Hello and Welcome! This is a course on Java Programming presented by the Java Technical Competency Development Team.

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 concepts of Java Runtime Environment to under stand the important constituents of JVM .

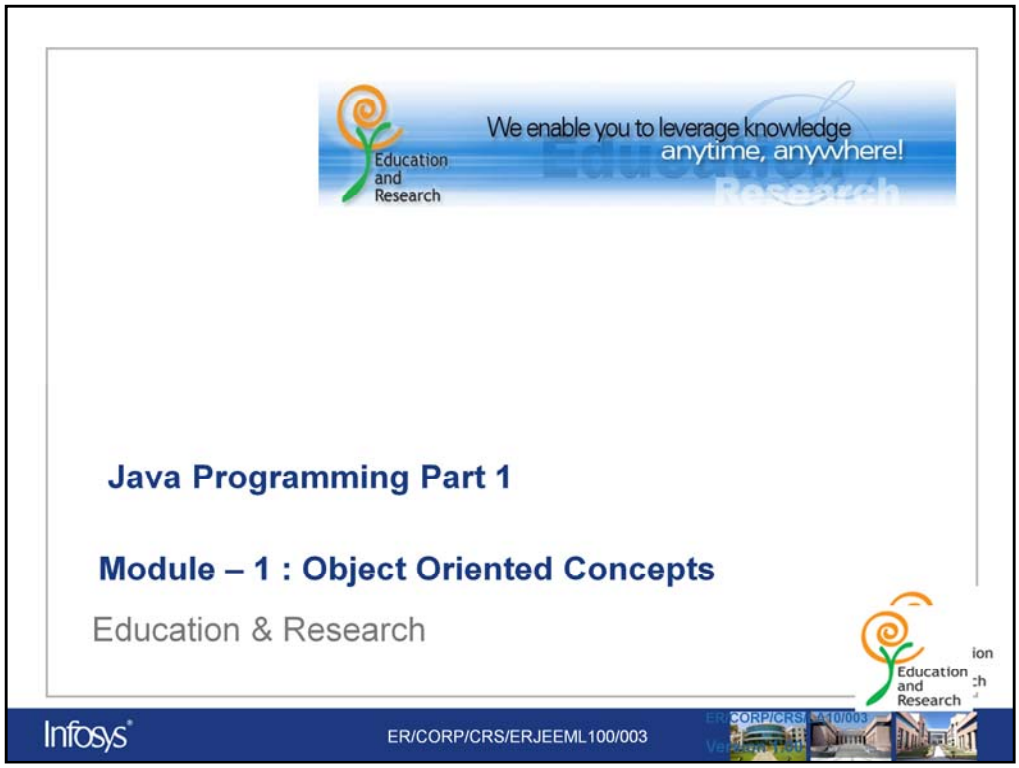
Infosys 2 We enable you to leverage knowledge anytime, anywhere!

By the end of the course, you will have the

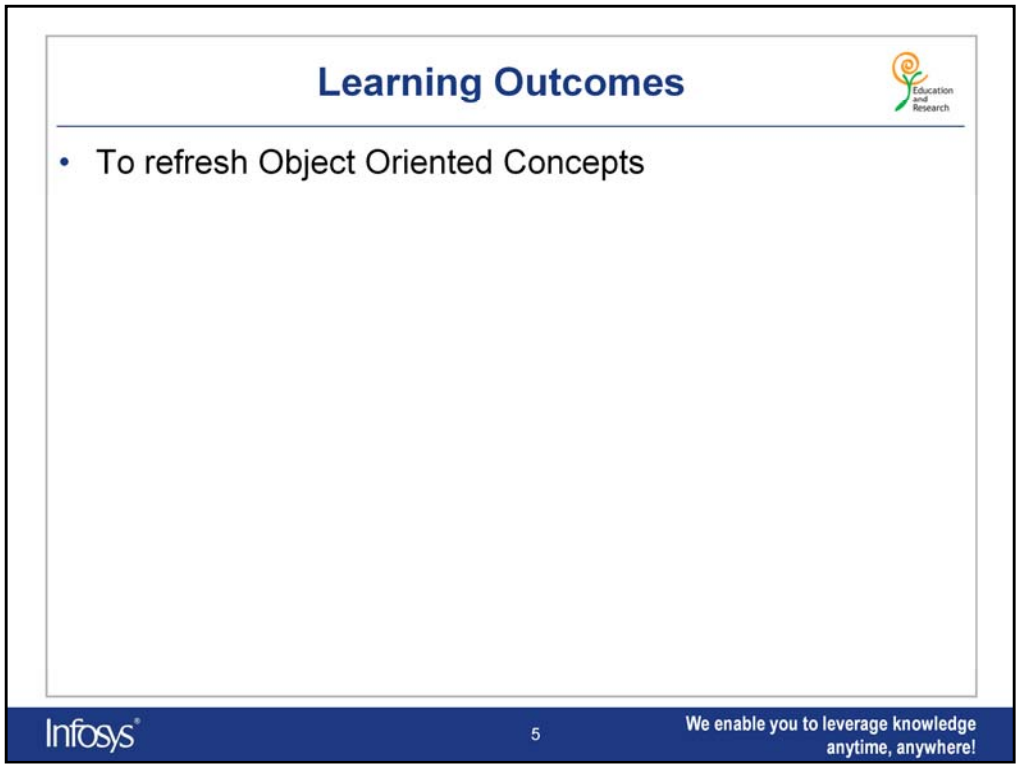
- Knowledge of the Java Architecture, Ability to write Java programs using keywords, control statements and various other constructs
- Ability to use the classes declared in the class libraries, and to create custom libraries.
- Ability to write programs that handles exceptions.
- Ability to understand Java Runtime Environment .

Session Plan

- Object Oriented Concepts
- Java Architecture
- Basic Constructs
- Java Class Libraries
- Exception Handling
- Java Runtime Environment



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



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 *template* or *prototype* that defines the member variables and the member 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 very useful to model real-world objects which we see in our daily life.

Infosys 6 We enable you to leverage knowledge anytime, anywhere!

What is an Object?
Object is an instance of a class.
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.

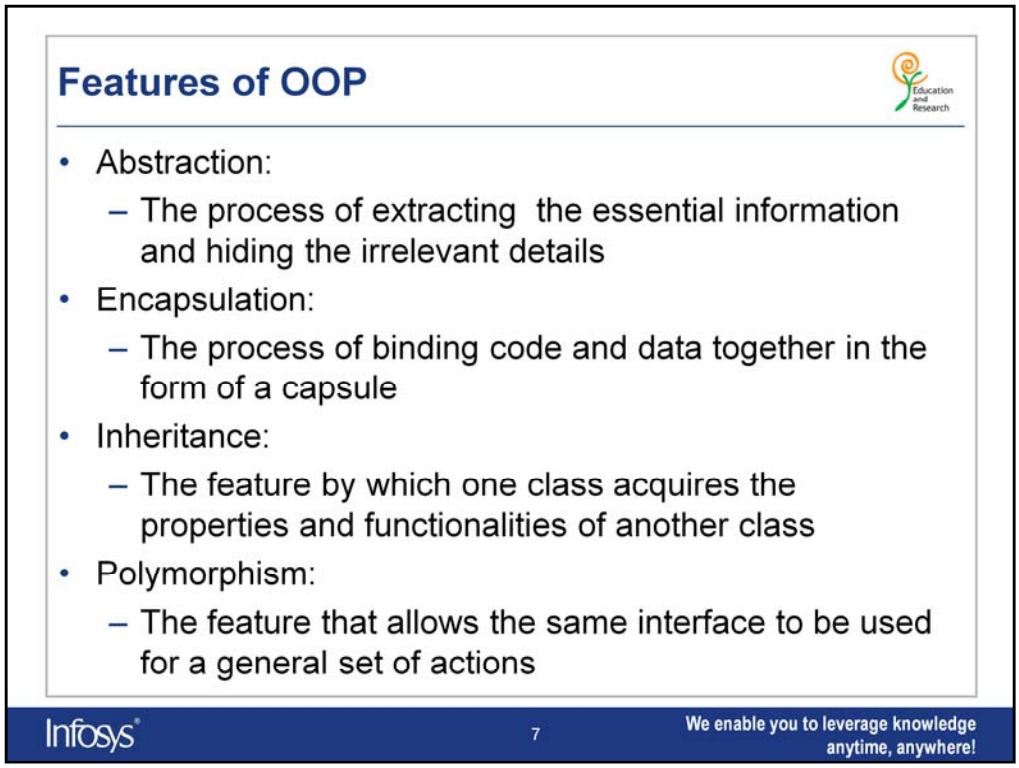
A class is a *blueprint* or *prototype* that defines the variables and the methods (or funtions)common to all objects of a certain kind.

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 obeject 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 Object Oriented Programming

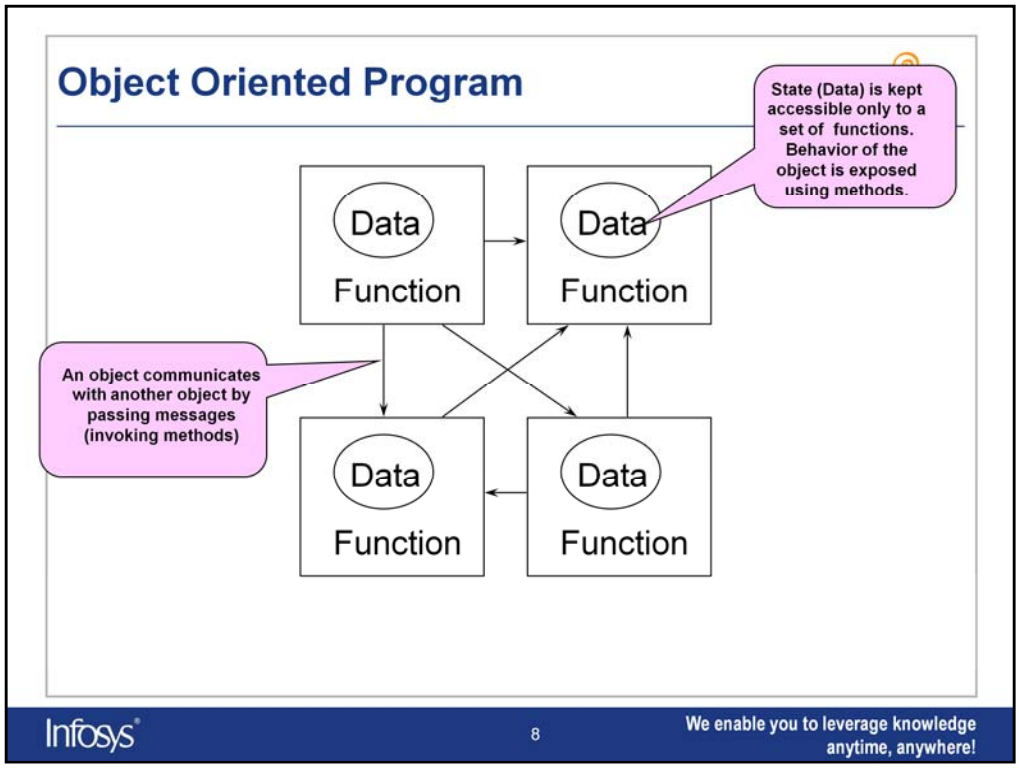
Abstraction is the process of exposing the relevant things and ignoring the irrelevant details. The easiest way to understand and appreciate this concept of handling complexity is by studying the example of Globe, a model/prototype of earth that is used by students to understand its geography. Globe 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 a way keeping the data and the methods which manipulates that data.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 through which one class acquires the properties and behavior of another class. Inheritance provides the idea of reusability of code and each sub class defines only those features that are unique to it.

Polymorphism is a basic feature that allows one interface that can 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 or 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)

Infosys 9 We enable you to leverage knowledge anytime, anywhere!

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

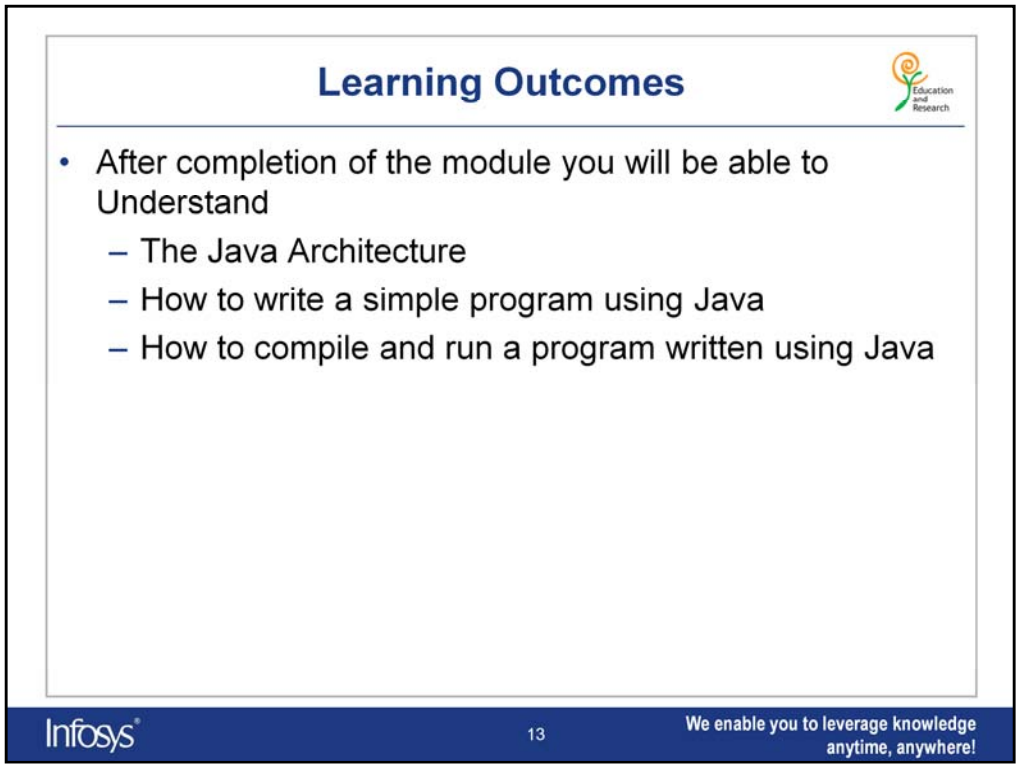
Object Oriented Programming vs Procedural Programming(contd)

Program modules are connected through parameter passing call	Program modules are integrated parts of complete program. Objects interact by passing messages to each other
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

Object Oriented Programming vs Procedural Programming(contd)

Passive data structures used by active methods	Intelligent data structures (object) encapsulates all passive procedures
Procedural languages: C, COBOL, PASCAL	OO Languages: C++, Small Talk, Java, C#

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

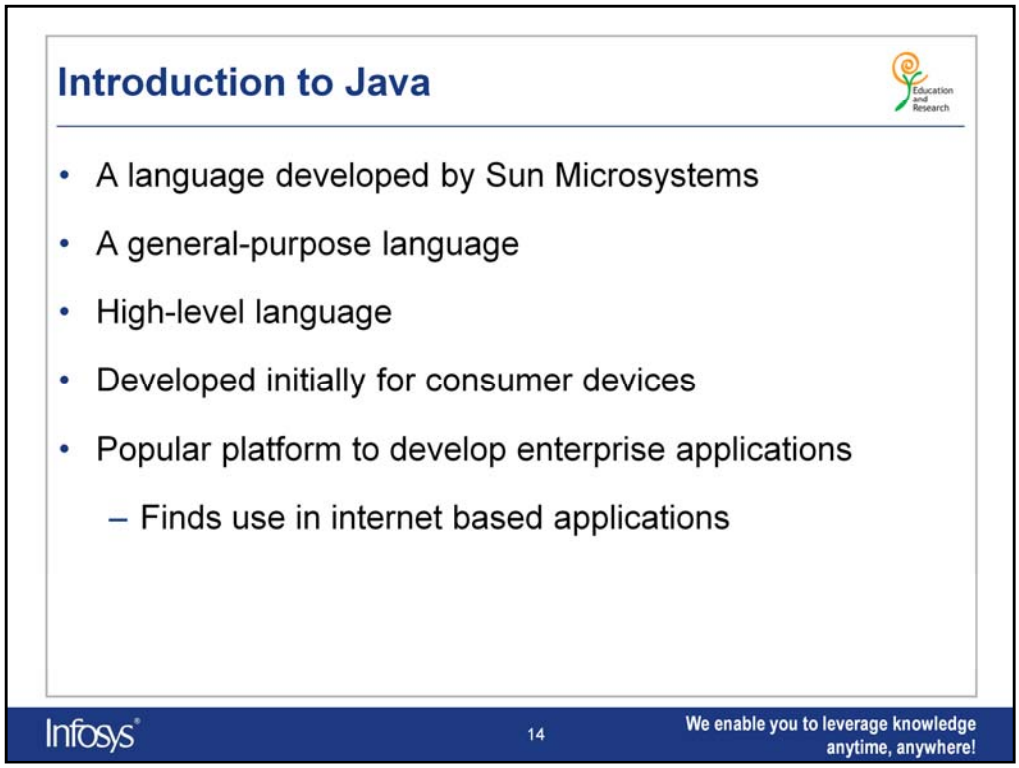


After completion of the module you will be able to Understand

The Java Architecture

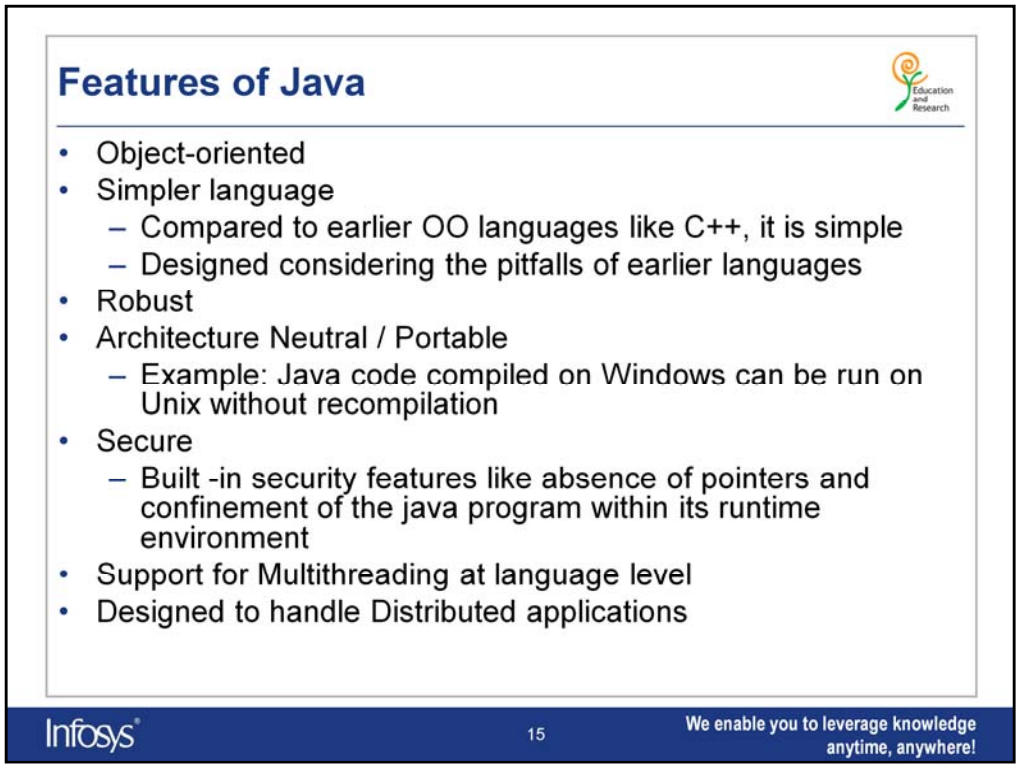
How to write a simple program using Java

How to compile and run a program written using Java



Java was developed 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. Java language was initially called “Oak” but was renamed as “Java” in 1995.Based after C++, the Java language is a small, simple, and portable across platforms and operating systems, both at the source level and at the binary level, which means that the same Java program can be ported and make run on any machine.

