

## Programming in Java

### Review of some of the basic object-oriented concepts in C++

#### Static:

A data member or a member function of a class which is declared as static, belongs to the entire class and is not part of any object of the class. Hence, there is only one copy of the static members which is shared by all objects, created for the class. Static members can be accessed by using the class name. Objects cannot be used to access the static members.

(eg) class ABC

```
{
public:
    int y;
    static int x;
    static void display( )
    {
        ....
    }
}
```

int main( )

```
{
    ABC ob1;           // object 'ob1' is created for the class ABC
    ABC.x = 10;         → since the data member x & the member function 'display( )' are static
    ABC.display( );     → members, class name 'ABC' is used to access them.
    ob1.y = 100;        → 'y' is a non-static member of class 'ABC'. So, it can be accessed by
                        using the object name, 'ob1'.
}
```

### Access Mode

- A class member (data member or member function) can be declared with any of the following three access mode specifiers
  - i) private  
A private member is accessible only within the class, in which it is declared. (ie., member functions within that class, can only access them).
  - ii) public  
A public member can be accessed even outside the class in which it is declared.
  - iii) protected  
A protected member can be accessed by the class in which it is declared and by its derived classes.

### Java

- In addition to the three access modes specified above, Java introduces yet another access mode specifier – 'Default' mode.
  - iv) 'default' access mode (or) 'package-private' mode  
To understand this, let us consider the hierarchy of Java standard class library.  
Java is a true object-oriented programming language. So, everything in Java is implemented as an object, (ie. All variables and functions should be placed inside a class).

Thus, even the built-in functions (or) predefined functions, some of which are essential for our programs to work at all (eg., `println()`, `read()`, `readLine()`) and some of which make writing our Java programs easier (eg., `sqrt()`, `sin()`, `toUpperCase()`), are defined inside classes.

(eg) `sin()` function has been defined in the class 'Math'.

`toUpperCase()` function has been defined in the class 'String'.

`println()` function has been defined in the class 'PrintStream'.

Since, Java's class library is a collection of classes, it is stored in sets of files, where each file contains a class definition.

(eg) File1	File2	File3	File4	File5....
class String	class System	class Math	class PrintStream	class FileOutputStream
{	{	{	{	{
...	...	...	...	...
<code>toUpperCase()</code>	}	<code>sin()</code>	<code>println()</code>	}
{		{	{	
...		...	...	
}		}	}	
}		}	}	

The classes which perform related functions are grouped together into related sets called, packages (analogy to header files in C or C++). Each package is stored in a separate directory.

Eg: (say) package1 stored in Directory1.

File1	File2	File3
class String	class System	class Math
{	{	{
...	...	...
}	}	}

Actually, these classes have been stored in the package java.lang in the standard class library.

(say) package2 stored in Directory2.

File4	File5
class PrintStream	class
{	FileOutputStream
private int x;	{
public int y;	...
protected int z;	}
int a;	
} Assumed variables	

In Java standard class library, these classes are grouped together in 'java.io' package

Let us assume there are 4 variables in the class 'PrintStream'.

- 'x' is a private variable. So it can be accessed only within that class directly.
- 'y' is a public variable. So, it can be accessed by any other class, defined elsewhere.

- 'z' is a protected variable. Hence can be accessed only by 'PrintStream' class and by those classes that are derived from 'PrintStream'.
- 'a' is a variable, which is declared without any access specifier. Thus, it assumes 'default' access mode which means that it can be accessed by all the classes defined within that package 'java.io'. Thus the class 'FileOutputStream' can also access the variable 'a', but the classes 'String', 'System' and 'Math' cannot access it, as they are outside the package 'java.io'. 'default' access mode can be imagined as a mode in between 'private' and 'public' mode.  
ie., a 'default' member is public to all the classes within the same package, in which it is defined. Hence, the name 'package-private' mode.

### Most frequently used Java Predefined Packages

- All packages are prefixed with 'java' or 'javax'.
  - 1) java.lang  
Supports the basic language features and the handling of arrays and strings. Classes in this package are essential for our programs to work and so it is always automatically loaded with all programs by default. No need to import them explicitly.
  - 2) java.io  
Contains classes for data input and output operations.
  - 3) java.util  
Contains the collection framework, legacy collection classes, event models, date and time facilities, internationalization and miscellaneous utility classes (a String Tokenizer, a random-number generator and a bit-array).
  - 4) java.awt  
Contains all of the classes for creating user interfaces and for painting Graphics and images.
  - 5) java.applet  
Provides classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
  - 6) java.net  
Provides classes for implementing networking applications.
  - 7) javax.swing  
Provides easy-to-use and flexible components for building GUI's. The components in this package are referred to as Swing components.

### Naming conventions followed in Java

- Source files are generally called, 'Compilation Units'.
- Functions are called 'Methods'.
- All class names and interface names start with a leading uppercase letter. (eg) System. If the class name contains multiple words, all the words are marked with a leading upper case letter. E.g., PrintStream, FileOutputStream.
- Names of all public methods and instance variables start with a leading lower case letter. When more than one word is used for a method name, second and subsequent words are marked with a leading upper case letter.  
(eg) println( ), toUpperCase( ), isDirectory( ), currentTimeMillis( )
- Private and local variables, package names are with lowercase letters. (eg) awt, util, etc.

- Generally, method names will be verbal phrases and class names will be nouns.

### Sample Java Program

Let us start to write the 'Hello World' program using Java. A Java program has to be written which will display the message "Hello World" on to the screen.

println( ) method is the one which is used to display any value provided to it, as an argument.

Therefore, println("Hello World"); → This method prints the value and also a '\n' character at last.

But this println( ) method is a non-static member of the class 'PrintStream'.

So, an object of 'PrintStream' class should be used to invoke the println( ) method.

Moreover, that object has to be associated with the standard output device (Monitor), in order for the message to be displayed on to the screen.

<pre>class System {     public static final PrintStream out;     ...     ... }</pre>	<pre>class PrintStream {     public void println( ) { ... }     ...     ... }</pre>
--	---

Partial definition of the two classes 'System' and 'PrintStream' are shown above.

In 'System' class, there is a static member called 'out' which is an object of 'PrintStream' class.

This refers to the Standard Output Stream.

Note: Similarly, there are two other variables in 'System' class – 'in' and 'err' which refer to standard input stream and standard error stream respectively.

Thus the static variable 'out' which is an object of 'PrintStream' can be used to invoke the 'println( )' method.

⇒ out.println("Hello World");

However, 'out' is a member of 'System' class and that too a static member. So, the class name 'System' should be used to access the variable 'out'.

Thus, System.out.println("Hello World"); should be used to display the message.

Every program begins from main( ) method. So, include the previous statement inside a main( ) method.

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

As said earlier, Java is a true object-oriented language; so place the above code, inside a class.

```
class Sample → name specified by the user
{
    public static void main(String args[ ])
    {
        System.out.println("Hello World");
    }
}
```

→ Type this in notepad and save as Sample.java  
File name is given as same as that of class name.

main( ) Method:

- i) The access specifier 'public' denotes that the main( ) method is accessible to all other classes.  
Why the main( ) method should be public?  
For the reason that it must be called by code outside of its class 'Sample', when the program is started. If it is private, or default, it cannot be accessed across the package and hence the Java interpreter cannot invoke it.
- ii) Why 'static'?  
main( ) method is inside the class sample. Had it been defined as a non-static member of the class, an object of class 'Sample' will be required to invoke it. But, as the interpreter needs to invoke this method even before any objects are created, the only other way to invoke the main( ) method would be to use the class name. Using the class name is possible only if main() method is defined as the static member of class 'Sample'.
- iii) void( )  
To denote that main method does not return any value.
- iv) String args[ ]  
An array of objects of the class type String. If any command line arguments are specified after the class name when invoking the Java application, the run time system passes them to the application's main method via this array of Strings.

Execution of the Program:

Z:\>javac Sample.java

→ name of the source file to be compiled  
→ command to invoke the java compiler, 'javac'

After successful compilation '.class' file will be created whose name is same as that of the class name given in the program.

Therefore, 'Sample.class' would have been created which will contain the bytecodes.

For executing the program, give the following command to invoke the Java interpreter.

Z:\>java Sample → This will load the program into Java Virtual Machine (JVM).

Note: Give the class file name, for this.

The file name can also be different than that of the class name.

(eg) Save the previous program in a file named firstprogram.java.

Then 'firstprogram.java' is the file name.

'Sample' is the class name.

This program should be executed as follows

Z:\>javac firstprogram.java → This will create Sample.class file

Z:\> java Sample → name of the class file

Note: If a program contains more than one class, then after compilation, those many .class files will be generated as that of classes residing in the program.

