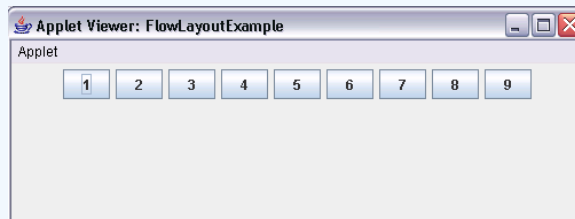
Components can be of different sizes and they can span multiple cells in the grid

Visual Effects of Layout

```
for(int i=0;i<9;i++){  
    JButton jb=new JButton(""+(i+1));  
    add(jb);  
}
```

Ⓔ Consider the above code. It will create and add 9 buttons to the container say JApplet.

Ⓔ When `setLayout(new FlowLayout());` is set the display will be like as follows



Copyright © 2005, Infosys
Technologies Ltd

190

ER/CORP/CRS/LA10/003
Version 1.00

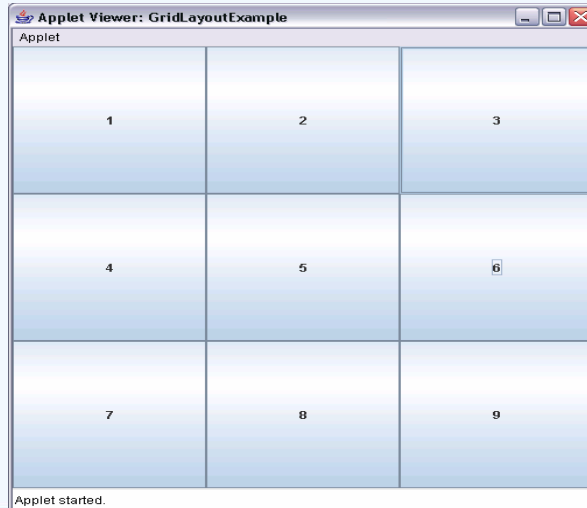
Infosys®

Let us now discuss how the layout managers changes the display. Consider the given code. It will create and add 9 buttons to the container say JApplet.

When `setLayout(new FlowLayout());` is set the display will be like as shown here.

Visual Effects of Layout [Grid Layout]

- When `setLayout(new GridLayout(3,3));` is set then the buttons will be arranged as.



Copyright © 2005, Infosys
Technologies Ltd

191

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

When `setLayout(new GridLayout(3,3));` is set then the buttons will be arranged as shown here.



Basic Java Programming

Conclusion



Conclusion

- Basic programming constructs
- Interfaces & packages
- Applets
- Exception Handling
- Swing Programming



Copyright © 2005, Infosys
Technologies Ltd

193

ER/CORP/CRS/LA10/003
Version 1.00

Infosys®

With this e-learning you have learnt,

What are the Basic programming constructs available in Java.

The concept of Interfaces & packages

How to write Applets

How to perform Exception Handling

How to build GUI's using swings

This course is a base course which help you learn the Basic concepts of java.

Hope you enjoyed learning.

Thank you.