Source: Sun Microsystems



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 failing any program are : (1) memory management pitfalls (2) unstructured handling of 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 levels that is binary and the source 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 developing 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 mutithreading 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 mutithreading.

Java is capable for the **distributed environment** of the programs on the Internet. Java allows objects to passed as values and execute 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

Infosys 16 We enable you to leverage knowledge anytime, anywhere!

Java is a platform independent language.

A platform is the environment of hardware and software in which a any program specific to that language 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)

- 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
- The byte code will be in a file with extension .class
- The .class file that is generated is the machine code of this processor.
  - Byte code is a binary language

Infosys 17 We enable you to leverage knowledge anytime, anywhere!

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…)

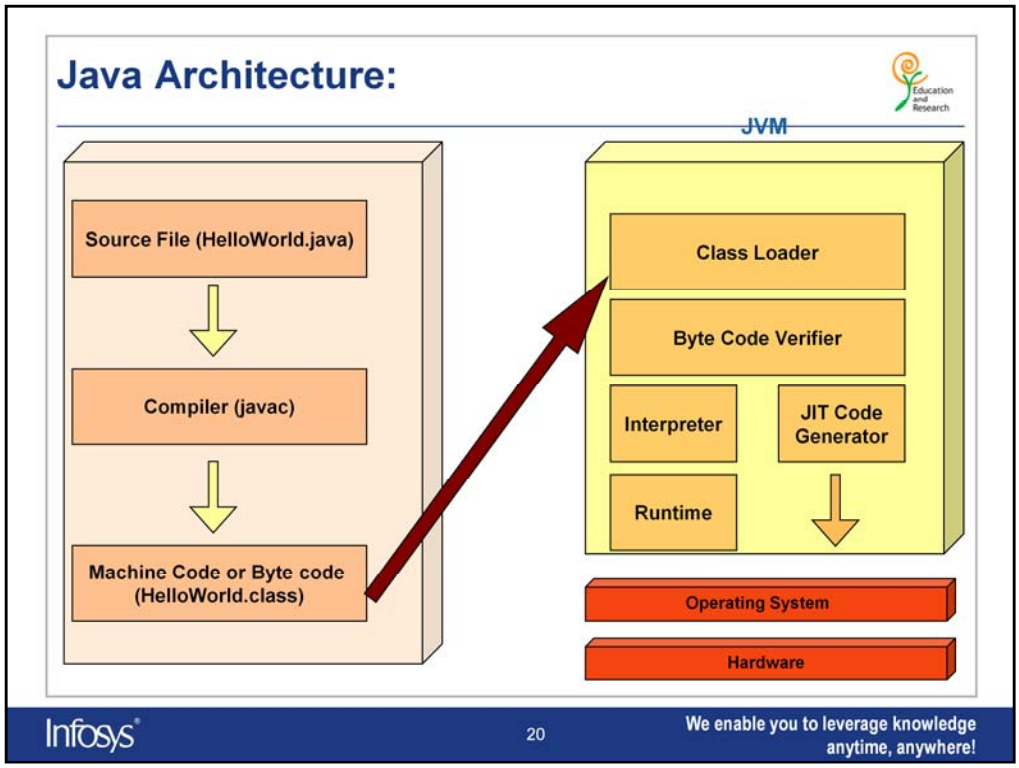
- The byte code is interpreted by the JVM
- JVM varies from platform to platform, or, JVM is platform dependent
- JVM can be considered as a processor purely implemented with software.
- The interface that the JVM has to the .class file remains the same irrespective of the underlying platform.
  - This 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

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

Java Architecture- (Contd…)

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

  Java can de-allocate objects memory 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

Infosys 21 We enable you to leverage knowledge anytime, anywhere!

When an object is not being used for longer duration ,memory occupied by it can be reclaimed. This is done by Garbage Collection. Garbage Collection occurs in a periodic fashion 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.

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.

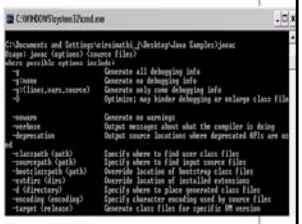
Infosys 22 We enable you to leverage knowledge anytime, anywhere!

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

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:.`



Infosys 23 We enable you to leverage knowledge anytime, anywhere!

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.

Environment variables used by JVM(contd)

- **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'

Source File Layout - Hello World

- We will have the source code first
- Type this into any text editor

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

- Save this as HelloWorldApp.java
- **Important :**
  - Take care!! cAsE of file name matters

Infosys 25 We enable you to leverage knowledge anytime, anywhere!

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 by using a plain text editor like notepad or textpad, or in an editor that is capable to save files in plain ASCII without any formatting of characters. It can contains one or more class definitions. In Java all code must be encapsulated within a class. Even the main method has to be within 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].

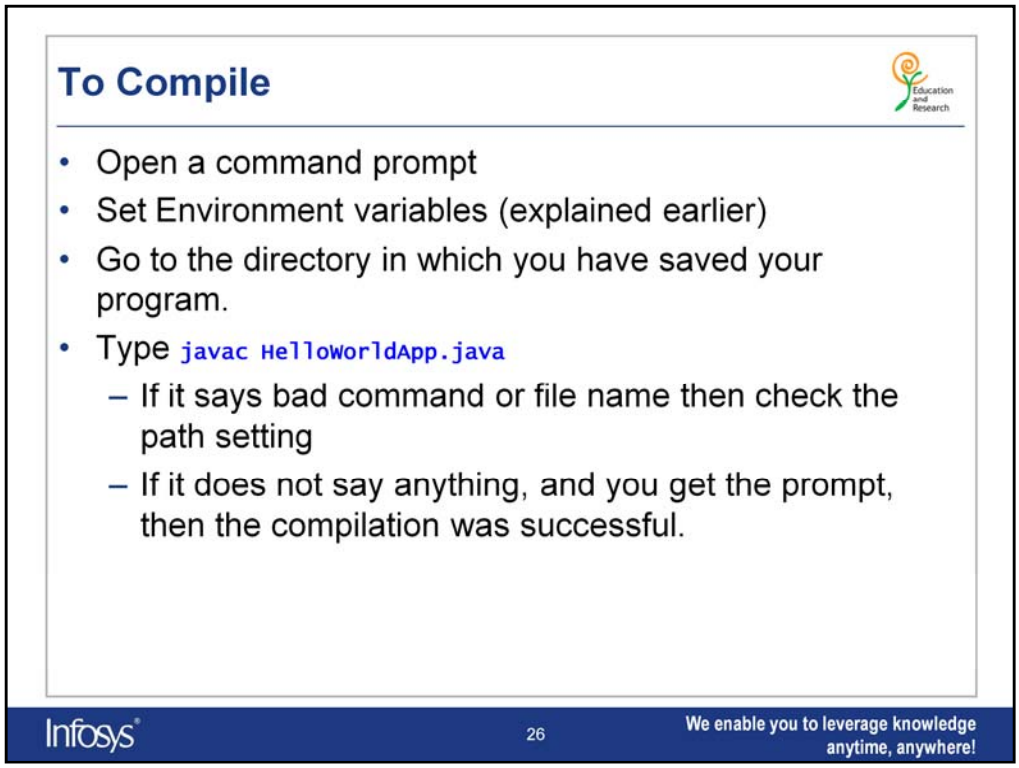
The extension must be .java

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 are mandatory to be written. If they are present, they must appear in the following order:

- **Package Declaration (optional – only one)**
- **Import Statements (optional – any number)**

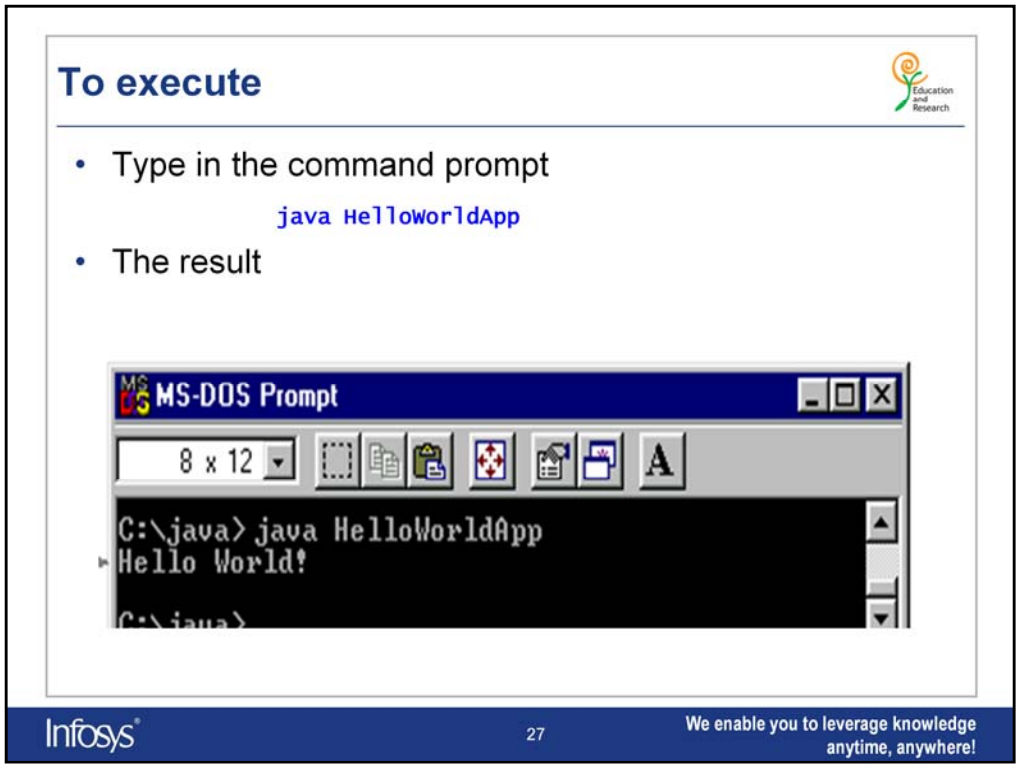


Compiling a Program

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.

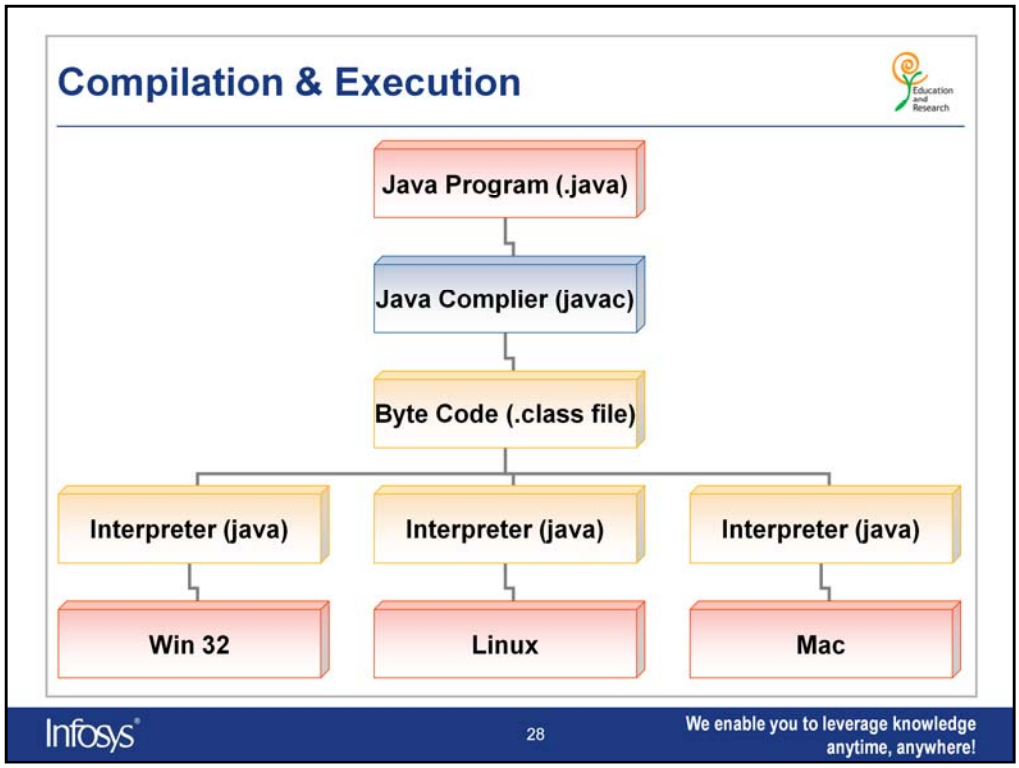


Executing a Program

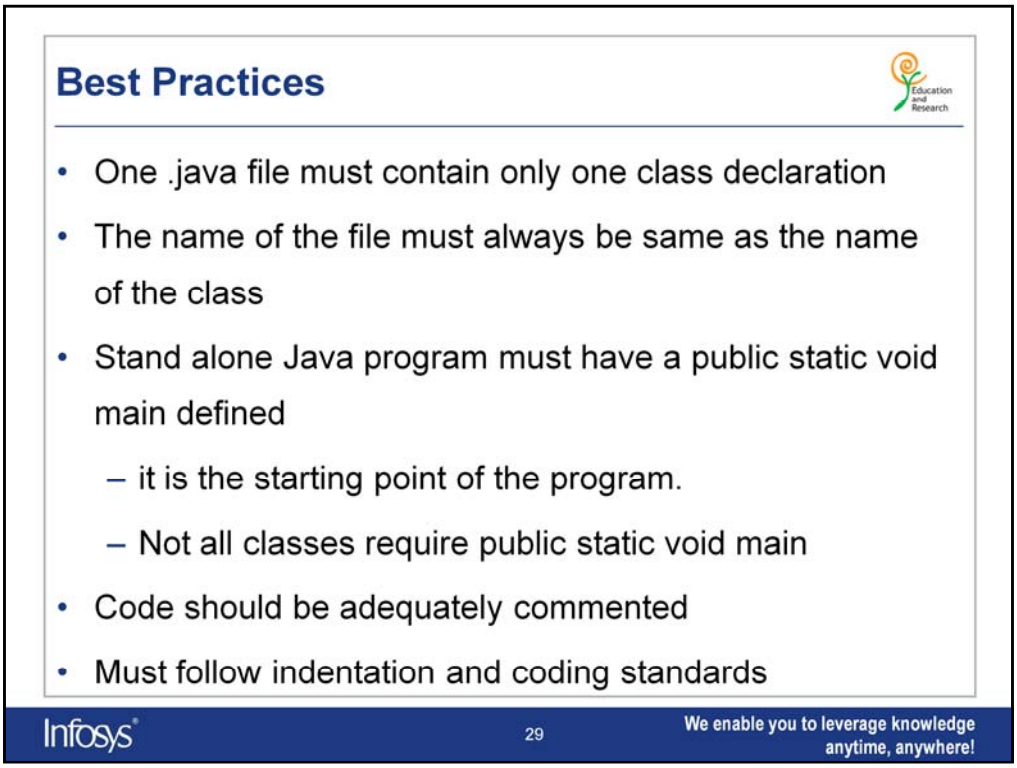
To actually run the program, you must use the Java interpreter, called **java.** To do so, pass the class name at the command-line:

For Example: Type "java HelloWorldApp"

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



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



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

One .java file must contain only one class declaration

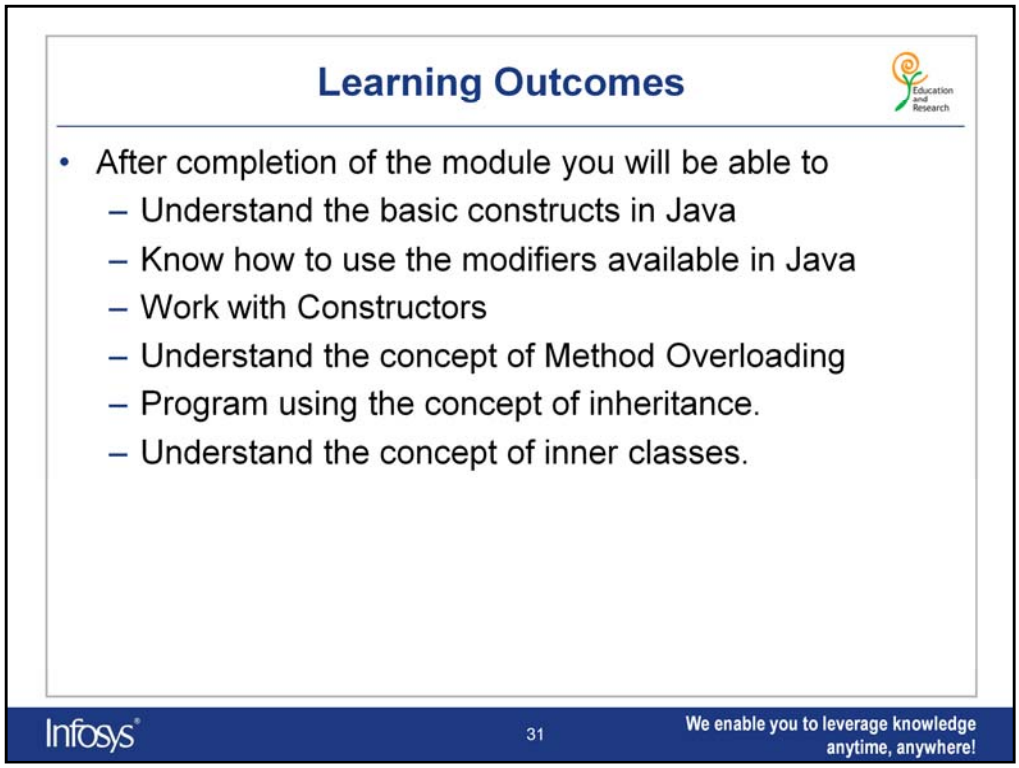
The name of the file must always be same as the name of the class

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

Code should be adequately commented

Must follow indentation and coding standards

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



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	

Infosys 32 We enable you to leverage knowledge anytime, anywhere!

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

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

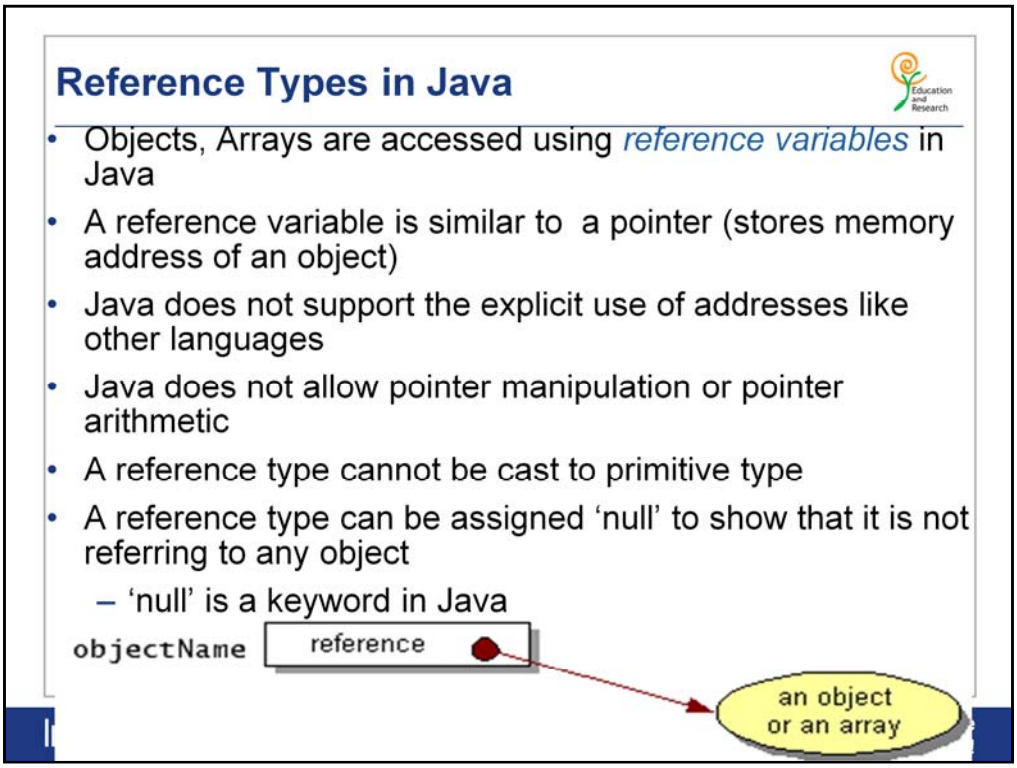
Infosys 34 We enable you to leverage knowledge anytime, anywhere!

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



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;
```

name

- In Java variables can be declared anywhere in the program

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

type

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

Infosys 36 We enable you to leverage knowledge anytime, anywhere!

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.

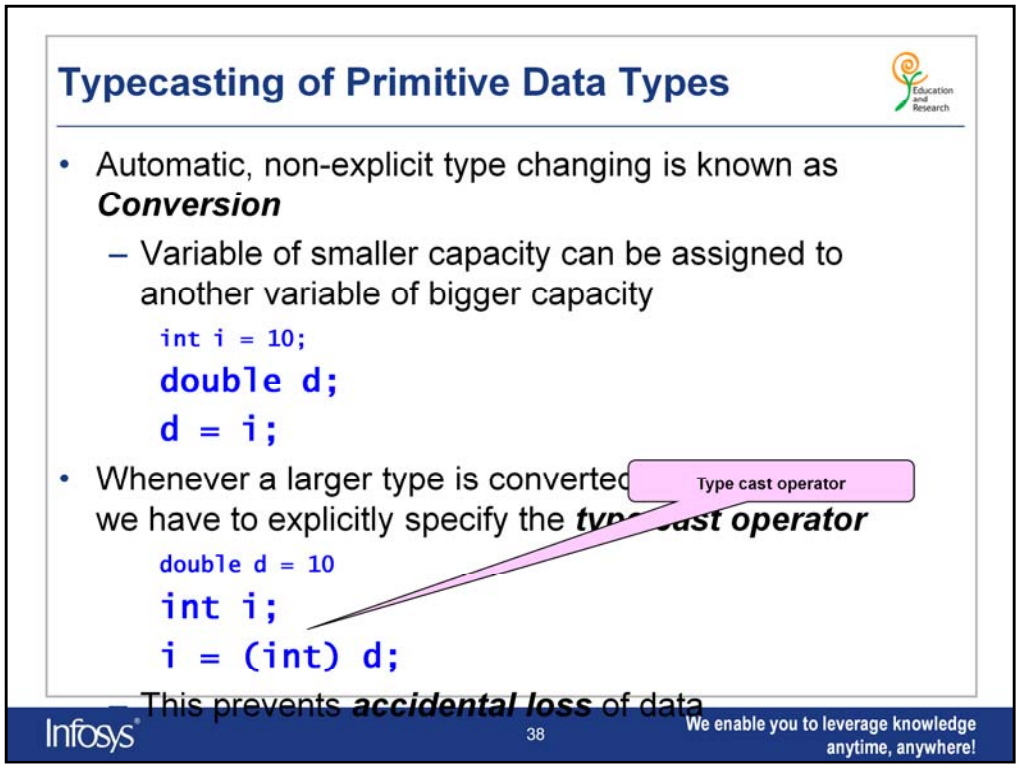
According to Java coding standards, the class name should always start with an upper case character. In case there is more than a word, each word should start with an uppercase letter and there should not be any _ in between the words. The name of a variable should be in lower case. In case there is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

Variables in Java : Assignment & Declaration

- Variables must have a type
  int count
- 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;

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



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. 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 -→|

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.

Widening and Narrowing(contd)

- **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

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

Infosys 41 We enable you to leverage knowledge anytime, anywhere!

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

Infosys 42 We enable you to leverage knowledge anytime, anywhere!

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

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

Infosys 44 We enable you to leverage knowledge anytime, anywhere!

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.

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

Infosys 46 We enable you to leverage knowledge anytime, anywhere!

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

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

When a member of a class having 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

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.

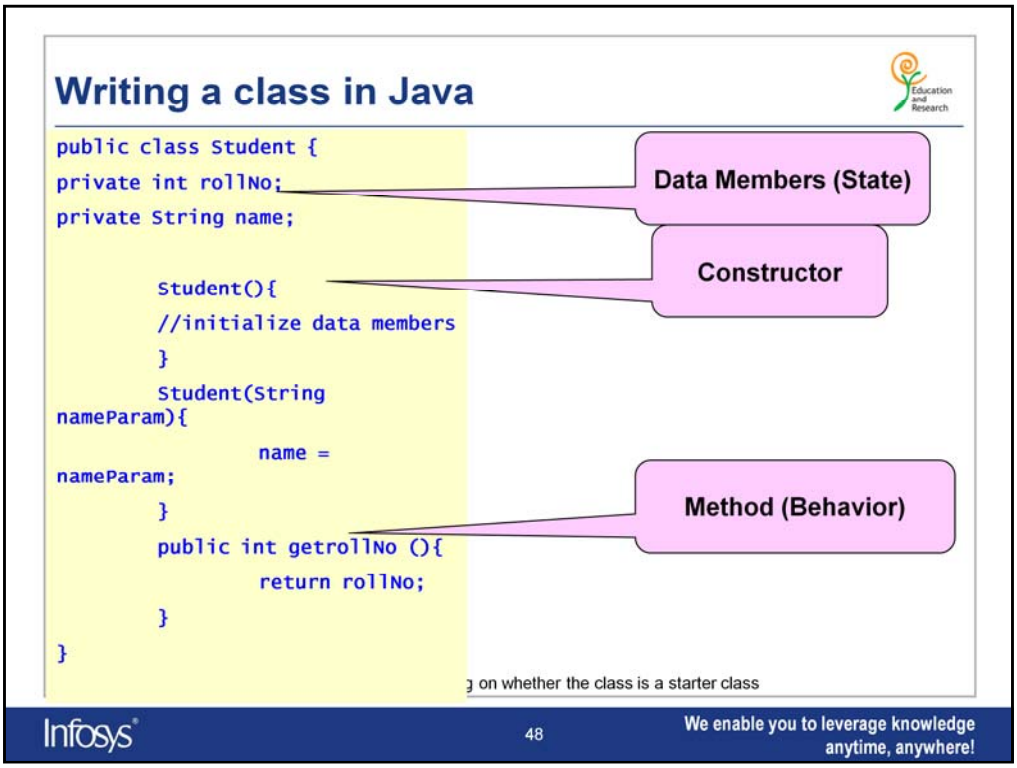
Static methods and variables can be used independently of any object via class names

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.



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.

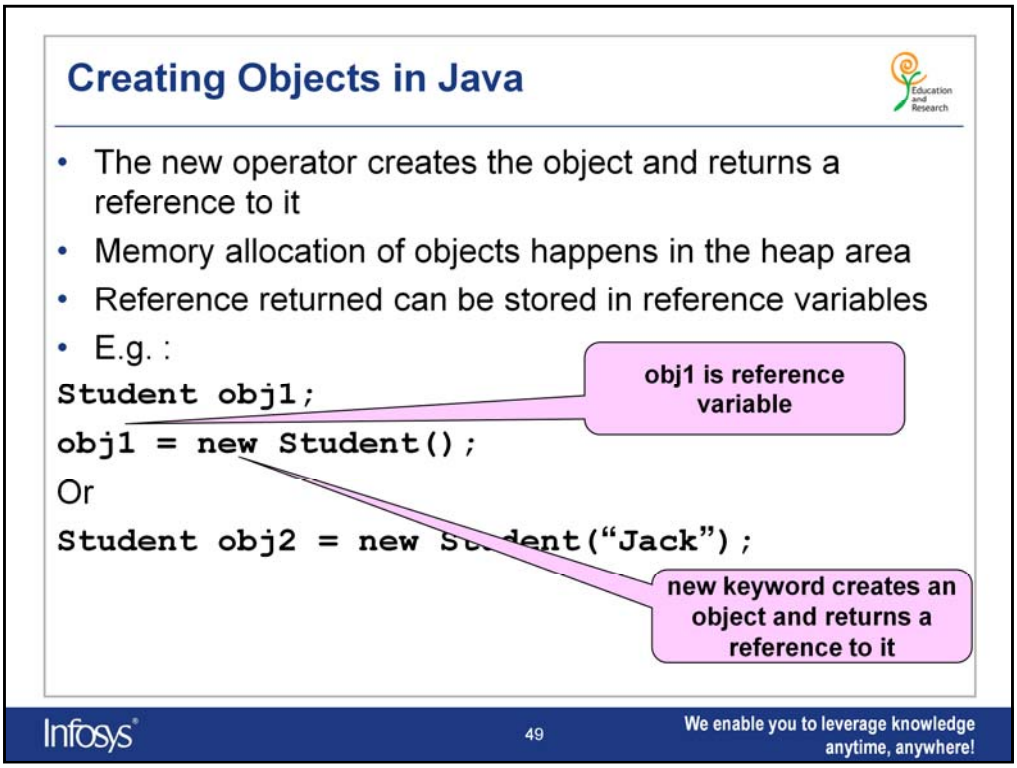
Where, student is the name of the class. Rollnumber and name forms the data members and getrollnumber() forms the method that access the datamember. A simple class definition will take the form of

```
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.



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

Infosys 50 We enable you to leverage knowledge anytime, anywhere!

A constructor is a special method that is used to do initialization 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 you write a constructor, remember that:

- it must the same name as the class
- it should 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)

- 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;
                              }
                              }
```


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.

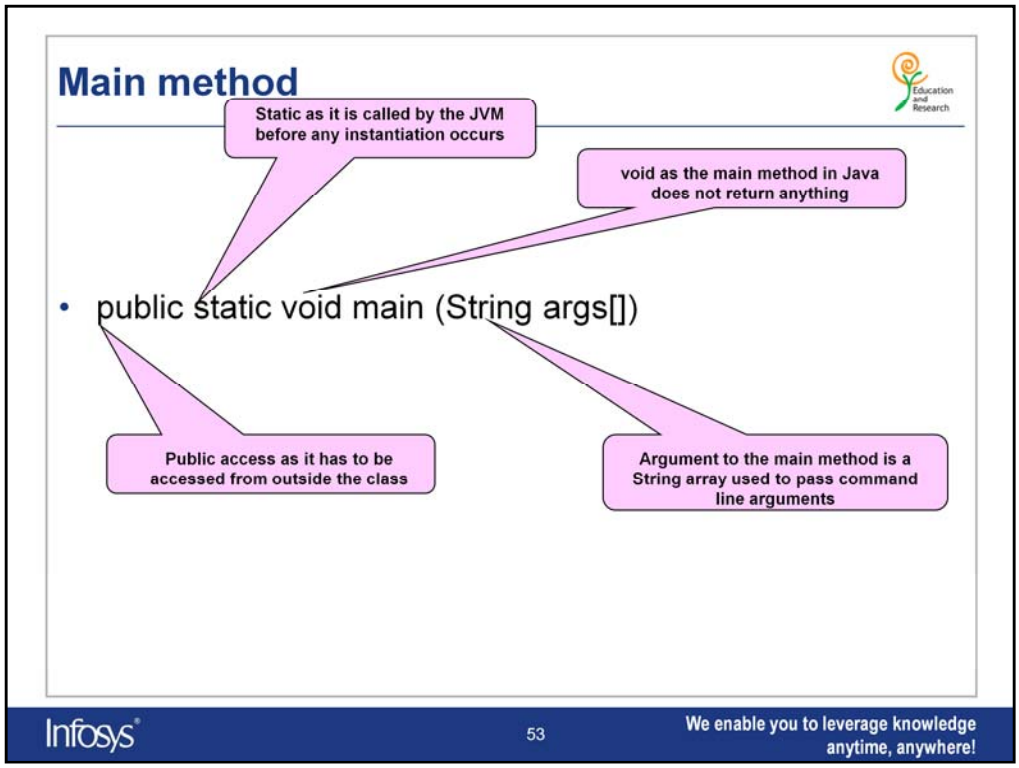
Here there are two constructors.

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 alons 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.

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.

Scope of Variables in Java

- **Two major scopes in Java – CLASS and METHOD**
- The variables which are 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

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 same type of data elements
- The size of an array is fixed and cannot increase to accommodate more elements
- In Java, array are objects and can be of primitive data types or reference types
- All elements in the array must be of the same data type

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;
```

Infosys 58 We enable you to leverage knowledge anytime, anywhere!