Important:

Rule I:

To have different names for the class and file, the class must have 'default' access mode. If the class is declared with 'public' access specifier, then the class name & the file name should be the same.

(eg) Sample.java ✓  
public class Sample } both should  
{ be same  
    public static void main(String args[ ]) {  
System.out.println("Hello World");  
    }  
}

firstprogram.java ✗  
public class Sample } They are different.  
{ Hence wrong  
    public static void main(String args[ ]) {  
    System.out.println("Hello World");  
    }  
}

Why the source file name should be same as the class name, if the class name is defined as public?

The compiler need not search every file in the directory to find a particular class. The idea is, if you know the name of the class, you know where it is. Thus it helps in more efficient look up of source (.java) files and compiled (.class) files during compilation and more efficient class loading during execution.

This convention does not apply to non-public (default) classes, for the reason that, they have a very limited visibility and can only be used within the package where they are defined. Thus, the compiler and the Java Runtime Environment would have already located the correct files.

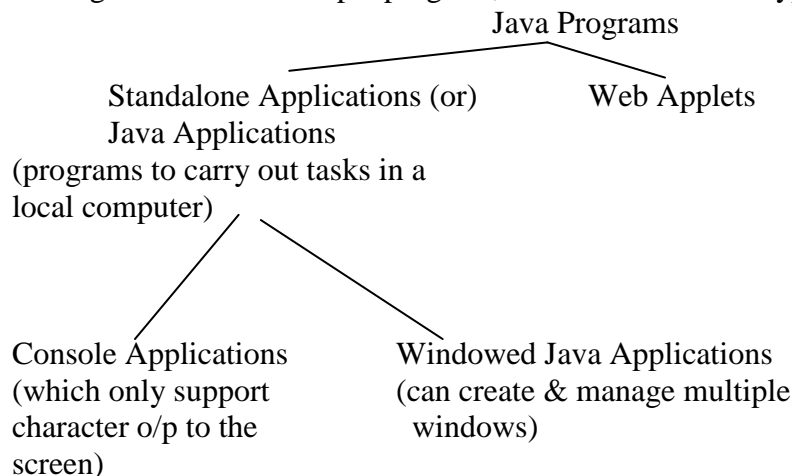
#### Rule II:

There should not be more than one 'public' class within the same source file.

Why?

After compilation, same number of .class files will be generated, as classes residing in the program. In this condition, you will not be able to easily identify which class needs to be interpreted by Java interpreter and which class contains entry point (main( ) method).

Having seen the first sample program, let us know the two types of Java programs.



#### Standalone Application:

(eg) previous "Hello World" program.

#### Web Applets:

These are small Java programs developed for Internet applications. Applets can be developed for doing everything from simple animated graphics to complex games and utilities. Since applets are

embedded in HTML document & run inside a web page, creating and running applets are more complex than creating an application.

Stand-alone programs can read & write files and perform certain operations that applets cannot do. It can run only within a web browser.

Java applets will not use the main method at all. As these are java programs that are embedded in web browsers, the web browser uses a different means of starting the execution.

## **Introduction to Java**

Computer language development occurs to implement refinements and improvements in the art of programming.

### **Earlier to C:**

- i) Fortran was good for scientific applications but not very good for system code.
- ii) Basic was easy to learn but lacked structured approach. It was not suitable for large programs.
- iii) Assembly Languages are highly efficient programs but not easy to learn or use effectively. Debugging was difficult.

Also, Basic, Fortran and COBOL relied upon GOTO as a primary means of program control.

### **Why C?**

It is the result of the need for a structured, efficient, high-level language, that could replace assembly code when creating system programs.

### **Java**

Java evolved from a project, developing a language for programming consumer electronic devices at Sun Microsystems, USA. The vision of this project was to develop smart consumer electronic devices that could all be centrally controlled & programmed from a hand held-remote-control-like device. It was realized that a platform-independent development environment was needed.

### **Primary Motivation:**

- I) As we all know, many different types of CPUs are used as controllers, in various consumer electronic devices. The trouble with C & C++ (and other programming languages) is that they are designed to be compiled for a specific target. Though it is possible to compile a C++ program for just about any type of CPU, to do so require a full C++ compiler targeted for that CPU. The problem is that compilers are expensive and time-consuming to create. Towards finding an easier and more cost-effective solution which could be a portable, platform-independent language, to produce code that would run on a variety of CPUs under differing environments, paved the way for creation of Java.
- II) At the time of development of Java, another factor was emerging – the World Wide Web (WWW). Had the web not taken shape at about the same time that Java was being implemented, Java might have remained a useful language for programming consumer electronics. With the emergence of WWW, Java was propelled to the forefront of computer language design since the web, too, demanded portable programs.

Patrick Naughton, James Gosling, Bill Joy and several others joined hands to develop the language. It was initially named "Oak" (1992) and later renamed as "Java" (1995).

Thus, aside from its ability to create programs that can be embedded in a web page, Java was designed to be machine independent.

Java is a language for professional programmers.

In addition to developing a platform-independent language, Java has refined the O-O paradigm used by C++, added integrated support for multithreading and provided a library that simplified Internet access.

Java was to Internet Programming what C was to System Programming.

### How Java changed the Internet?

Java applets are special kind of programs designed to be transmitted over the Internet and automatically executed by a Java-compatible browser.

If the user clicks a link that contains an applet, the applet will be automatically downloaded and run in the browser.

Applets are used to display data provided by the server, handle user input or provide simple functions such as a loan calculator, that execute locally, rather than on the server. They are dynamic, self-executing programs.

### Difference between C and Java

- No 'sizeof', 'typedef'
- No data types like 'struct' & 'union'
- No keywords like auto, extern, register, signed, unsigned
- No explicit pointer type
- Does not have a preprocessor. So, cannot use #define, #ifdef, #include . . .
- Functions with no arguments must be declared with empty parenthesis and not with 'void' keyword, which is allowed in C.
- Adds labeled break and continue statements
- Adds new operators such as >>> and instanceof.

### Difference between C++ and Java

- Java is a pure OO language but C++ is C with O-O extension.
- All class objects should be dynamically allocated.
- No operator overloading.
- No multiple inheritance but accomplished by a feature called 'interface'.
- No template classes as in C++.
- No pointers.
- No global variables. Every variable & method is declared within a class.
- No header files in Java.
- No destructor function. Instead finalize( ) method is used.
- Java **uses call by value** to pass all arguments but the effect differs between whether a primitive type or a reference type is passed.

### Java Environment

- JDK (Java Developer's Kit)
- JRE (Java Runtime Environment)
  - Java API (Application Program Interface)
  - JVM (Java Virtual Machine)

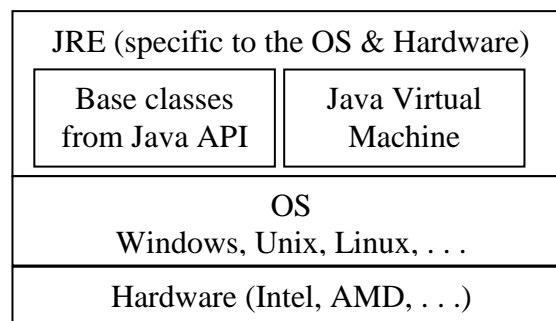


### Java Developer's Kit (JDK)

- Collection of tools used for developing and running Java programs. There are 7 components of JDK.
  - i) javac (Java Compiler) – Compiles the Java source code and creates the class file, which consists of bytecodes.
  - ii) java (Java Interpreter) – executes the class file by translating the byte codes.
  - iii) javap (Java Disassembler) – used to display public interfaces of a class file. We can get both methods and variables of the public interfaces.
  - iv) jdb (Java Debugger) – helps to find errors in the Java program. It is a command line debugger and so it is not of much use as a GUI debugger.
  - v) javah (Java Header File Generator) – produces header files that enable the user to use native C or C++ code within a Java class.
  - vi) javadoc (Documentation) – helps to create documentation in the form of HTML files, from Java source code files automatically.
  - vii) appletviewer – helps to run Java applets without actually using a Java Compatible Browser.

### JRE (Java Runtime Environment)

Java has been designed to work in a platform-independent manner. In order to achieve this dream concept of "Write Once and Run Anywhere (WORA)", we have the JRE running in the machine on top of the Operating System. Every JRE is specific to a particular OS and the hardware used. JRE are available for a wide variety of hardware & software combinations, making Java, a very portable language.



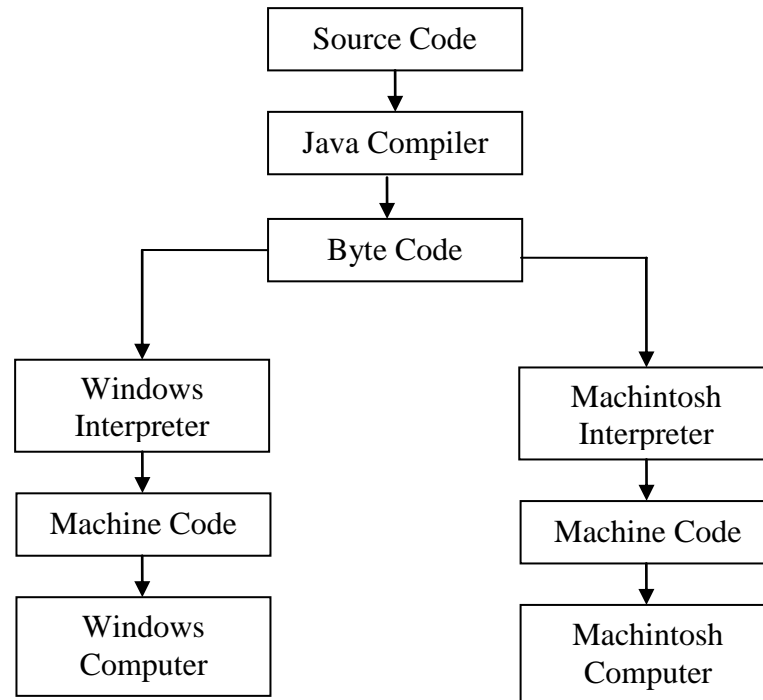
#### I) Java API

Java Standard Library or (API) includes hundreds of classes & methods grouped into several functional packages.

(eg) java.lang , java.io , java.net , java.util , java.applet , java.awt

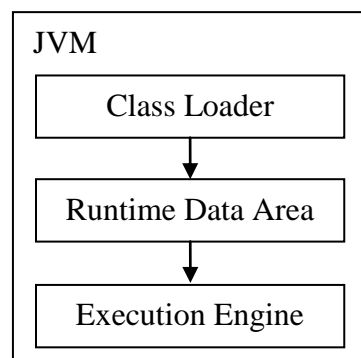
The JRE contains a set of core class libraries that are required for the execution of Java programs.

It also incorporates JVM (Java Virtual Machine) which is an abstract machine. It can be implemented in software to varying degree of hardware. JVM is a virtual CPU that is simulated by a program to run on the actual CPU called native hardware.



The output of java compiler is bytecode which is machine-independent. Byte codes are highly optimized set of instructions designed to be executed by the Java Runtime system, called JVM. Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments. Only the JVM needs to be implemented for each platform.

#### Architecture of JVM:



The class loader loads the classes to be executed to the runtime data area & the execution engine executes the instructions in the given sequence, as per the logic and produces the results.

#### Runtime Data Area

- i) Heap
- ii) Stack
- iii) Method area
- iv) Registers
- v) Runtime Constant Pool (or) Native method execution stack

methods in C or C++

Though JVM differs from platform-to-platform, all understand the same Java Bytecode.

If a Java program were compiled to native code, instead of bytecode, then different versions of the same program would have to exist for each type of CPU connected to the internet which is not feasible.

Generally, if a program is compiled to an intermediate form & the interpreted by a virtual machine it runs slower than it would run, if compiled to exe code. But in Java, difference is not so great. Since bytecode has been highly optimized, the use of bytecode enables the JVM to execute programs much faster than you might expect.

- a) The simplest form of execution engine (in JVM) just interprets the bytecode one at a time. Thus Java interpreter is a reasonable name for a JVM that interprets bytecode.
- b) Another type of execution engine called Java HotSpot Compiler is faster, but it requires more memory. In this scheme, the bytecode of a method are compiled to native machine code, the first time the method is invoked. This native machine code is cached so that when the same method is invoked again, the cached code is reused & the execution is completed fast.
- c) A third type of execution engine is adaptive optimizer. In this scheme, the virtual machine compiles to native code & just optimizes the heavily used portions of the codes. The code that is not heavily used remains as bytecode and is interpreted.

### Features of Java

- Simple, small and familiar
  - Java inherits C/C++ syntax and many object-oriented features of C++. Hence, easy to learn Java.
  - Redundant or sources of unreliable code are not part of Java. (eg) no pointers, no preprocessor header files, no goto, no operator overloading, and multiple inheritance.
- Secure
  - An applet program that downloads and executes automatically on the client computer must be prevented from doing harm to that machine. Java achieves this protection by confining an applet only to the Java execution environment and not allowing it access to other parts of the computer.
  - Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet.
  - Absence of pointer in Java ensures that programs cannot gain access to memory locations without proper authorization.
  - Java cannot allow pointers, because doing so would allow Java programs to breach the firewall between the Java execution environment and the host computer. (A pointer can be given any address in memory – even addresses that might be outside the Java runtime system). Lack of pointers is not a significant disadvantage to Java. It is designed in such a way that as long as you stay within the confines of the execution environment, there would not be any need to use a pointer.
- Portable
  - WORA – Write Once, Run Anywhere, Anytime, forever
  - Java programs can be easily moved from one computer system to another, anywhere and anytime.
  - Changes and upgrades in OS, processors and system resources will not force any changes in Java programs. That's why Java is a popular language for programming on Internet which interconnects different kinds of systems worldwide. A Java applet can

be downloaded from a remote computer onto our local system via Internet and executed locally.

- Portability is achieved by 2 things
  - a) Generates bytecode that can be implemented on any machine.
  - b) Sizes of the primitive data types are machine-independent.
- Object-oriented
  - Java is a true O-O language. Everything in Java is an object. It comes with an extensive set of classes, arranged in packages, which we can use in our programs by inheritance.
  - Object model in Java is simple and easy to extend while primitive data types, like Integers are kept as high-performance non-objects.
- Robust
  - Two main reasons for program failure are
    - a) Memory Management Mistakes  
Memory management was tedious in traditional languages like C/C++, as we must manually allocate and free all dynamic memory. This leads to problems when we forget to free memory that has been previously allocated or worse when we try to free some memory, that another part of their code is still using.  
Java is designed as garbage collected language relieving the programmers, from virtually all memory management problems. Thus, deallocation of unused objects, is completely automatic.
    - b) Mishandled Exceptional Conditions (ie., run-time errors)  
Exceptional conditions arise in situations like "division by zero", or "file not found". But Java provides object-oriented exception handling for this. All run time errors in a Java program can and should be managed by the program.
- Multi-threaded
  - Multithreading means handling multiple tasks simultaneously. Java supports multi-threaded programs. (eg) we can listen to an audio clip while scrolling a page and at the same time, download an applet from a distant computer. This improves the interactive performance of graphical applications. The JRE comes with an elegant solution for multiprocess synchronization that enables you to construct smoothly running interactive systems.
- High Performance
  - Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high-performance, by using a Just-in-Time compiler, without losing the benefits of platform independent code.
- Interpreted
  - Java is both compiled and interpreted.
- Distributed
  - Java was designed for distributed environment of Internet, as it handles TCP/IP protocols.
  - Accessing a resource using URL is not much different from accessing a file. Java also supports Remote Method Invocation (RMI) which enables a program to invoke a method across a network.
  - Has the ability to share both data and programs.
  - Enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

- Dynamic
  - Supports functions written in C & C++, called native methods. This facility enables the programmers to use the efficient functions available in these languages. Native methods are linked dynamically at run time.
  - Java is capable of dynamically linking in new class libraries, methods and objects.
  - Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response.
- Servlets
  - Java on the server side.
  - Servlet is a small program that executes on the server. As applets dynamically extend the functionality of a web browser, servlets dynamically extend the functionality of a web server. It is used to create dynamically generated content that is then served to the client.  
(eg) an online store might use a servlet to look up the price for an item in a database. The price information is then used to dynamically generate a web-page that is sent to the browser.  
Servlet's performance is higher than CGI, in this regard.
  - They are highly portable (as they are compiled into bytecode & executed by JVM).  
Only requirement: Server should support the JVM & a servlet container.

### Java Object-Oriented Programming Concepts

Primary motivation for the development of various languages is to manage the increasing complexity of programs that are manageable and reliable.

Disadvantage of a structured programming language like C:

C is a process-oriented model i.e., it gives importance to what is happening.

There is no code reusability (with inheritance concepts)

C++ is basically a procedural language with object-oriented extension.

Java is a pure object-oriented language, latest C#.

#### Object-Oriented Programming:

Allows us to decompose a problem into a number of entities called objects and then build data and functions around these entities.