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];
```

Infosys 59 We enable you to leverage knowledge anytime, anywhere!

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 decalaration and construction of array in a single step.

Arrays[Contd…]

Initializing an Array

```
<elementType>[] <arayName> =
  {<arrayInitializerCode>};
```

Example:

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


Array initialization can be done as shown here.

Enhanced For loop(For Arrays)

- This can be used to simplify the looping through an array and collection.
- In basic for loop we have three components but here we have two.

Syntax:

```
for(declaration : expression)
```


The two components of the for loop are:

Declaration : A variable compatible with the array we are accessing.

Expression: An array variable or a method call that returns an array.Type of the array can be any.

Example

- int [ ] i ={1,2,3,4};
  for(int n : i)
  System.out.println(n);

  Which prints:
  1234

Variable Arguments

- Variable length argument is a new feature in J2SE 5.0.
- It allows programmer to create such method which can receive unspecified number of arguments .

- **Syntax:**
- public void method(int … var){ //Definition}

Infosys 63 We enable you to leverage knowledge anytime, anywhere!

Syntax of the var-agrs goes first of all data type you have ellipsis (...) in a method's parameter list and that indicates that the method is capable to receive a variable number of arguments of that specific type. According to syntax use of the ellipsis can occur only once in a parameter list, next rule is that the ellipsis, together with its type, must be placed at the very end of the parameter list.

Variable Argument Demo-1

```
class VariableArguementTest{
public void average( int … arg)
{
        double total = 0.0;

  // calculate total using the enhanced for statement
   for ( double d : arg )
   total += d;

  return total /arg.length;
}
}
```

Variable Arguments Demo-2

- The method in previous can be called by using any of the method invocation .

  double d1=10,d2=20,d3=30,d4=40;

  1.average(d1,d2);

  2.average(d1,d2,d3);

  3.average(d1,d2,d3,d4);

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;}
    ….}
```

Infosys 66 We enable you to leverage knowledge anytime, anywhere!

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

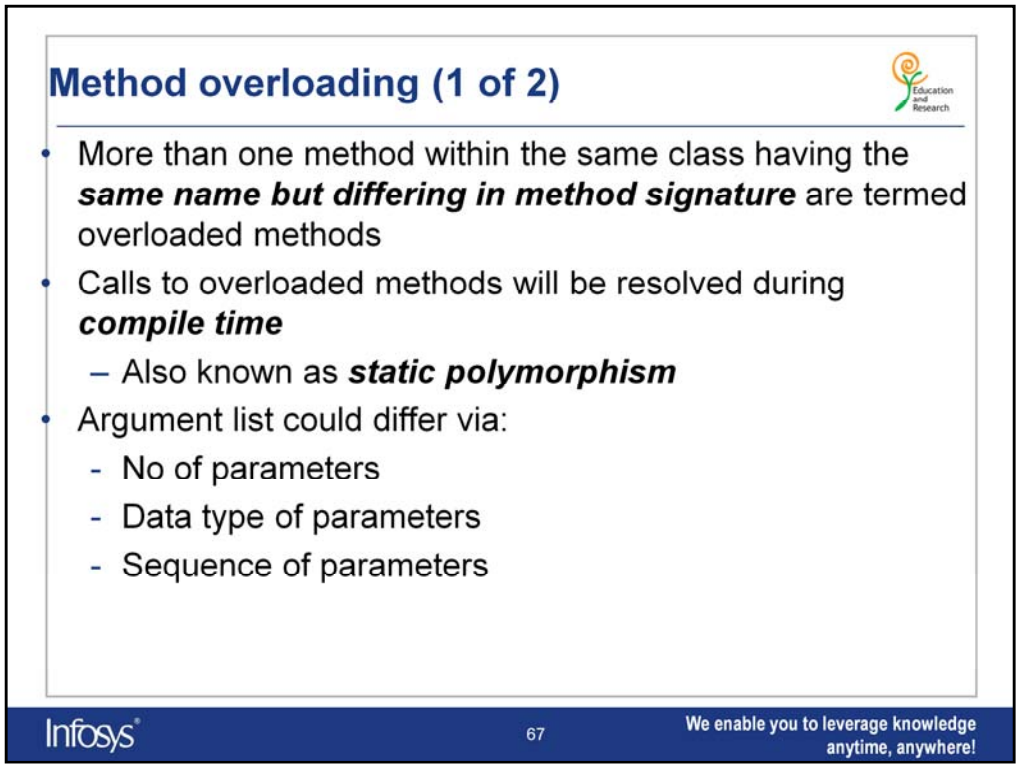
When a local variable has the same name as an instance variable, the local variable hides the instance variable.

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;



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.

Method overloading (1 of 2)

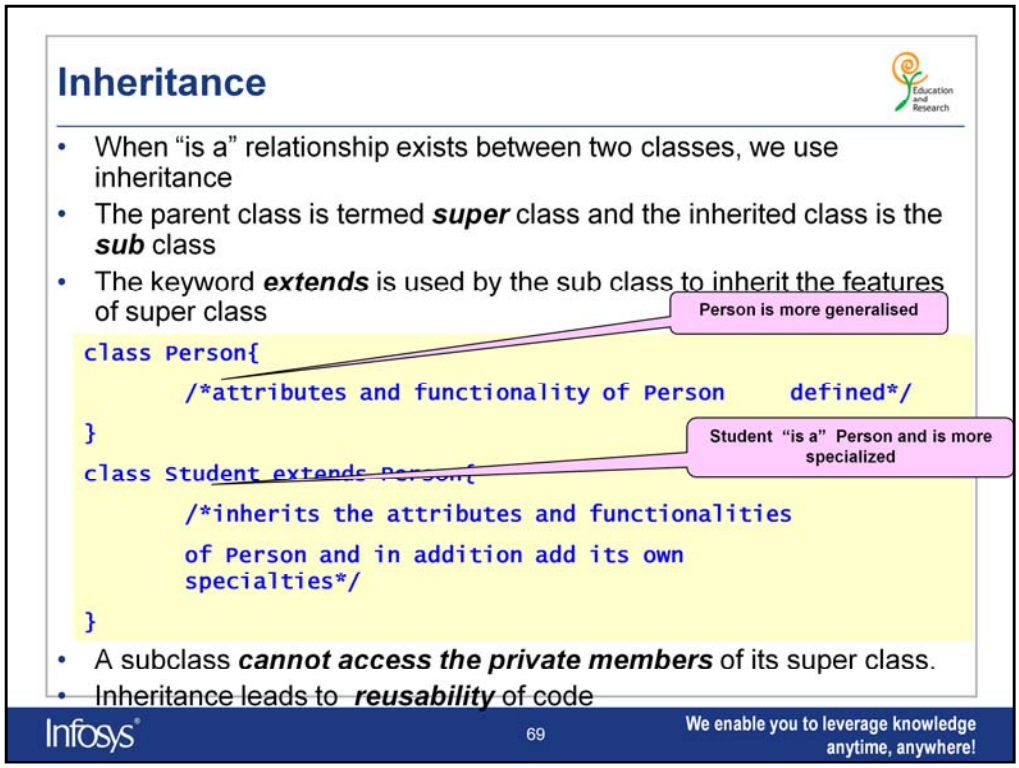
- Ex.

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

Overloaded methods

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

Not overloading. Compiler error.



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

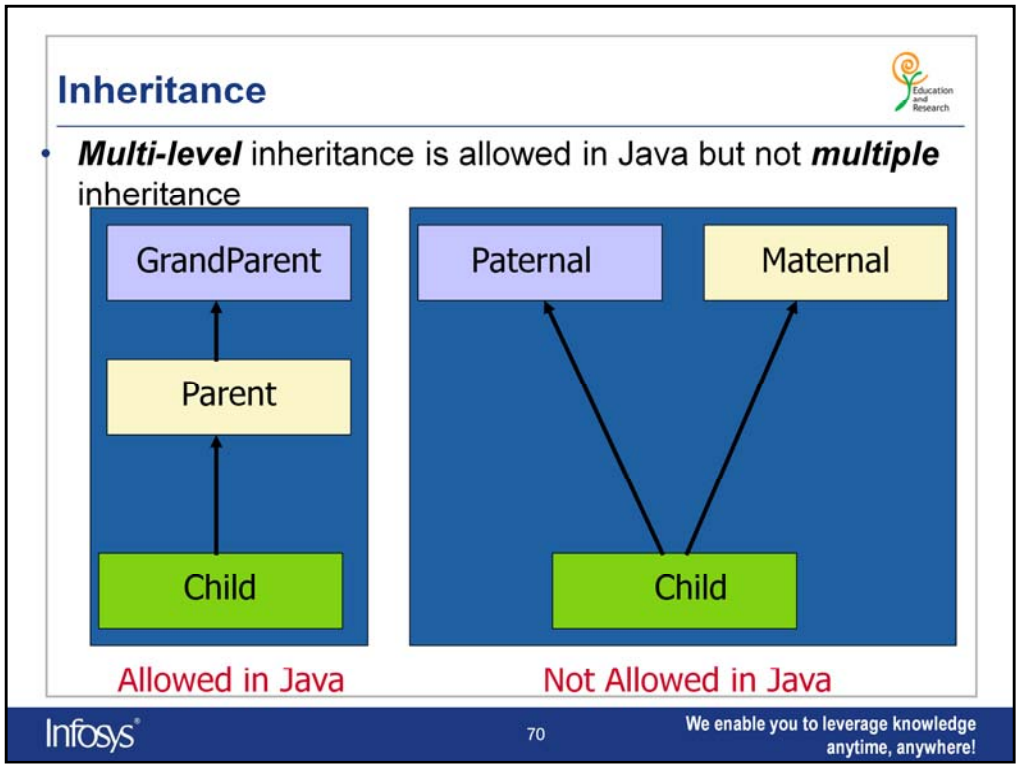
A sub class is the specialized version of a super class.

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

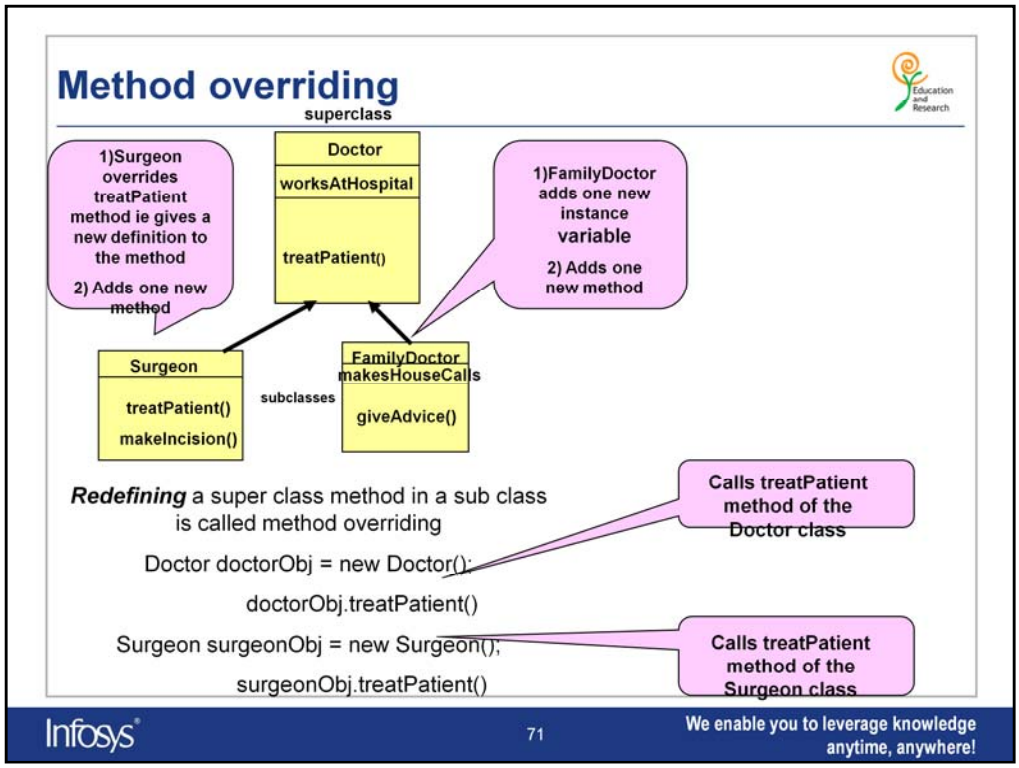
Although a sub class includes all of the members of its super class it can not access those members of the super class that have been declared as private.

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

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



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***

Infosys 72 We enable you to leverage knowledge anytime, anywhere!

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?

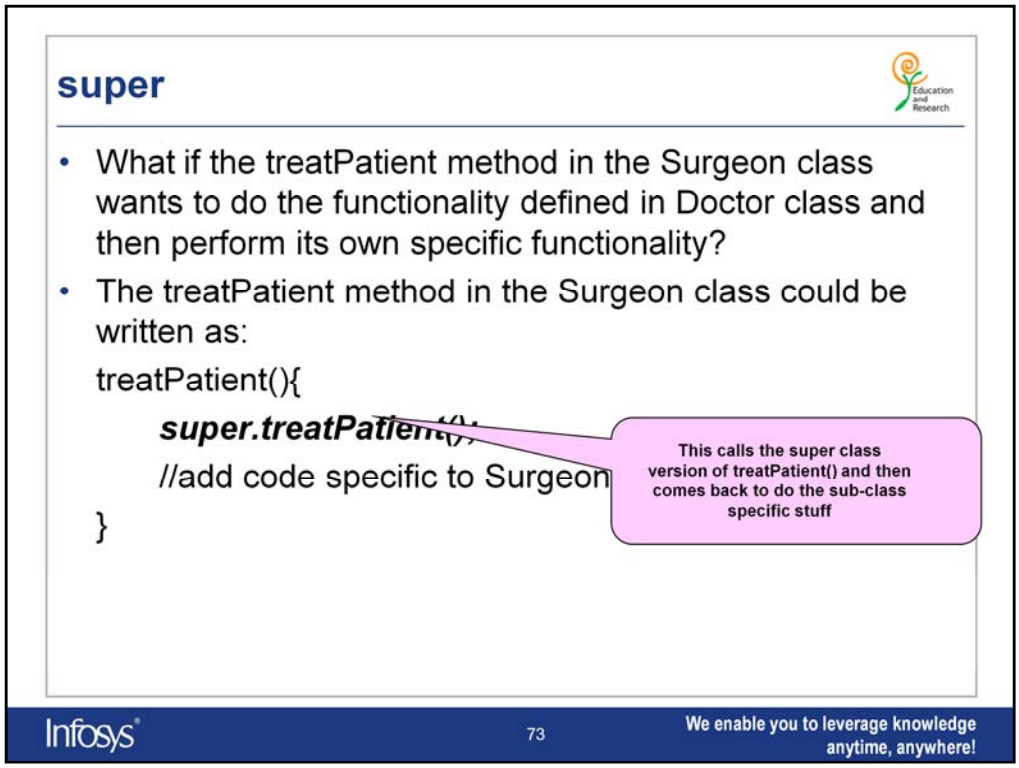
It calls Surgeon's version of the treatPatient method as the reference is pointing to a Surgeon object

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 overridding as an example for runtime polymorphism and method overloading as an example for compile time polymorphism.



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

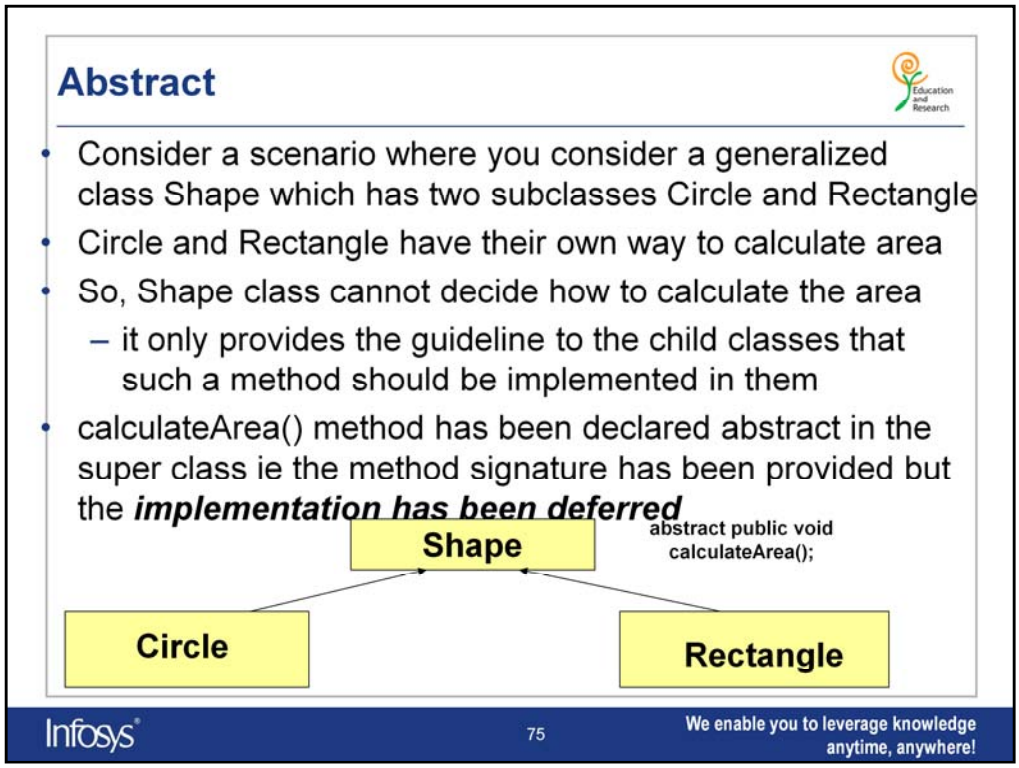
Here member can be either a method or an instance variable.

Polymorphism

STATIC	DYNAMIC
• Function Overloading – within same class more than one method having same name but differing in signature	• Function Overriding – keeping the signature and return type same, method in the base class is redefined in the derived class
• Resolved during *compilation time*	• Resolved during *run time*
• Return type is not part of method signature	• Which method to be invoked is decided by the object that the reference points to and not by the type of the reference

Infosys 74 We enable you to leverage knowledge anytime, anywhere!

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



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

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, 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

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

Infosys 78 We enable you to leverage knowledge anytime, anywhere!

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:

•Class contains one or more abstract methods.

•Using inheritance the class inherits abstract methods (from an abstract parent) for which it does not provide implementations.

•If the class does not provides implementation to the interface methods during interface implements.

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**

Infosys 79 We enable you to leverage knowledge anytime, anywhere!

The final modifier applies to classes, methods, and variables.

•A final declared class may not be sub classed.

•If final is 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 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

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

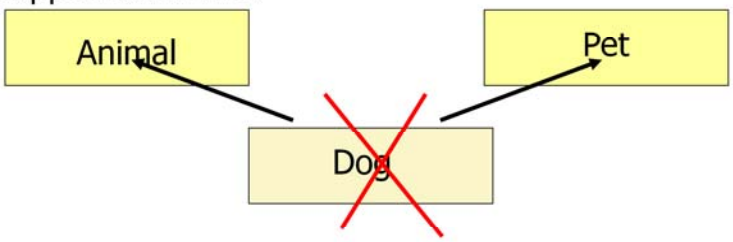
So, should we place the functionalities of a Pet in class Dog?

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



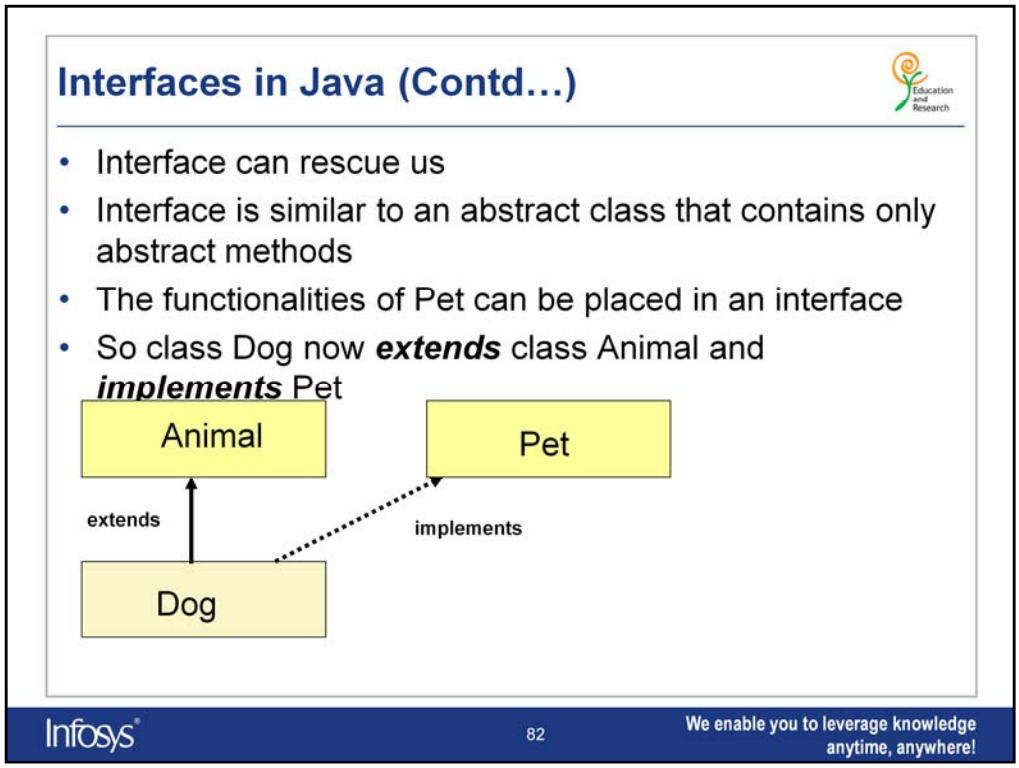
So what is the solution?

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.



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
}
```

Infosys 83 We enable you to leverage knowledge anytime, anywhere!

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
```


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

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

- Prefer interface inheritance to implementation inheritance (abstract class)because it promotes the design concept of **coding to an interface and reduces coupling.**

Inner Classes

- A class which is defined within another class is known as Inner class.
- Inner class should be non-static nested class. It has access to all of the variables and methods of its outer class.
- Scope of inner class is limited to the scope of its enclosing class.
- **ANONYMOUS INNER CLASSES** : An anonymous inner class is a class without any name assigned to it.

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();
  } }
```


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

```
class AnonymousTest
{

    public static void main(String arg[])
                    {

 Thread thread=new Thread  (
 new Runnable       {

 public void run(System.out.println("Testing Anonymous Inner
                    ");
                )
                    }       );

                    }

}
```


Here is an example for Anonymous inner classes.

Java I/O Basics

- Java programs perform I/O through streams.
- A stream is an logical path that either produces or consumes information
- Two types of streams:
  - Byte Stream
  - Character Stream
  - ❖ In Byte stream at the high level there are two abstract classes : InputStream and OutputStream
  - ❖ In Character Stream there 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

Infosys 90 We enable you to leverage knowledge anytime, anywhere!

Java programs perform I/O through streams.

A stream is an logical path that either produces or consumes information

Two types of streams:

Byte Stream

Character Stream

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

In Character Stream at the highe level there 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 which are – 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.

Use of Scanner Class

- Scanner class of java.util package allows user to read various types of data.
- Below are particularly two constructor of this class which are most useful

- **1.**Scanner scanner = new Scanner (System.in)

  Here System.in is an InputStream

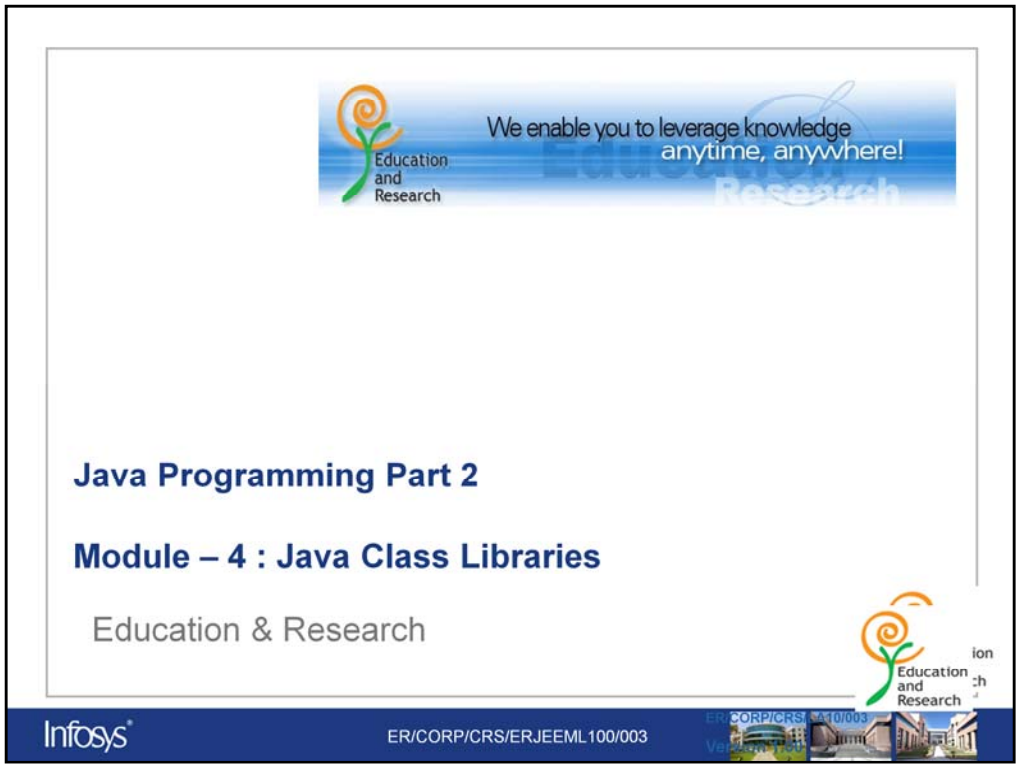
  2.Scanner scanner = new Scanner(new FileReader("file"))

  We will talk more about this while dealing java.io in Java Programming Part 2.

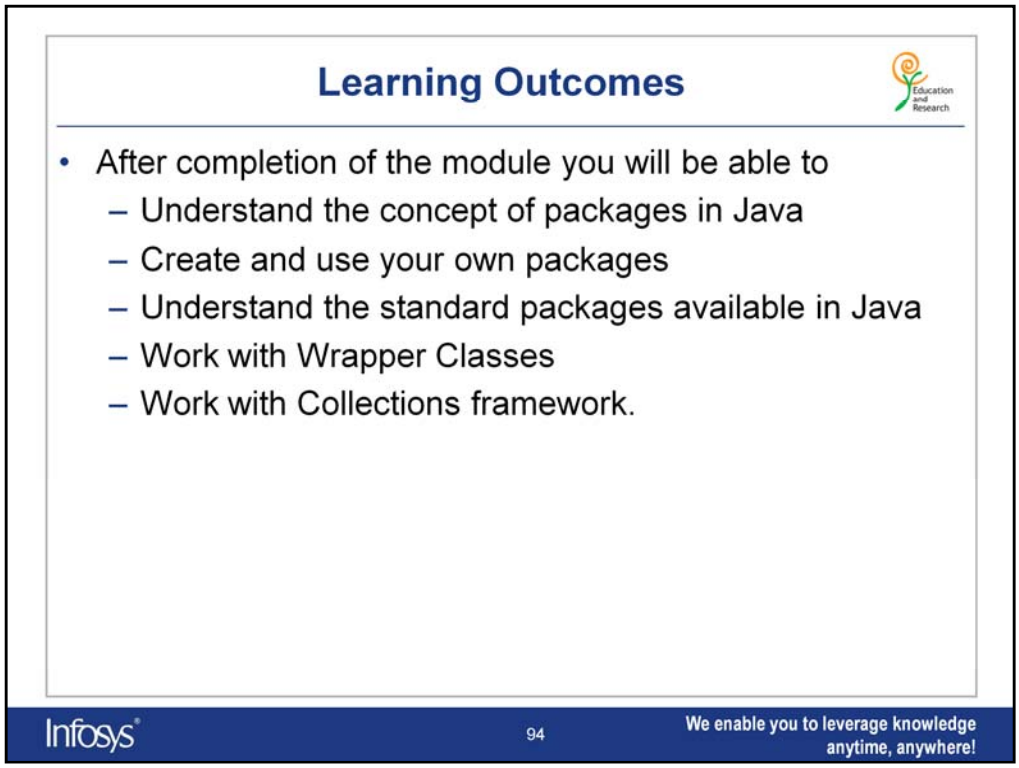
Scanner class looks for the tokens in the input which are seperated by whitespaces .Whitespaces can be a blank space,a carriage return or a tabline character.

Scanner class Demo