It is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

i.e., object is considered to be a partitioned area of computer memory that stores data & a set of operations that can access the data.

Since the memory partitions are independent, the objects can be used in a variety of different programs without modifications.

Three Object-Oriented Programming Principles:

- 1) Encapsulation
- 2) Inheritance
- 3) Polymorphism

#### 1) Encapsulation:

Mechanism that binds together data and code that manipulates it and keeps them safe from outside interference & misuse.

Insulation of the data from direct access is Data Hiding. In Java, the basis of encapsulation is the class.

#### Class:

Defines the structure & behavior (data & code) that will be shared by a set of objects. Each object of a given class contains the structure and behavior defined by the class. Thus, objects are referred to as instances of a class.

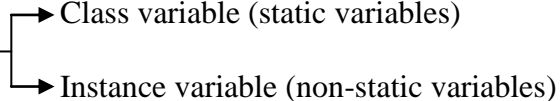
A class is a data type & hence cannot be directly manipulated. It is a logical construct.

#### Object:

Instance of class data type. It is a physical reality.

Instantiation of an object is the process of creating an object of a particular class.

Code and data of a class are its members.

Data is called member variable — 

Code that operates on the data is called member method or just method.

Purpose of a class is to encapsulate complexity which is done by making each method or variable in a class as private or public.

Public interface of a class represents everything that external users of the class need to know or may know. The public interface should be carefully designed not to expose too much of the inner workings of the class.

#### Data Abstraction:

Act of representing essential features without including the background details or explanations. It is the process of abstracting common features from object and procedures and creating a single interface to complete multiple tasks.

(eg) a programmer may note that a function that prints a document exists in many classes & may abstract that function, creating a separate class that handles any kind of printing.

#### 2) Inheritance:

Process by which object of one class acquires the properties of objects of another class. It provides the idea of reusability. We can add additional features to an existing class without modifying it.

Each subclass (derived class) defines only those features that are unique to it. Without this, each object would need to define all of its characteristics explicitly.

#### 3) Polymorphism:

Creating a generic interface to be used for a group of related activities. It is the ability to take more than one form.

(eg) the operator '+' can be used to add 2 numbers

'+' can also be used to concatenate 2 strings

#### Dynamic Binding:

Binding: Linking of a procedure call to the code to be executed in response to the call.

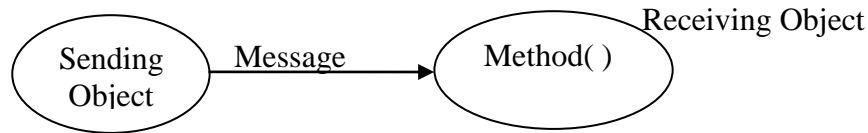
Dynamic Binding: The code associated with a given procedure call is not known until the time of the call at runtime. It is associated with polymorphism & inheritance.

(eg) draw( ) procedure is redefined in each subclass that defines the object. At runtime, the code matching the object under current reference will be called.

### Message Communication:

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another.

A message for an object is a request for execution of a procedure and therefore will invoke a method in the receiving object that generates the desired result.



### Object Based Language:

- Supports only data abstraction & not inheritance or polymorphism and message based communication.
- Supports encapsulation & object identity.  
Object-based Language = Encapsulation + Object identity  
Object-oriented Language = Object-based features + Inheritance + Polymorphism

### New Language Features of Java SE (Standard Edition 7)

- A string can control a switch statement.
- Binary integer literals.
- Underscores in numeric literals.
- Expanded try statement called try-with-resources that supports automatic resource management (eg: streams can now be closed automatically when they are not needed).
- Type inference (via the diamond operator) when constructing a generic instance.
- Enhanced exception handling – 2 or more exceptions can be caught by a single catch.
- Compiler warnings associated with some types of varargs methods have been improved.
- Fork/Join framework provides important support for parallel programming.

## **Working with Command Line Arguments**

- User input can be provided through command line arguments.
- The command line arguments are specified after the class name when invoking the Java application.
- The run time system passes them to the application's main method via an array of Strings.

Example 1: Write a program to read the first name & last name of a student using command line arguments and display them.

```
public class studentName    → It is recommended to use 'public' access specifier for the class, so  
{                          that it is available across the package in which it is defined. So,  
    public static void main(String args[ ])    definitely, the name of the file should be  
    {                                          'studentName.java'  
        System.out.print("The student's name is:");  
        System.out.println(args[0] + args[1]);
```

```

    }
}

```

print( ) method – displays the value provided to it as an argument and does not print a '\n' character at the end, unlike println( ) method.

printf( ) method can also be used to display formatted output.

When executing this program, the command should be as follows:

Z:\> java studentName James Stephenson

After the class name 'studentName', two values are supplied by the user. If no values are given while executing (ie., if you simply give 'java studentName'), a run time error will be produced (ArrayIndexOutOfBoundsException) as the program expects 2 command line arguments, args[0] & args[1].

These 2 values will be received by the args[ ] array. It is an array of String objects, as said earlier. So, the values given in the command line are received as string values and stored in the args[ ] array.


Therefore, args[0] will have string "James" stored in it.

args[1] will have string "Stephenson" stored in it.

To display them together as a single name, '+' operator is being used.

'+' operator, when used with two strings, concatenates them, whereas when used with two numeric values, adds them. In the above program, args[0] & args[1] are strings. So, '+' concatenates them.

The println( ) method could have been written like the one below.


 A single space character is also concatenated.  
 System.out.println(args[0] + " " + args[1]);

This inserts a space in between the 2 names, giving an output like James Stephenson

Example 2: Write a program to read the marks of 5 subjects scored by a student using command line arguments and display his average.

```

public class Average
{
    public static void main(String args[ ])
    {
        float avg; int m1, m2, m3, m4, m5;
        m1 = Integer.parseInt(args[0]);
        m2 = Integer.parseInt(args[1]);
        m3 = Integer.parseInt(args[2]);
        m4 = Integer.parseInt(args[3]);
        m5 = Integer.parseInt(args[4]);
        avg = (float) (m1 + m2 + m3 + m4 + m5) / 5;
        System.out.println("His average is:" + avg);
    }
}

```

While executing the program, all the 5 subjects marks should be provided as command line arguments.

Z:\> java Average 70 75 80 65 90

These values will be received by the first 5 elements of args[ ] array.

∴ args[0] will now contain 70, args[1] – 75, args[2] – 80, args[3] – 65 and args[4] – 90

Though these are numeric values, they are received as String objects in the args[ ] array.



If these values are to be used in arithmetic operations, the string values have to be converted into integers, floats or any other numeric data type values.

What has to be done for doing this conversion?

For this, let us have small introduction on Wrapper Classes.

Only primitive data types are not implemented as objects in Java, for efficiency and performance considerations. However, even for these primitive data types, object representations are required for the following reasons, sometimes.

- i) There are Collection classes which deal with only objects. So, to store a primitive data type in one of these classes, we need to wrap the primitive type in a class.
- ii) When primitive data types are passed to a function, they are passed by value. But if they are converted into objects & passes, though they are passed by value, it will however produce the effect of passing by reference.

Thus, for all the 8 primitive data types in Java, equivalent Wrapper Classes are there.

Data type	Wrapper Class	
1) boolean	<u>B</u> oolean	Note that the class names start with capital letter. Except Boolean & Character, all other classes are subclasses of an abstract class called ' <u>N</u> umber'. So, all these <u>6</u> classes and Boolean class have their respective parsexxx( ) method.
2) byte	<u>B</u> yte	
3) char	<u>C</u> haracter	
4) short	<u>S</u> hort	
5) int	<u>I</u> nteger	
6) long	<u>L</u> ong	
7) float	<u>F</u> loat	
8) double	<u>D</u> ouble	

ie.,

Class	Methods
Boolean	parseBoolean(String s)
Byte	parseByte(String s)
Short	parseShort(String s)
Long	parseLong(String s)
Integer	parseInt(String s)
Float	parseFloat(String s)
Double	parseDouble(String s)

Note: For Character class, there is no equivalent parsexxx( ) method.

- These are static methods and hence have to be invoked with the class name only.
- These methods convert the string representation of any numeric value to their corresponding number representations.

↗ Class name itself is used for invoking the method as it is static

(ie.,) Float.parseFloat("57.8"); → will give the o/p 57.8

Integer.parseInt("12"); → will give the result 12

In our example, the command line arguments have to be converted into integers. Thus we use the `parseInt( )` method of 'Integer' class to convert `args[0]`, `args[1]`, . . . `args[4]` into their equivalent integers & store them in the variables `m1`, `m2`, `m3`, `m4` & `m5`.