```
class ScannerDemo{
public static void main(String str[])
{
Scanner scanner =new Scanner (System.in);
Sytem.out.println("Enter you username:")
String userName =scanner.nextLine();
if (userName.equals("admin"))
        { System.out.println("Welcome to Java World");}
else{ System.out.println("Authentication check Fail")}
        }
                                        }
```



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



After completion of the module you will be able to

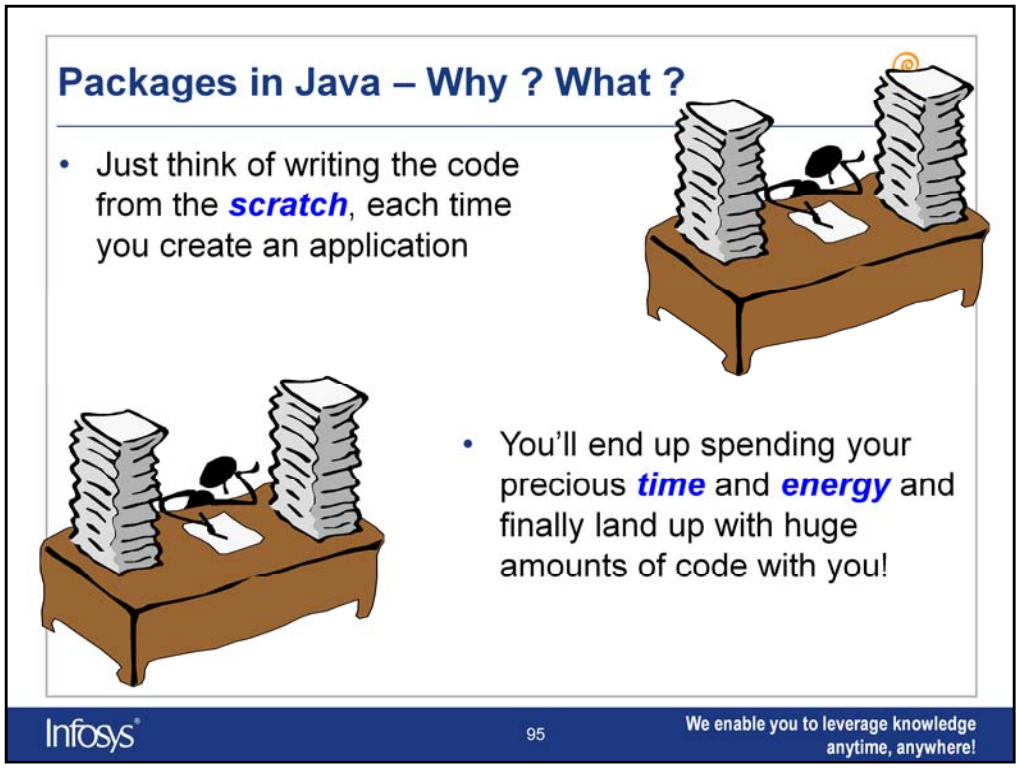
Understand the concept of packages in Java

Create and use your own packages

Understand the standard packages available in Java

Work with Wrapper Classes

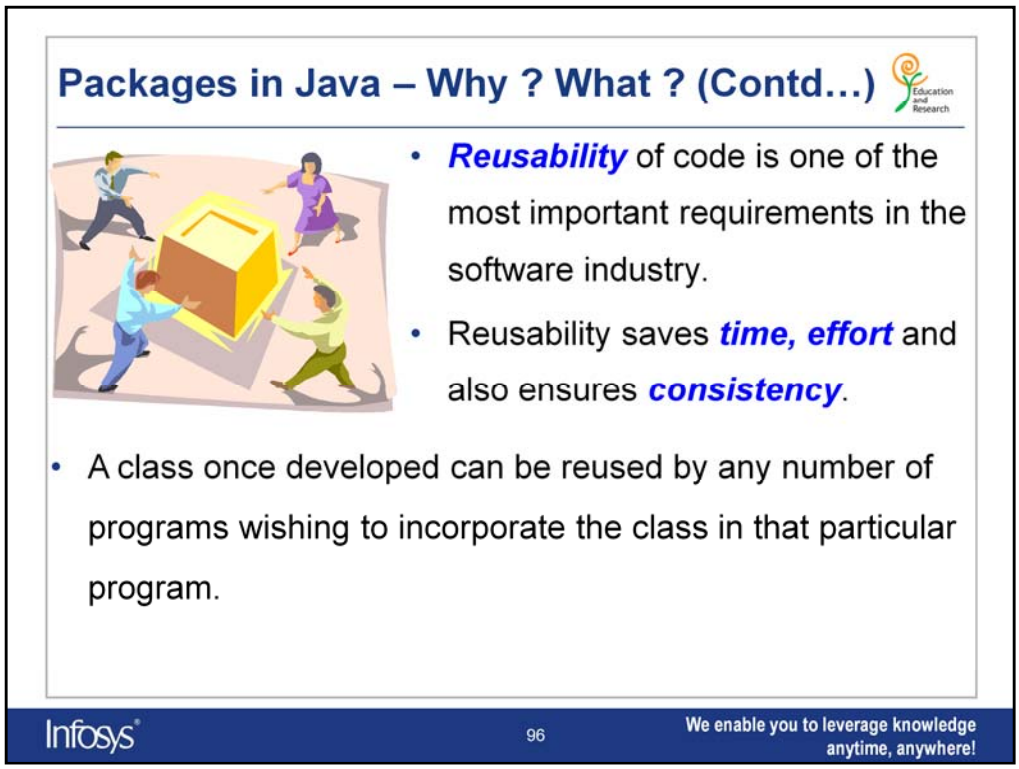
Work with Collections framework.



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!



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***.

Infosys 97 We enable you to leverage knowledge anytime, anywhere!

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.

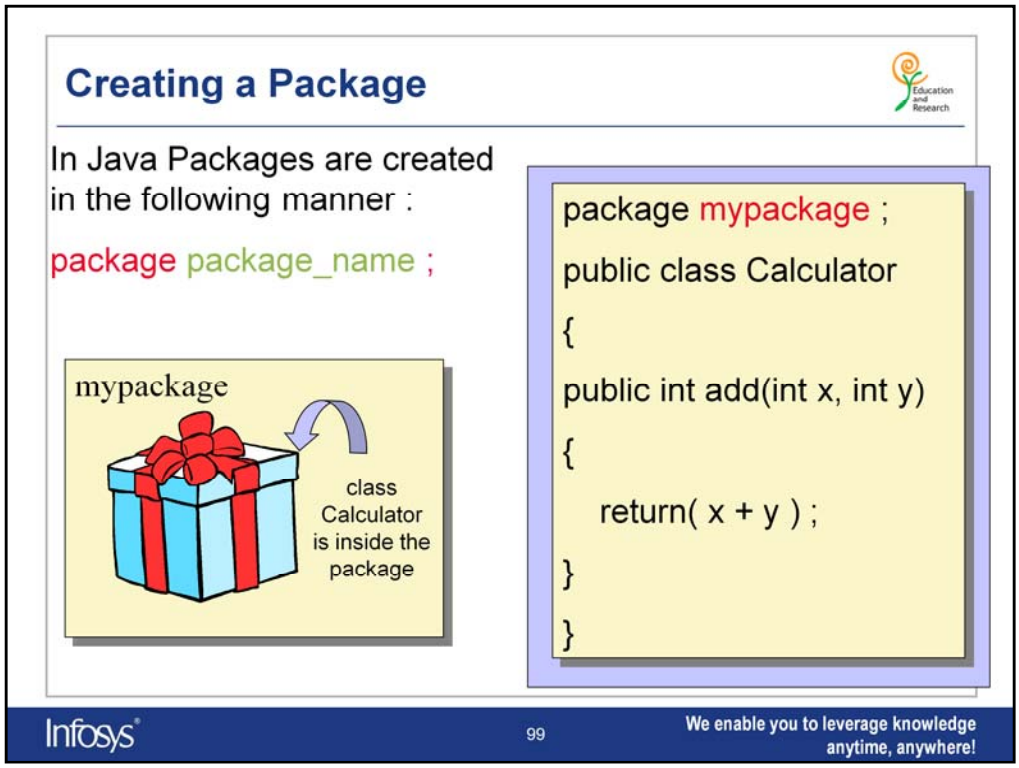
A java program can contain any of the following four parts.

- 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

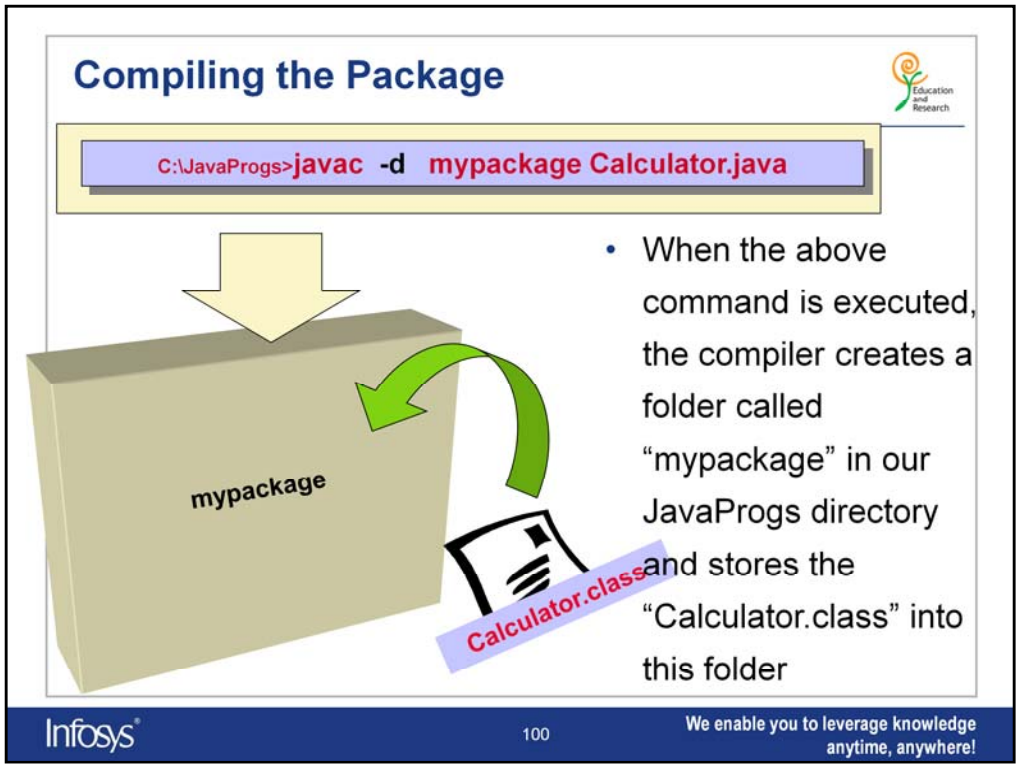
- ***Organize*** your classes into smaller sub units and make it easy to locate and use the appropriate class file.
- Avoid ***namespace conflicts***
  - Package names can be used to ***identify*** your classes
- Packages helps you to ***protect*** your classes, data and methods in a larger way than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



Creating a Package

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

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



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 full qualified name becomes difficult
- 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 :

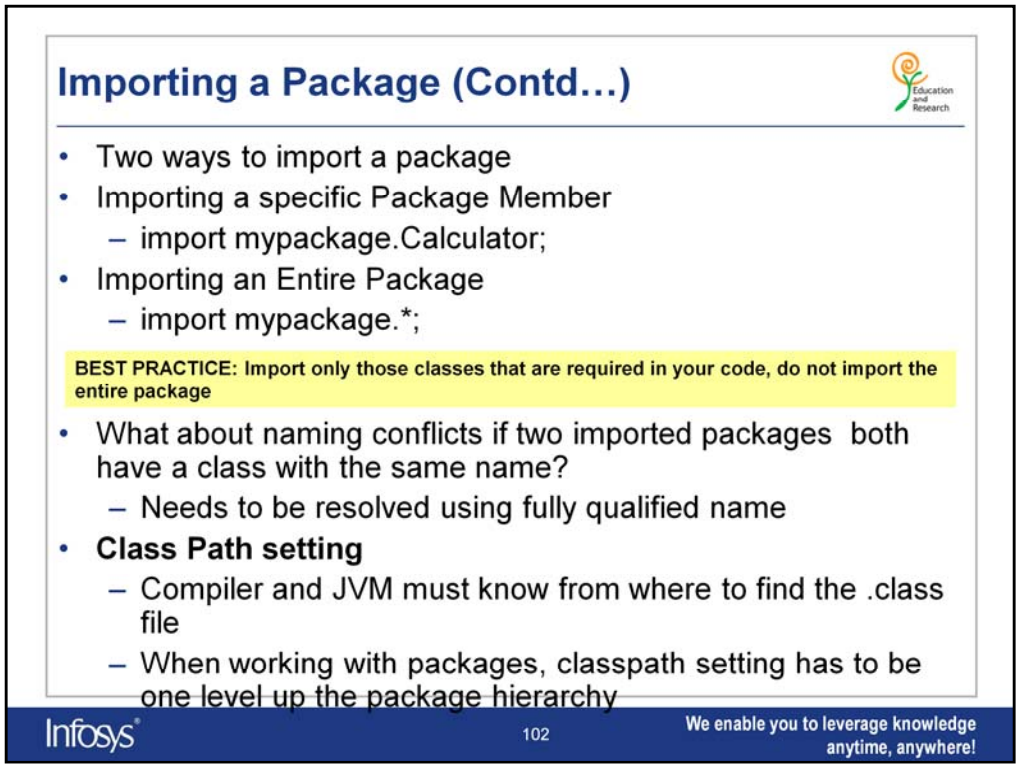
  - `import mypackage.mysubpackage.MyClass;`

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.



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 (*) which is 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 packages entirely by default:

They are 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(.)

The specific location that java compiler will consider, as the **root of any package** hierarchy is, controlled by **Classpath**

When working with packages, the classpath setting needs to be done till one level up the package

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

```
package mypackage ;
import java.io;
```

ILLEGAL

```
import java.io;
package mypackage ;
```


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;

import java.awt.*;
import java.uti.*;

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

Package statement should be the first statement. It is optional

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

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

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

Infosys 105 We enable you to leverage knowledge anytime, anywhere!

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

Infosys 106 We enable you to leverage knowledge anytime, anywhere!

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 defalut protected and then public.

Access Specifiers (3 of 3)

Keyword	Applicable To	Who can Access
private	Data members and methods	All members within the same Class only
(No keyword, usually we call it default)	Data members, methods, classes and interfaces	All classes in the same package
protected	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
public	Data members, methods, classes and interfaces	Any class

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.

Static import

- Static import is being used to access the static member of other class without writing full qualified path.

- E.g.-

```
import static java.lang.System.out;
class StaticImportTest{
public static void main(String arg[])
{
out.print("Testing Static Import")        ;
}
}
```

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.

Infosys 109 We enable you to leverage knowledge anytime, anywhere!

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.

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.

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

This is the Most widely used Package

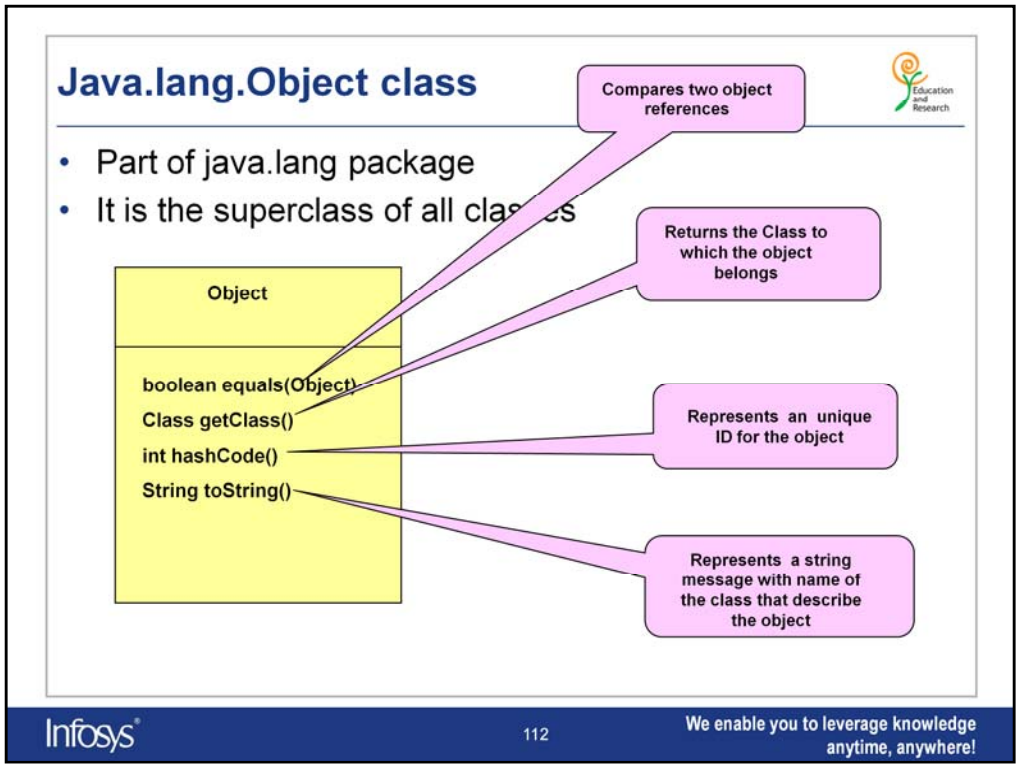
To convert numbers from strings : parseByte(), parseShort(), parseInt() and parseLong() are used.

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

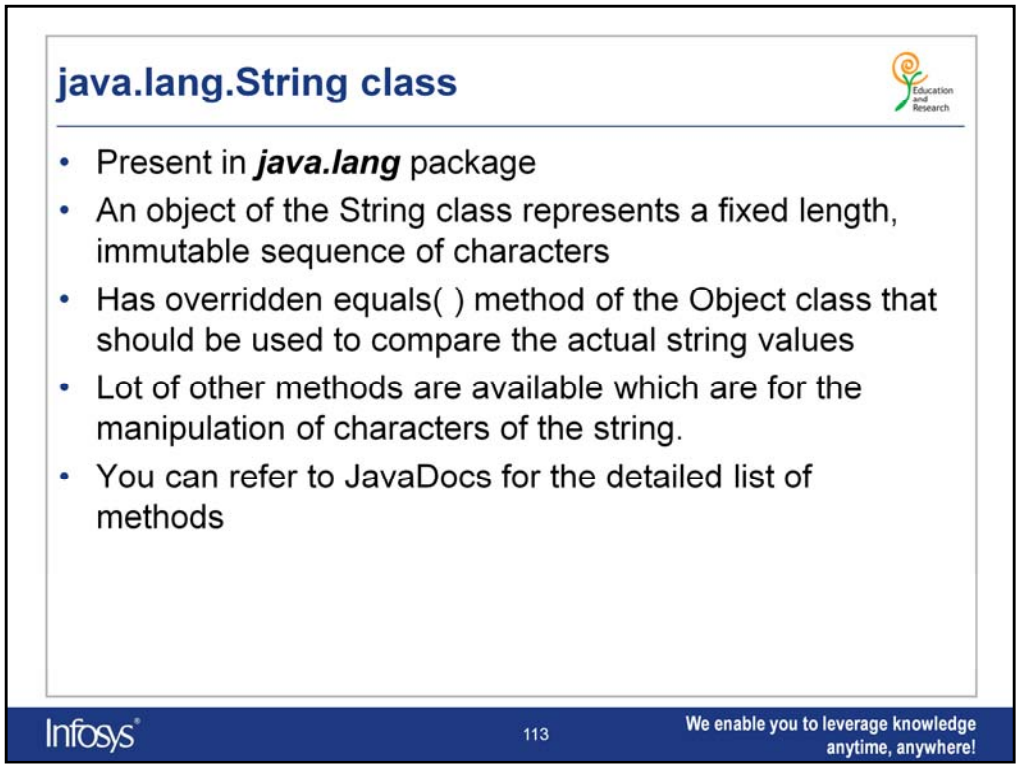
System class holds a collection of static methods and variables. The standard input, output and error output of Java run time are in the in, out, err variables.

Throwable class supports Java Exception Handling

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



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 sub class, directly or indirectly , of the Object class.



Java.lang.String

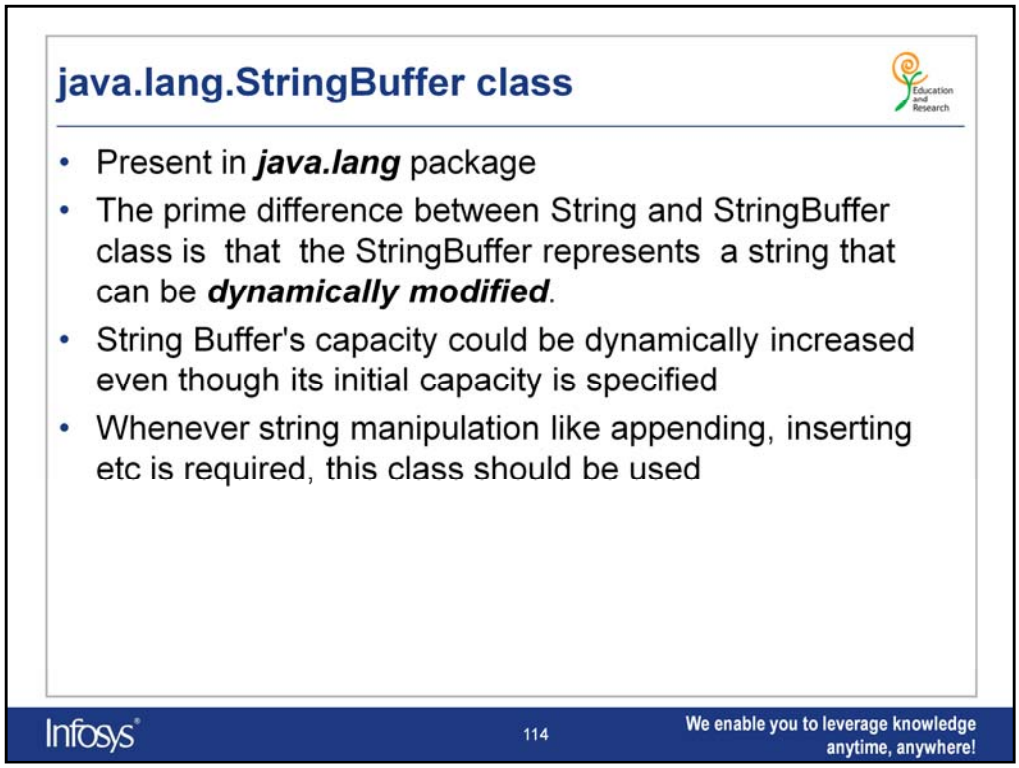
Present in ***java.lang*** package

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

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

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

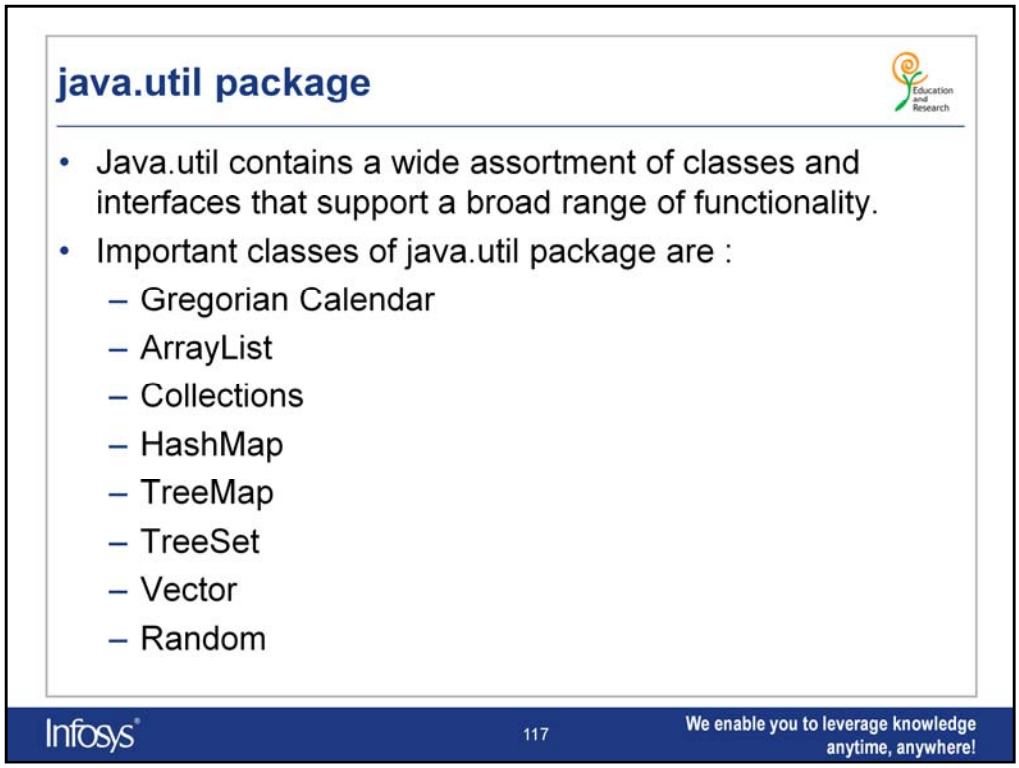
Here is an example for Wrapper Classes.

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

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 :

- Gregorian Calendar
- 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

Infosys 118 We enable you to leverage knowledge anytime, anywhere!

The Java 2 platform includes a new *collections framework*. A *collection* is an object that represents a group of objects 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

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

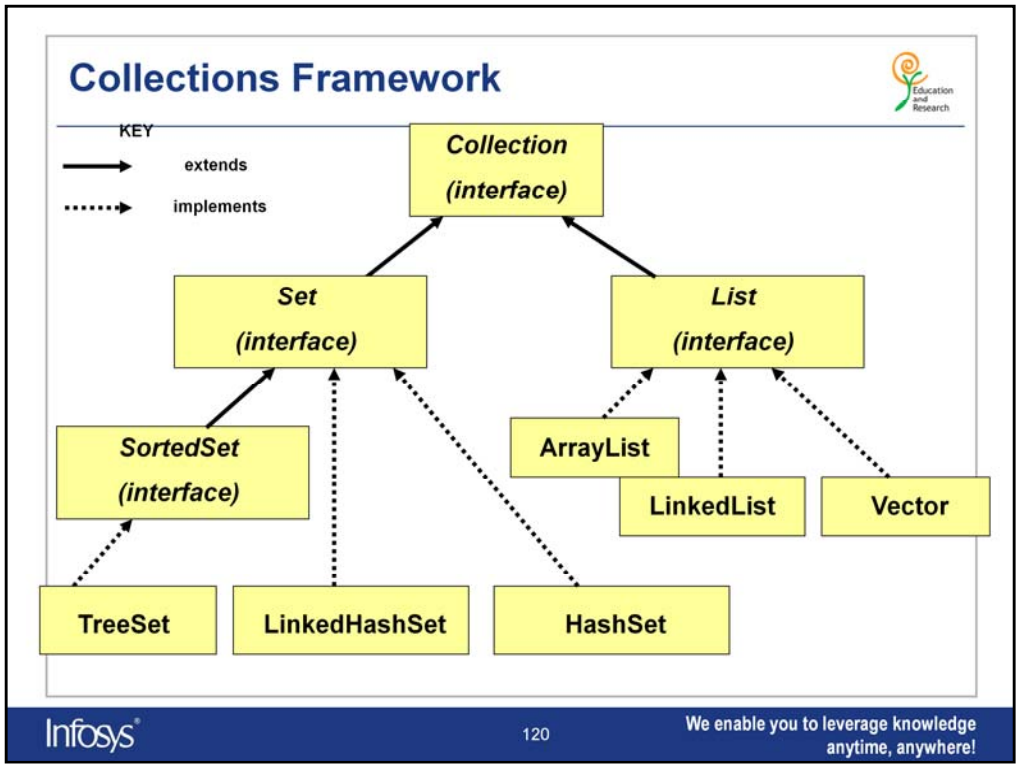
All collection classes in the Java API implement any one of the three interfaces

List, Set, Map

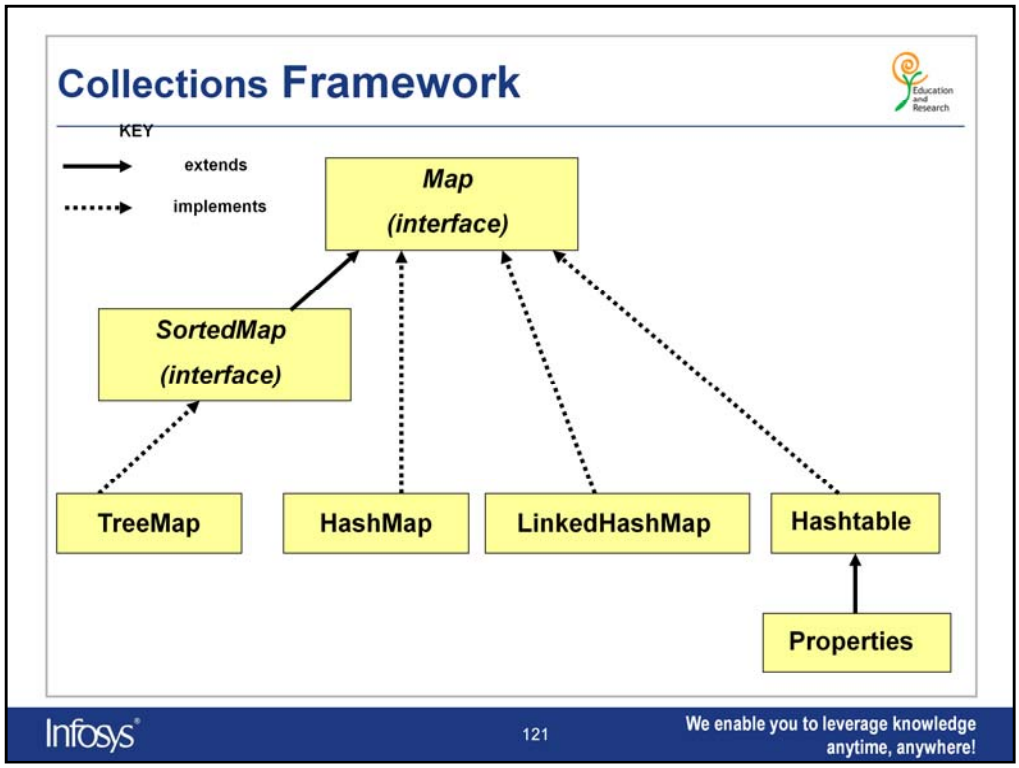
List for A collection of objects ***where the element present at a particular index*** is known

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

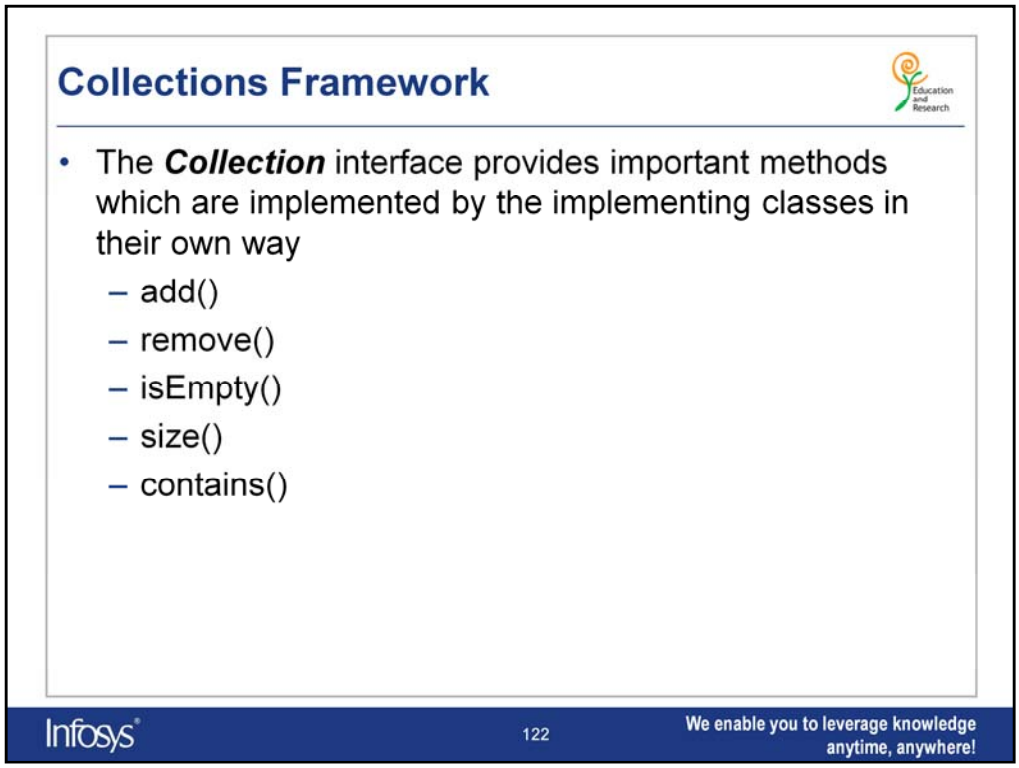
Map for A collection that provides a ***key-value*** pair manner



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.



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()

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()***

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

StringTokenizer class is an implementation of Enumeration interface

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

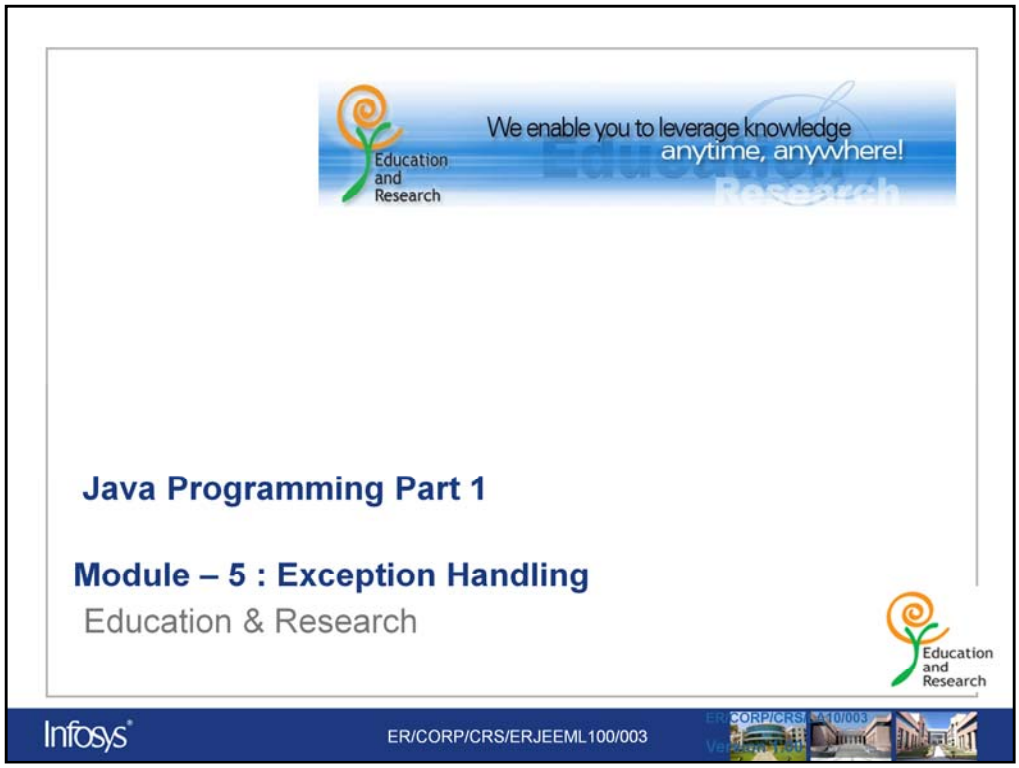
It has methods,

> ***hasMoreTokens() which*** returns a boolean, based on whether more tokens are to be returned
>
> ***nextToken()*** which returns the next token

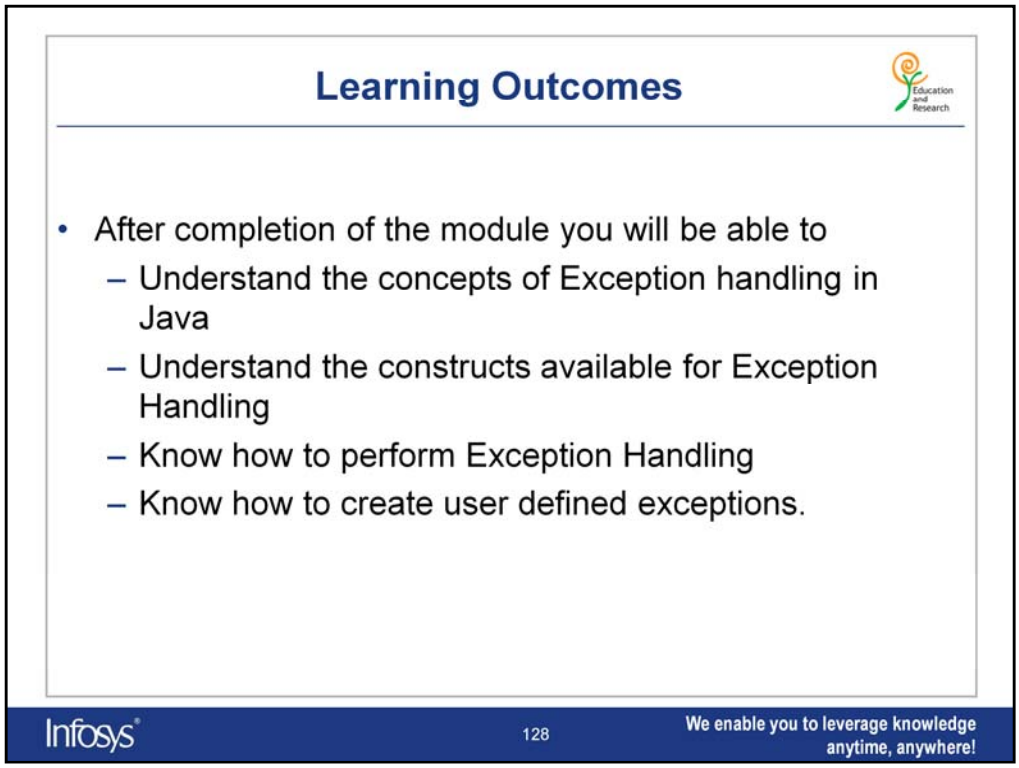
Vectors

```
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);
  }}
```


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



We shall now move on to the module on Exception Handling



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);
   }
}
```

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

Infosys 129 We enable you to leverage knowledge anytime, anywhere!

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 exception handler,which is 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 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

The Java programming language gives a mechanism known as exceptions which is helpful to get programs report and useful to handle errors. When an error occurs, the sourse program throws an exception. It shows that the normal flow of the program is interrupted and that the runtime environment looks for an *exception handler that is* a block of code that can handle that specific type of error. Either exception handler can handle the exception or if the exception is irrecoverable then it provides 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.

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

for example the attempt to divide by zero. This exception is created 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
ArithmeticException	is	It is a subclass of Exception, which describes the type of the error happened.

Manually generated exceptions are typically used to report some error condition to the caller of a method.

Exception Handling in Java (Contd…)

- 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

Exception Handling in Java (Contd…)

- Java exception handling is based on five keywords **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.

Infosys 133 We enable you to leverage knowledge anytime, anywhere!

Java exception handling is is based on five keywords **try, catch, throw, throws,** and **finally.**

Program statements that you suspect about 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 is a block of statements within which an exception may occur.
- The *catch* statement works in combination with try block and identifies a block of statements that can handle a specific type of exception. The statements written under catch block 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 irrespective to an error occurs within the try block.

The *try* statement is a block of statements within which an exception may occur.