Try these:

```
int n = 5, m = 10;
system.out.println("The 2 values are: " + n);    → Output: The 2 values are: 5
system.out.println(m);                          10
system.out.println("The sum is: " + m + n);      The sum is: 105
```

Instead of displaying the expected result 15, 105 is displayed. This is because, the '+' operator

When used with 2 integers, adds them;

But when used with 2 strings, concatenates them;

Also when used with a string and an integer (numeric data type) concatenates them.

If the previous statement is considered

"The sum is: " + m + n

'+' operators follows L → R Associativity.

∴ "The sum is: " + m expression will be evaluated first yielding → "The sum is: 10"

Now this is a string to which 'n' is added (ie., concatenated)

"The sum is: 10" + n is evaluated second producing the output The sum is: 105

How to print '15' as we expected?

i) `System.out.println("The sum is: " + (m + n));`

Use parenthesis to violate the normal operator precedence order. So, (m + n) will be evaluated first, giving 15 to which the string "The sum is: " is prefixed next.

→ The sum is: 15

ii) `System.out.println(m + n + " is the sum");`

From L → R, '+' is first used for 2 integers. Therefore, addition takes place (&not concatenation), giving 15.

The expression now reduces to 15 + " is the sum".

One integer + One string goes for concatenation operation.

Output: 15 is the sum

iii) `System.out.print("The sum is:");`

`System.out.println(m + n);` → Output: The sum is 15

Using other operators like \*, %, /, etc in the `println( )` method is quite straight forward.

(eg) `System.out.println("The product is: " + m * n);`

→ The product is: 50

`System.out.println("The quotient is: " + m / n);`

→ The quotient is: 2

} '\*', and '/' have higher precedence and so will be evaluated first even before concatenation operator '+'

But, `System.out.println("Difference is: " + m - n);` → Error

Because, ("Difference is: " + m) is evaluated first & results in a string. Trying to subtract an integer 'n' from a string will throw an error.

Ex – 2: `inthis_age = 20;`

`System.out.println("His age after 2 years will be " + (his_age + 2));`

→ His age after 2 years will be 22  
System.out.println("His age after 2 years " + his\_age + 2);  
→ output: 202

## Scanner Class

Using Scanner class for getting user input:

Scanner class is defined in the java.util package.

∴ To write a program using 'Scanner' class, import java.util package as follows:

import java.util.\*; → This imports all the classes defined in 'util' package.

(or)

import java.util.Scanner; → imports only the Scanner class in the 'util' package.

If more than 1 class of a package has to be used, it is better to go for the first way of importing the classes. However, there is no need to write an import statement as 'import java.lang.\*;', as this package is imported by default.

Scanner Class:

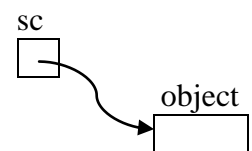
It is used to read formatted input from the console, a string, a file or any source that implements the 'Readable' interface or 'ReadableByteChannel'.

To make use of this Scanner class, to read user input, first an object of this class must be created. In Java, all objects are created dynamically (at run time), using new operator.

(eg) new Scanner(System.in); → Creates an object for Scanner class & initializes it by invoking the Scanner(. . .) constructor.

In the sub-expression above, 'new' operator instantiates the Scanner class (ie., creates an object for the Scanner class); it also invokes the constructor of the class; after creating the object (ie., allocating space for the object in the memory) it returns a reference to the object (address), it creates. This reference (address) should be stored in an appropriate variable. To store the reference of the 'Scanner' object, the reference variable, should be of the same type, 'Scanner'.

{ Scanner sc; → creates a reference variable 'sc' of type 'Scanner'  
sc = new Scanner(System.in); → creates an object for 'Scanner' & stores its  
reference in the reference variable, 'sc'



The above 2 statements can be combined into a single line as follows.

Scanner sc = new Scanner(System.in);

This reference variable can be used to invoke the methods of the Scanner class, for reading the input.

There are several types of constructors defined in Scanner class. The most important of these are

i) Scanner(File f)

→ This constructor creates an object of Scanner class that uses the file reference variable 'f', as the source of input.

Example can be learnt in 'I/O Streams' chapter.

ii) Scanner(String s)

→ This constructor creates an object of Scanner class that uses the string 's' as the source of input.

Example:

String s = "Anu 9.3 Ajay 8.9";

Scanner sc = new Scanner(s); → When appropriate methods are invoked for reading input, it will be read from the string.

iii) Scanner(InputStream is)

→ This constructor creates an object of Scanner class that uses the specified inputstream as the source of input.

Example: Scanner sc = new Scanner(System.in);

The standard input device, which is the keyboard by default, is provided as the source of input.

Having studied how to instantiate the Scanner class & its various constructors, let us see, some of the methods in the Scanner class, useful for getting the input.

Return Type	Method	Description
String	next( )	Reads the next token of any type from the input source & returns it.
String	nextLine( )	Reads & returns the next line of input as a String.
int	nextInt( )	Reads the next token as an integer value
long	nextLong( )	Reads the next token as a long value.
short	nextShort( )	Reads the next token as a short value.
byte	nextByte( )	Reads the next token as a byte value.
float	nextFloat( )	Reads the next token as a float value.
double	nextDouble( )	Reads the next token as a double value.
boolean	nextBoolean( )	Reads the next token as a boolean value.

Note:

Listed above are only a few methods which will suffice for writing basic programs. Other methods can be learnt from the book. Also note that the method names follow the naming convention (first word starts with lowercase letter & the subsequent words start with a leading capital letter – (eg) nextByte( )). There is no method for taking character input.

Example:

Write a Java program using Scanner class to read the Employee name, EmpId, and salary and display them back.

```
import java.util.*;
public class Scanner_Sample
{
    public static void main(String args[ ])
    {
        Scanner sc = new Scanner(System.in);
        String s = sc.next( ); → returns the read value in the form of a String.
        int id = sc.nextInt( ); → returns the read value in the form of an integer.
        float salary = sc.nextFloat( ); → returns the read value in the form of a float.
        System.out.println("Emp name:" + s + " Id:" + id + " Salary:" + salary);
    }
}
```

If the input is given as

Anu

10  
15000.00

they are read using the methods shown above and stored in the appropriate variables.

Note: Instead of next( ) method, nextLine( ) could also have been used, in the example, as both return a String.

next( ) and nextLine( ) methods can be used for reading numeric input type also.

(eg) value 23.5 can be read using next( ) or nextLine( ) methods. But since a string representation of the value 23.5 ("23.5") is being returned, it cannot be directly used in arithmetic operations. Hence they have to be converted into the corresponding numeric type, using the parseXXX( ) methods.

(eg) Assume 23.5 is given as input.

```
String s = next( );
```

```
System.out.println(s); → O/P: 23.5
```

```
int x = s * 10; → error because the string "23.5" cannot be used in multiplication operation.
```

So, 

```
int n = Integer.parseInt(s);
```

```
int x = n * 10;
```

 } should be done

Difference between next( ) and nextLine( ) methods:

next( )	nextLine( )
<ul style="list-style-type: none"> <li>- Can read the input only till the space. It cannot read 2 words separated by space.</li> <li>- After reading the input, it places the cursor in the same line.</li> <li>- It reads the new tokens &amp; leaves the '\n' character in the buffer itself. If there is another call to the next( ) method, it simply throws the '\n' character in the buffer, away and reads the next token. <u>Any nextXXX( ) method works the same way.</u></li> </ul>	<ul style="list-style-type: none"> <li>- Reads the input including the space between the words (ie., till '\n' character).</li> <li>- After reading the input, it positions the cursor in the next line.</li> <li>- It reads up to '\n' character but if there is any '\n' character left in the buffer, by the previous next( ) methods, it is read as the next token by this method, as a blank line.</li> </ul>

Example:

Write a program to read the Employee ID (Integer) and the Employee Name.

Assume the input to be

10

James Anderson

→ The input is stored in the buffer as  
ie., 10 \n James Anderson \n

```
Scanner sc = new Scanner(System.in);
```

```
int ID = sc.nextInt( ); → reads 10 and leaves the '\n' character in the buffer itself.
```

```
String s = sc.next( ); → throws the '\n' character in the buffer & reads up to space character (James)
```

```
System.out.println(ID + " " + s);
```

Output: 10 James

If the full name has to be read, nextLine( ) method must be used. Let us explore what happens if nextLine( ) method is used.

Input: 10 \n James Anderson \n

```
int ID = sc.nextInt( ); → reads 10 and leaves the '\n' character in the buffer itself.
```

String s = sc.nextLine(); → reads the '\n' character left behind in the buffer by the previous nextInt() method & thus displays a '\n' character as the output, which is not visible, like other characters.

System.out.println(ID + " " + s);

Output: 10

How to overcome this problem?

In case you need to use a nextLine() method, after calling any of the next(), nextInt(), ... methods, provide an extra nextLine() method before actually starting to reading the subsequent input data.

Previous Example Input: 10 \n James Anderson \n

int ID = sc.nextInt(); → reads 10 and leaves the '\n' character in the buffer itself.

sc.nextLine(); → reads the '\n' character left in the buffer

String s = sc.nextLine(); → read the next data James Anderson

System.out.println(ID + " " + s);

Output: 10 James Anderson

Write a program to read the name of the school, number of programmes (B.tech, M.Tech, MCA, M.S, ...) offered in the school and the name of the School Director.

Sample Input:

SITE

8

Dr.Saravanan R (next() method will not read the last name 'R')

```
import java.util.Scanner;
```

```
public class School
```

```
{
```

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

```
{
```

```
    Scanner sc = new Scanner(System.in);
```

```
    String school = sc.next(); → Reads 'SITE' & leaves the '\n' in the buffer
```

```
    int nr_pgm = sc.nextInt(); → Throws the '\n' char in the buffer & reads 8; it also leaves the '\n' char (following 8), in the buffer itself.
```

```
    sc.nextLine(); → Reads the '\n' character
```

```
    String director = sc.nextLine();
```

```
    System.out.println(school + " offers " + nr_pgm + " programmes and is headed by" + director);
```

```
}
```

```
}
```

Output: SITE offers 8 programmes and is headed by Dr.Saravanan R

Note:

The above program can also be written without import statement, with the following modifications:

```
java.util.Scanner sc = new java.util.Scanner(System.in);
```

In all the places where 'Scanner' class has to be used, the entire path including the package name should be specified. Hence, it is easier to write the 'import' statement, to avoid this.

If a 'next' method cannot find the type of data it is looking for, it throws an InputMismatchException.

(eg) int x = sc.nextInt();

If the input is "Hello", InputMismatchException (error) will be thrown, because nextInt( ) method expects an integer where as it sees a string as the input.

For this reason, it is better to first confirm that the desired type of data is available before calling the 'next' method.

Return Type	Method	Description
Boolean	hasNext( )	Returns true if another token of any type is available else returns false.
Boolean	hasNextLine( )	Returns true if a line of input is available.
Boolean	hasNextInt( )	Returns true if an int value is available for reading. Else returns false.
Boolean	hasNextLong( )	Returns true if a long value is available for reading. Else returns false.
Boolean	hasNextShort( )	Returns true if a short value is available for reading. Else returns false.
Boolean	hasNextByte( )	Returns true if a byte value is available for reading. Else returns false.
Boolean	hasNextFloat( )	Returns true if a float value is available for reading. Else returns false.
Boolean	hasNextDouble( )	Returns true if a double value is available for reading. Else returns false.
Boolean	hasNextBoolean( )	Returns true if a boolean value is available for reading. Else returns false.

All these methods determine if the specified type of input is available.

Previous Example:

```

if(sc.hasNextInt( ))      → If the input is "Hello", the program simply
int x = sc.nextInt( );    terminates as the 'else' part is executed. If the
else                      input is 5, it will be assigned to 'x'.
System.exit(1);

```

Note:

exit( ) method terminates the currently running JVM. It is a convention to use '0' to denote a normal exit and any other number to denote an abnormal termination.

Difference between [hasNext( ) and hasNextLine( )] and other hasNextXXX( ) methods:

hasNext( ) and hasNextLine( ) methods return true, for any type of input that is given. Even if the enter key is pressed, denoting a new line character, these methods read the '\n' character and return true.

Example:

Write a program to continuously read the integers from the user, until any other data type is entered. and find their average.

```

import java.util.*;
public class ReadInput
{
public static void main(String args[ ])
{

```

```

int x, sum=0, n=0;
Scanner sc = new Scanner(System.in);
while(sc.hasNextInt()) → This loop will continue as long as you enter integer values.
{
    x=sc.nextInt();           Once a string, float or any other data apart from integer is
    n++;                     entered, this loop will terminate.
    sum+=x;
}
floatavg = (float) sum / n;
System.out.println(avg);
}

```

I/P: 1  
 2  
 3  
 2.5 → not an integer, so the loop does not  
 continue to read input  
 O/P: 2.0

If `hasNext()` method is used, in the loop condition, it works as an infinite loop, as this method would return true for all kinds of input. Hence, there must be some statement within the loop, to terminate the loop.

<pre> while(sc.hasNext()) {     if(sc.hasNextInt( )     {         x=sc.nextInt();         n++;         sum+=x;     } } </pre>	<p>Let the Input be</p> <p>1 2 3</p> <p>Hello → This makes the loop go into an infinite loop, as the same string "Hello" will be processed by the <code>hasNext()</code> method continuously for all the iterations, returning true.</p>
---	--

To convert this into a finite loop which stops when any input other than integer is entered, 'else' part can be provided.

```

while(sc.hasNext())
{
    if(sc.hasNextInt( )
    {
        x=sc.nextInt();
        n++;
        sum+=x;
    }
    else
    break;
}

```



Having multiple methods within the same source file:

Write a program to call the function factorial( ) recursively to calculate the factorial of a given number and invoke this from the main method.

Important:

Remember main( ) is a static method. Therefore, it can call only other static methods directly. Non-static methods can be called through an object of the class, or the reference variables which references the object.

Therefore, there are 3 ways of writing the factorial( ) method:

- i) Write the factorial( ) method as a static method and call this method directly from the main method.
- ii) Write the factorial( ) method as a non-static method and call this method, using the object created for the class.

(eg) new factorialclass( ) . factorial( ) Invoke the recursive method

Creates the object for the class which includes the main( ) and the factorial( ) method.  
This object can be used to invoke the factorial( ) method.

- iii) Write the factorial( ) method as a non-static method. Call this method by making use of the reference variable which references the object created for the class.

factorialclass f = new factorialclass( );  
f.factorial( );

Program:

```
class factorialclass
```

```
{
    int factorial(int n)
    {
        if(n<1)
            return 0;
        else if(n==1)
            return 1;
        else
            return n*fact(n-1);
    }
}
```

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

```
{
    int n = Integer.parseInt(args[0]);
    factorialclass f = new factorialclass( );
    System.out.println(f.factorial(n));
}
```

Method (iii)

(or)

```
System.out.println(new factorialclass( ) . factorial(n));
```

Method (ii)

```
    }
}
```

## Core Java

### Method (i)

```
public class factorialclass
```

```
{
```

```
static int factorial(int n)
```

```
{
```

```
    ...
```

```
}
```

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

```
{
```

```
    int n = Integer.parseInt(args[0]);
```

```
    System.out.println(factorial(n));
```

```
}
```

```
}
```

→ call the method directly without using any object.

### Structure of a Java Program

Documentation Section	→ to write a brief explanation about the program
Package Statement	→ to organize logically related java classes It is used to create user defined packages
Import Statements	→ Imports the required packages to the current program
Interface Statements	→ to define the interfaces which will be studied later
<pre>class className {     public static void main(String args[] )     {     } }</pre>	

### Tokens

- The smallest individual units in a program are known as tokens.
- Java program is a collection of tokens, comments and whitespaces.
- There are 5 types of tokens
  - Reserved Keywords
  - Identifiers
  - Literals
  - Operators
  - Separators

### Whitespaces

- Java is a free-form language ie., no need to follow any special indentation rules. (eg) any Java program can also be written in a single line if there was at least one whitespace character between each of them. White space is a space tab or a newline.

(eg)

```
classMainClass { public static void main(String args[ ]) { System. ...
```

Comments

- |                 |                         |                       |
|-----------------|-------------------------|-----------------------|
| i) //           | → a single line comment | } similar to C or C++ |
| ii) /* ... */   | → a block comment       |                       |
| iii) /** ... */ | → Documentation comment |                       |

It is used to generate the documentation automatically. The 'javadoc' tool is used for this. This tool or utility parses the entire program and finds the /\*\* and \*/ symbols and take the comments inside the block for documentation. It creates a HTML file with these comments and prepares the documentation of the program. This documentation generated can be viewed using a browser such as Netscape Navigator or Internet Explorer.

A documentation comment can also include HTML tags, as well as special tags beginning with @ that are used to document methods and classes in a standard form. The @ is followed by a keyword that defines the purpose of the tag.

(eg)

@author – to define the author of the code

@deprecated – to denote that they should not be used in new applications

@exception – to document exceptions that the code can throw

@param – to describe the parameters for a method

@throws – a synonym for @exception

@version – to describe the current version of the code

@return – to document the value returned from a method

...