The *catch* statement works in combination with try block and identifies a block of statements that can handle a specific type of exception. The statements written under catch block 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 irrespective to 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)
}
```


Here's the general form of these statements

Exception Handling in Java (Contd…)

Program 1

```
class Hello
 {
   public static void
  main (String args[])
   {
     /* Now let's say
  hello */

  System.out.print("Hell
  o");

  System.out.println(arg
  s[0]);
   }
```

Program 2

```
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
 whenever you are");
```

Infosys

We enable you to leverage knowledge anytime, anywhere!

Consider Program 1

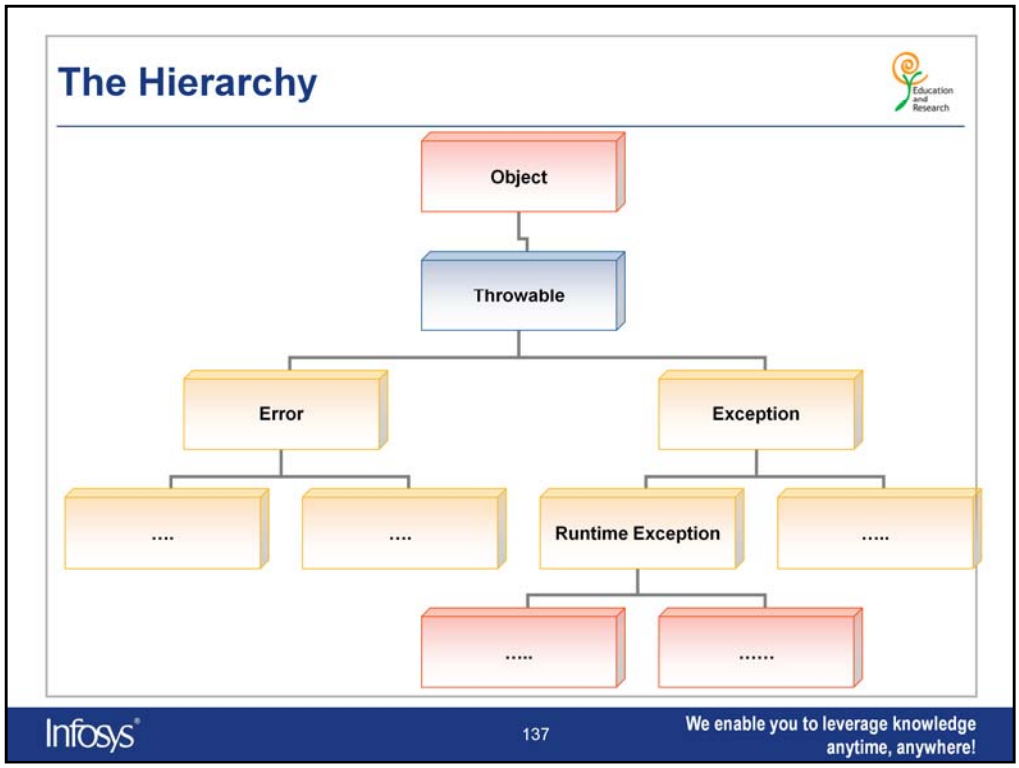
what will happen when you run the program without giving it any command line arguments? The runtime system generates an exception, Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException at Hello.main (at Line Number 7)

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".

We can fix this problem by checking the size of array and value send to it to rectify the problem but how many time we need to deal with such errors.

The job of exception handling is to provide smooth flow of execution and if there are any exception that should be dealt in separate block of code.

To handle it properly you can use try block to implement your logic where exception may occur and catch to execute logic in case exception occurs.



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 a 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

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.

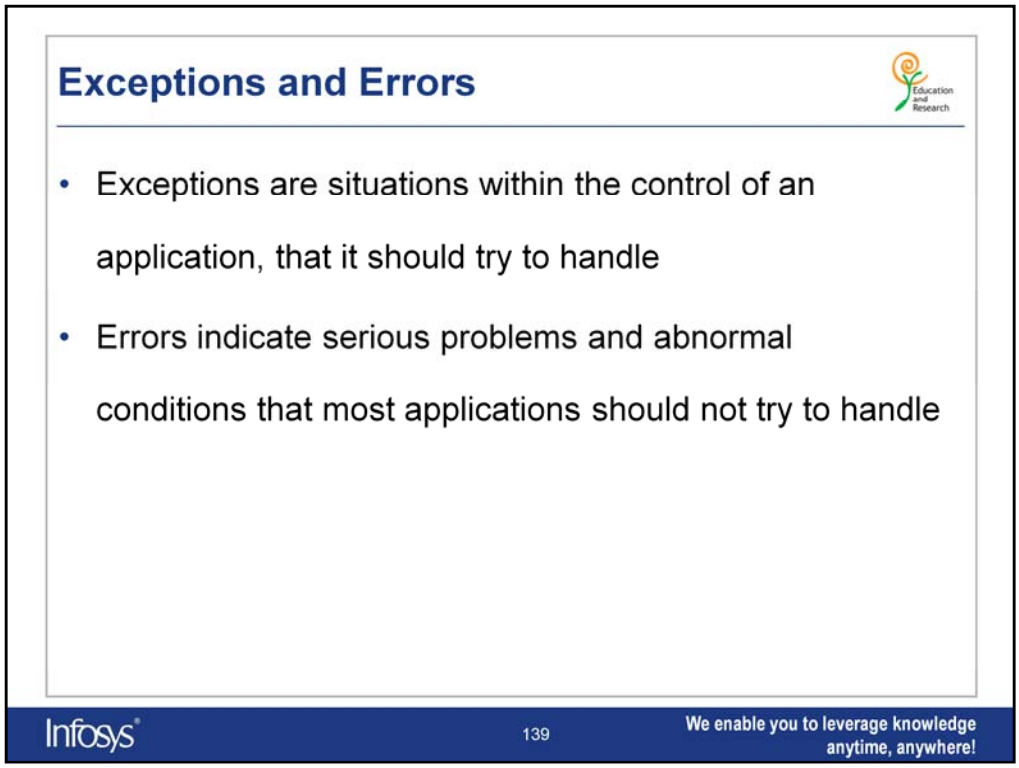
Examples of Checked Exceptions :

1 ClassNotFoundException
2 IllegalAccessException
3 NoSuchFieldException
4 INterruptedException etc.

Unchecked Exceptions : compiler does not check to see if a method handles or throws these exceptions

Examples of UnChecked Eceptions

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ClassCAstException
- NegativeArraySizeException etc.



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

Error occurs when a dynamic linking failure or some other "hard" failure in the virtual machine occurs. Typical Java programs are not supposed to catch these Errors.

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

•IllegalStateExecption

•IndexOutOfBoundsException

•InstantiationException

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, there should be a catch clause that specifies the exception type that is to be caught.
- A try block is always followed by a catch block

Try – Catch – Finally- (Contd..)

Multiple Exceptions

- If there are multiple exception exist in a classes in the **try** block, then several **catch** blocks are allowed to handle them separately, each handling different exception type.

If there are multiple exception possible in a class that might be in the **try** block, then several **catch** blocks are allowed to handle them separately, each handling different exception type.

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"); }}
```

Infosys 142 We enable you to leverage knowledge anytime, anywhere!

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.

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");
   }  } }
```


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

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 situations 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.

The situations due to which **prevention of execution** of the code in a finally block may occur 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("ArithmeticExceptio
  n:"+e);
   }
   finally{
        System.out.println("inside
```

Infosys 14.7 We enable you to leverage knowledge anytime, anywhere!

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

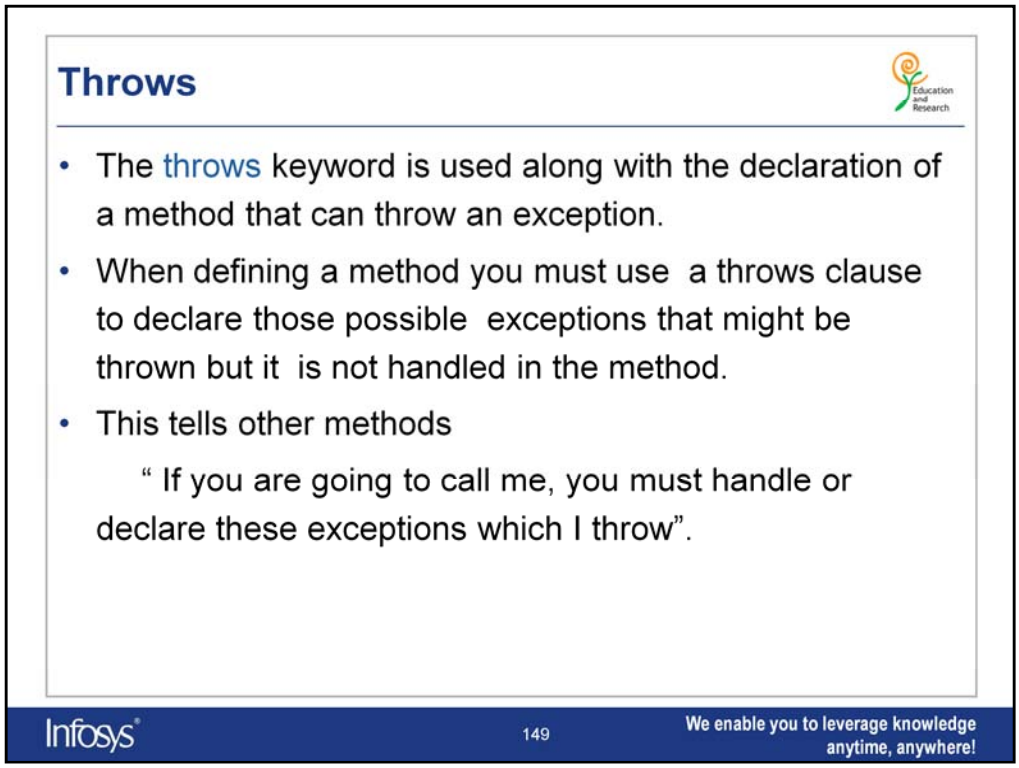
```
public void read() throws IOException{
            // Some code that causes IO Exception
            throw new IOException();
  }
```

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

Infosys 148 We enable you to leverage knowledge anytime, anywhere!

Throw statement is used explicitly to throw an exception.

Its syntax is **throw ThrowableInstance;**



Throws

If any method is capable in throwing an exception but it not handling it then that method has to declare it .These are basically checked exception to whom you have to handle or declare. 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
- Create an instance of the user-defined exception class.
- Use the throw keyword to throw the instance created.

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");
}
}
```


Try the code and analyse the results.

We shall now move on to the module on Swings.

Learning Outcomes

- After completion of the module you will be able to
  - Understand the Java Runtime Execution and Memory allocation to different part of your program.

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

Java Runtime Environment

- Overview
- Execution Environment
- Bytecode verifier
- JVM Heap
- Garbage Collector

Overview

- Java Virtual Machine is a piece of software and data structures which operates on Virtual Machine model.
- It operates on Java Bytecode which is being generated using Java Source code.
- JVM is important component of Java platform and it is available for many hardware and software plateforms
- The Bytecode generated can be used across the plateform that can be describe as “Compile once Run anywhere”.

Execution Environment

- The programs intended to be executed must be in the correct binary format that comes in the form of .class file.
- To distribute large application the class file are bundled together in the .jar file.
- JVM executes this .class or .jar file by interpreting it or compiling it by using JIT compiler like Sun's HotSpot.
- JIT compilers are being used to speed up the execution of the programs.
- JVM has its own stack base architecture like we find in microprocessors or microcontrollers.
- JVM the instance of Java Runtime Environment.
- It comes in action when a Java Program is executed.

Bytecode Verifier

- JVM's one of the responsibility is to verify the bytecode before it is executed.
- Mainly it perform this task in three steps:
- 1.Branches are always to valid location.
- 2:Checking for type safety and initialization
- 3.Access rights are rigidly controlled.

The first two steps occurs primarily when the class is loaded and ready for use and third step occurs at the time when program is running and one classes data and methods try to access others.

JVM Heap

- JVM uses Heap which is a memory area for the dynamic memory allocation.
- The Heap is divided in three different segment:

  1.Young Generation

  2. Old Generation or Tenured Generation.

  3:Permanent Generation

Young Generation: It stores the objects which are short lived and Garbage collected immediately.

Old Generation : It stores the objects that persist longer .It is also known as Tenured generation.

Permanent Generation: It is used to store class definition and data associated with it.

Initially there was no concept of Permanent generation and Objects and classes used to stored at the same place but it was found that the class loading occur much rarely that the objects are being collected .Dividing the storage in two different areas resulted into significance performance improvement.

Garbage Collector

- Garbage collector solves the burden of freeing the unused memory locations.
- The objects which are not used for longer duration and are not needed by the program can be thrown away.
- Garbage Collector determines the objects which not being referenced by the program and free that memory area occupied by such unreferenced objects.
- In the process of freeing the memory the garbage collector must run the finalizer methods of the objects being freed.
- A Garbage collector may also combat Heap fragmentation.

We enable you to leverage knowledge anytime, anywhere!

Education and Research

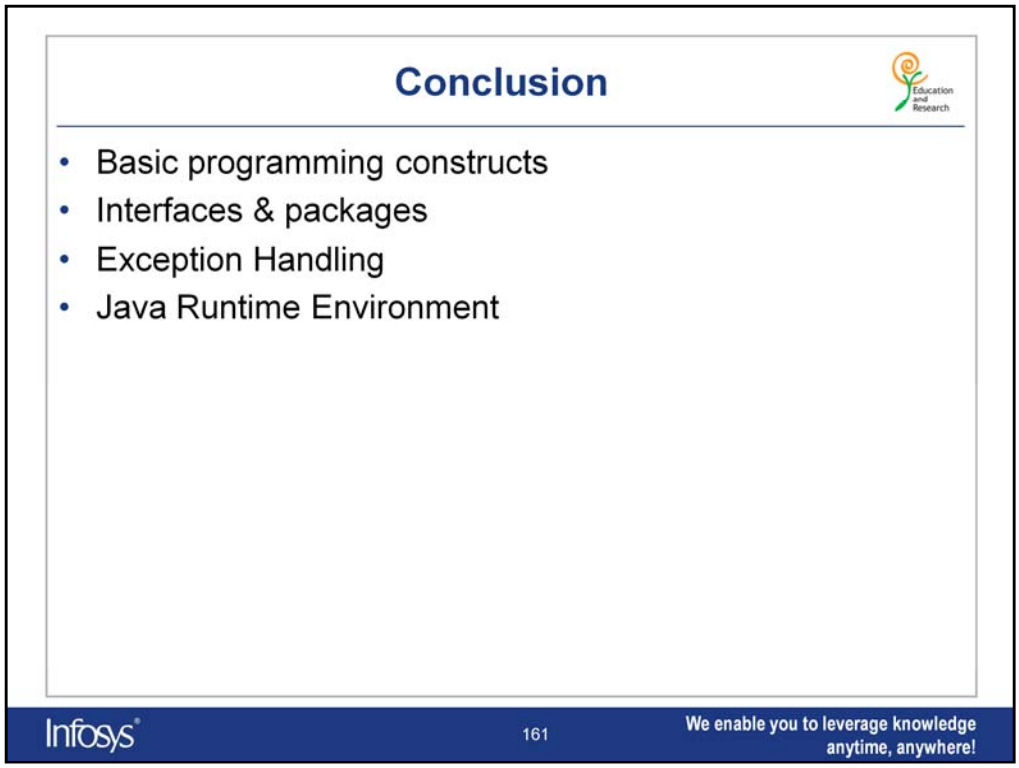
Java Programming Part 1

Conclusion

Education & Research

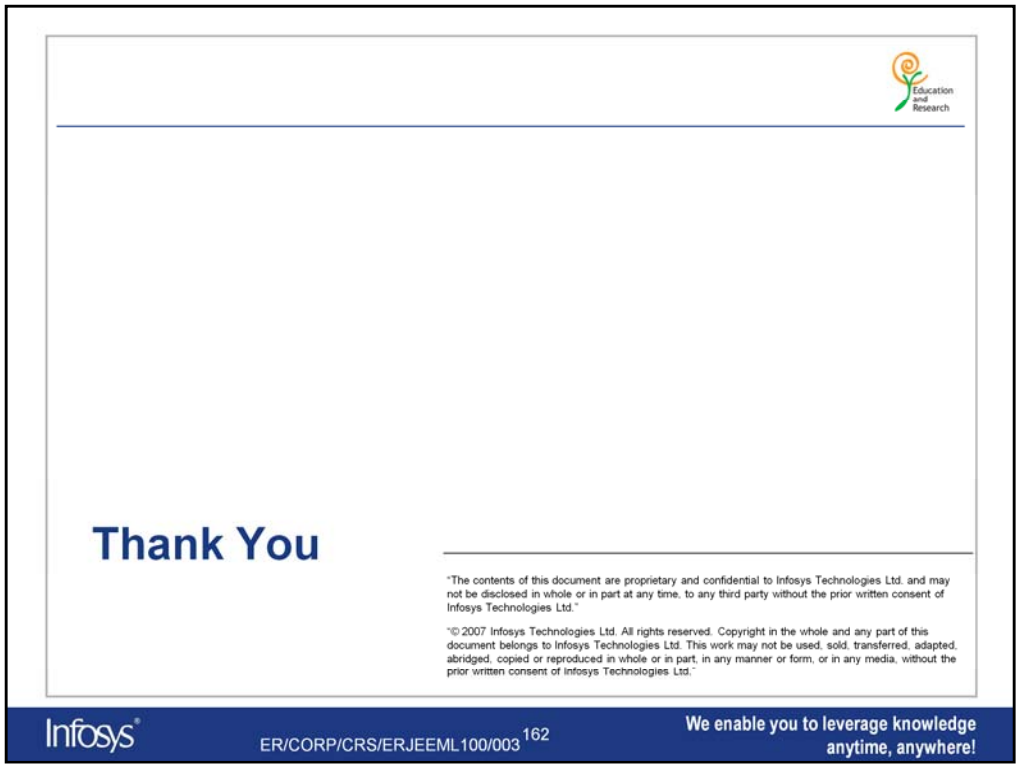
Infosys

ER/CORP/CRS/ERJEEML100/003



With this e-learning you have learnt,
What are the Basic programming constructs available in Java.
The concept of Interfaces & packages
How to perform Exception Handling
Concepts of Java Runtime environment

This course is a base course which help you learn the Basic concepts of java.
Hope you enjoyed learning.
Thank you.



Make sure that the copyright notice is present in this last slide. It is inserted as a text box in this slide only and not part of 'Slide Master'.