Apart from these specified tags, all HTML tags except the header tags can also be used.

Reserved Keywords

- There are around 50 keywords and they cannot be used as identifiers.
- 'const' and 'goto' are reserved but they are not used.

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

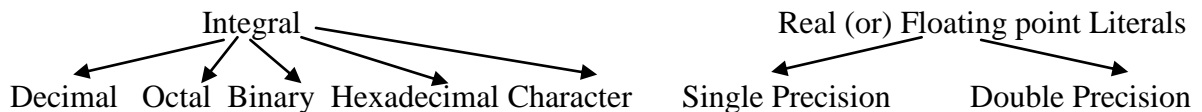
- In addition to these keywords, Java reserves the following.  
true, false, null ➔ Though they appear to be keywords, 'true and false' are Boolean literals & 'null' literal respectively; the values defined by Java.

## Identifiers

- Names given to classes, variables, methods, arrays, etc.
- Case sensitive.
- Uppercase and lowercase letters, numbers and underscore ('\_') are used.
- \$ is also used (not intended for general use).
- The names must begin with an alphabet or an underscore character or a dollar sign. Cannot start with a digit.
- A keyword cannot be used as an identifier name.
- White space and special characters are not allowed.

## Literals or Constants

- These are fixed values directly used in the program and they remain unchanged during the execution of the program.
  - i) Logical literal – (eg) true, false
  - ii) Numeric or arithmetic literal – (eg) 100, 98.6
  - iii) String literal – "test"
  - iv) Null literal - null
- Numeric or Arithmetic literals



- Octal values are denoted in Java by a leading zero – 0
- Hexadecimal values are denoted with a leading 0x or 0X (eg) 0XA → decimal integer 10
- Binary values are denoted with a leading 0b or 0B
  - (eg) int x = 0b1010 → denotes decimal integer 10
- With JDK7, underscore can also be used  
(eg) int x = 12\_789\_603\_540;  
No '\_' should be used at the end of the literal but more than one '\_' is possible between 2 digits.  
(eg) int x = 123\_\_789;  
In Binary, '\_' is useful for visually grouping the values into 4-digit units.  
(eg) int x = 0b1101\_0101\_0001;
- Character Literals  
Visible characters can be directly entered.  
(eg) 'a', '@', '?' ... A character literal represents an integer value of the Unicode.  
Octal and hexadecimal characters can also be directly entered  
(eg) '\141' (octal) → 'a'  
'\u0061' is the ISO-Latin-1 'a'  
Unicode is always preceded by '\u' and has 4 hexadecimal digits.  
For other characters, escape sequences are used.  
\n, \r, \f, \t, \b, ...  
Note:  
Unicode values run from '\u0000' to '\uFFFF' in which '\u0000' to '\u007F' correspond to the 128 ASCII characters.

➤ Floating point Literals

- Decimal values with fractional component.
- All floating point literals default to double precision.
- To explicitly specify a float literal, append 'F' or 'f' to the constant.
- For double also, explicit representation with 'D' or 'd' is possible.
- Hexadecimal floating point literals are also supported.

(eg) 0X12.2P2 → valid floating-point literal. 'P' called the binary exponent indicates the power of 2 by which the number is multiplied.

$$\begin{aligned} \therefore 0X12.2P2 &\rightarrow 12.2 \\ &\quad \begin{array}{l} \rightarrow \times 16^{-1} = 2 \times 16^{-1} = 1/8 = 0.125 \\ \rightarrow \times 16^0 = 2 \times 1 = 2 \\ \rightarrow \times 16^1 = 1 \times 16 = 16 \end{array} \\ &\quad \quad \quad 18.125 \times 2^2 \\ &\quad \quad \quad = 72.5 \end{aligned}$$

- Underscore is allowed.

(eg) float f = 2.3 + 4.7; → error; possible loss of precision  
required : float  
found : double

this is because the floating point literals 2.3 & 4.7 are treated as double literals, by default (not as float).

Therefore, float f = 2.3f + 4.7f; → must be provided.

➤ String Literals

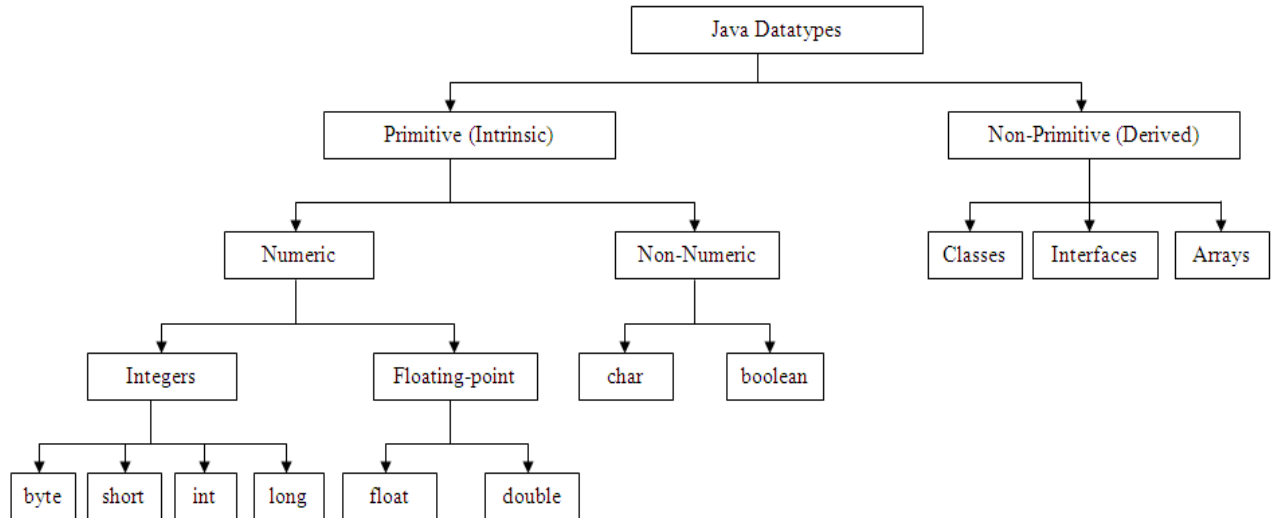
- Sequence of zero or more characters enclosed within double quotes.  
(eg) "Java", " " (null or empty string)

Separators

- Used for grouping and separating.
  - ; → statement terminator
  - , → separates consecutive identifiers
  - . → to separate the class name & method name; package name & class name
  - () → to group the arguments supplied to a method
  - { } → to list values of automatically initialized arrays, to separate a block of code
  - [ ] → to give the array index and to specify the size of an array

Java Datatypes

- Java is a strongly typed language.
- Every type is strictly defined and all assignments (including parameters passing) are checked for type compatibility. There are no automatic conversions of conflicting types as in some languages. Any type mismatches are errors that must be corrected before the compiler will finish compiling the class.



- The 8 primitive data types are not object-oriented since making the primitive types into object would have degraded the performance too much.
- C & C++ allow the size of an int to vary based on the execution environment. But in Java, because of the portability requirements, all data types have a strictly defined range (eg) int – 32 bits (4 bytes).
- Integers
  - No support for unsigned, positive only integers.
  - long – 64 bits
  - int – 32 bits
  - short – 16 bits
  - byte – 8 bits (-128 to 127)
  - byte: variables of this type are useful while working with a stream of data from a network or a file and working with raw binary data.  
(eg) byte a,b;   a& b can store any number between -128 to +127
  - short : short n;
  - int: Always better to use 'int' even to control loops and to index arrays; although short or byte would be sufficient, because when these are used in an expression, they are promoted to 'int', when the expression is evaluated.
  - long: long x;
- float
  - 32 bits – single precision & faster on some processors and takes half as much space as double precision but imprecise, when values are too large or too small.
- double
  - 64 bits – faster than single precision on some modern processors.
  - Optimized for high speed mathematical calculations.
  - Floating-point datatypes (float and double) support special values such as POSITIVE\_INFINITY, NEGATIVE\_INFINITY & NaN (Not a Number).
  - NaN is used to represent the results of operations such as division of zero by zero, where an actual number is not produced.
  - These are static constants defined in both Float and Double wrapper classes.
  - static MIN\_VALUE → gives the smallest positive non-zero value of type double or float

static MAX\_VALUE → gives the largest positive non-zero value of type double or float  
 static POSITIVE\_INFINITY  
 static NEGATIVE\_INFINITY  
 static NaN

(eg)

System.out.println( 0f / 0f ); → o/p: NaN

System.out.println( 1f / 0f ); → Infinity

System.out.println(-1f / 0f ); → -Infinity

'NaN' would result, if we do operations like the following:

System.out.println( 0f / 0f );

System.out.println(Double.POSITIVE\_INFINITY - Double.POSITIVE\_INFINITY);

System.out.println(Double.POSITIVE\_INFINITY / Double.POSITIVE\_INFINITY);

System.out.println( 0 \* Double.POSITIVE\_INFINITY);

...

#### ➤ char

- not like in C/C++, which is 8 bits wide.
- Java uses Unicode to represent character. It can represent all of the characters found in all human languages and hence it is an international character set. So, it requires 16 bits (2 bytes – 0 to 65536). No negative char is allowed. The standard ASCII character set ranges from 0 to 127 and the extended 8-bit char set ISO-Latin-1 ranges from 0 to 255 within the unicode.

For global portability, Unicode is used to represent characters, though not efficient for languages like English, French,... It supports more than 34,000 defined characters derived from 24 languages from America, Asia, etc. The Unicode standard has now been extended to allow up to 1,112,064 characters and these extra characters are called supplementary characters.

- char – can also be used to perform arithmetic operations.

(eg) char ch1 = 'X'; ch1++; => Y

#### ➤ boolean

Boolean variables are used to denote true/false values (logical values).

(eg) boolean b=true, c=false;

System.out.println("b is" + b); → b is true

if(b)

System.out.println("executed"); → executed

if(c)

System.out.println("not executed"); → no o/p since 'c' is false

Note: The outcome of result of a relational operator is a Boolean value.

(eg) System.out.println(10 > 9); → true

Thus, if a loop has to be continued as long as the number, n is not equal to zero, it should be written as

while(n!=0) → this expression produces

{ 'true' result

....

}

while(n) → wrong as '0' does not

{ denote 'false' & other

.... +ve number do not

} denote 'true'

## Variables

- Java variables are classified into 2 types based on the type of data being stored in it.
  - Reference variables (store reference address)
  - Primitive data type variables (store data)
- Based on their scope, they are classified into 3
  - Instance variables – associated with objects and take different values for each object.
  - Class variables – global to a class & belong to all objects of the class. Only one memory location is created for each class variable.
  - Local variables – variables declared & used inside methods
- Dynamic initialization of variables is possible in Java.  
(eg) `double c = (a * a + b * b) – (d * d);`

## Type Conversions:

### i) Widening Primitive Conversion: (Implicit Conversion)

Possible only if the

- a) 2 types are compatible. (eg) double to byte → not possible
- b) Destination type is larger than the source type.

No automatic type conversion from numeric types to boolean.

There are totally 19 widening conversions on primitive types.

- byte to short, int, long, float or double
- short to int, long, float or double
- int to long, float or double
- long to float or double
- float to double

A widening conversion does not lose information.

### ii) Narrowing Primitive Conversions: (Explicit Conversion)

- Done when automatic type conversion is not possible.
- Uses typecasting.

(eg) `int a = 120;`

`byte b = (byte)a;`

`System.out.println(b) → 120`

For any value assigned to a byte variable, modulo division by 256 will be done and the resulting remainder will be assigned to the byte variable.

Therefore,  $120 \% 256 = 120$ .

ii) `int a=130;`

`byte b = (byte)a;`

`System.out.println(b); → -126`

i.e., 130 reduced modulo 256 ( $130 \% 256 = 130$ ). But the last number in the positive side for a byte is +127. After the final limit +127, (130-127) it wraps around to the negative side.

➤ `-128 -127 -126.....-1 0 1 2 3 .....126 127` ─

3 numbers ahead in the negative side is -126.

iii) `byte x=(byte)400;`

*// byte x= 400; will be an error*

`System.out.println(x);`

O/P: -112

( $400 \% 256 = 144$ ). After the final limit +127, it wraps to the -ve side.  $144 - 127 = 17$  numbers ahead in the -ve side

iv) `byte x=(byte)-356; → -100 (since  $-356 \% 256$ )`

v) `byte x=(byte)-400; → 112`

This is because  $-400 \% 256 = -144$ . But the final limit for a byte in the negative side is only -128. Remaining 16 numbers should be wrapped around to the +ve side. This yields 112.



is -112

vi) Truncation

A conversion which occurs when a floating-point value is assigned to an integer type.

(eg) `int i = 25.25;`  $i \rightarrow 25$

`byte b = (byte)323.142;`  $\rightarrow 67$  (because the fractional part is lost and also the value is reduced modulo 256)

vii) Widening and Narrowing Conversion:

byte to char

First, the byte is converted into an int via widening primitive conversion and then the resulting int is converted to a char by narrowing primitive conversion.

(eg) `byte b = (byte)130;`

`char cc = (char)b;`  $\rightarrow$  here 'b' is converted to 'int' by widening conversion; however 'int' has to be converted to 'char' by narrowing conversion.

viii) Automatic type promotion in expressions:

Sometimes in an expression, the precision required of an intermediate value will exceed the range of either operand.

(eg) `byte b=50, a=40, c=100;`

`int d=a*b/c;`

$\rightarrow$  exceeds the range of byte.

For this reason, whenever evaluating an expression, Java automatically promotes each byte, short or char operand to int.

$\therefore a*b$  is performed using int and not bytes & hence it is legal.

But these automatic promotions may sometimes cause compile-time errors.

(eg) `byte b = 50;`

`b = b*2;`  $\rightarrow$  error because when the expression is evaluated, the result has already been promoted to int. So, int result cannot be assigned to a byte without typecast.

`byte b = 50;`

`b = (byte)(b*2);` ✓

Example:

`short oranges=5, apples=10, fruits;`

`fruits=oranges+apples;`

This will not compile because the calculation is carried out using 32-bit Arithmetic (int) & hence will produce a 32-bit result (int), but the variable 'fruits' is only 16-bits long (short).

It should be `fruits=(short) (oranges+apples);`

Example:

Type promotion while invoking the methods.

`classtypepromotion`

```
{
staticvoid display(byte a, byte b) $\rightarrow$ Method1
{
    System.out.println("byte");
}
staticvoid display (char a,char b) $\rightarrow$ Method2
{
```

```

        System.out.println("char");
    }
    static void display (int a, int b) → Method 3
    {
        System.out.println("int");
    }
    public static void main(String args[])
    {
        display(20,30);
        display ((byte)20,(byte)30);
        display ('a','b');
    }
}

```

Output: int  
byte  
char

Suppose display( ) methods 1 and 2 are not defined, still the program will execute, due to automatic type promotion.

class type promotion

```

{
    static void display (int a, int b)
    {
        System.out.println("int");
    }
    public static void main(String[] args)
    {
        display(20,30);
        display ((byte)20,(byte)30);
        display ('a','b');
    }
}

```

Output: int  
int  
int // This is because of byte and char values getting  
// automatically type promoted to int.

## Operators

### Arithmetic Operators:

Same as in C or C++, with the following additional features.

- i) '%' operator can be applied for floating-point types also.  
(eg) double y = 42.25;  
∴ y % 10 = 2.25
- ii) ++ and -- operator can be used for floating-point types also.  
(eg) float a = 3.5f;  
System.out.println(++a); → 4.5

### Bitwise Operators

Operator	Meaning
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
~	Bitwise One's Complement

<<	Left Shift
>>	Right Shift
&=	Bitwise AND and assignment
=	Bitwise OR and assignment
^=	Bitwise XOR and assignment
>>=	Right Shift and assignment
<<=	Left Shift and assignment

The above operators are similar to C.

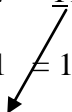
<< - left shift operator shifts the bits of the given number specified number of times to the left.

(eg)  $a = a \ll 2$ ;  $\rightarrow$  shifts the bits of 'a', 2 times, towards the left.

Additionally, Java provides the following 2 bitwise operators.

>>>shift right zero fill (or) unsigned shift right operator

>>>= unsigned shift right assignment

>>	>>>
<p>The operator shifts the bits to the right by the specified number.</p> <p>(eg) int a = 100; System.out.println(a &gt;&gt; 1); O/P: 50</p> <p>Because, (a = 100) in 32-bit binary is 0000 0000 0000 0000 0000 0000 0110 0100</p> <p>If it is shifted to the right by one bit, the following result will be obtained. 0000 0000 0000 0000 0000 0000 0011 0010 = 50</p> <p>But here, the leftmost bit (sign bit) is used for filling the gaps caused by losing the right most bits, due to the shift. ie., after shifting, the higher order bits to be filled, depends on what the sign of the number is.</p> <p>(eg) Let us take the negative number, -2.</p> <p>-2 in binary is got as follows: +2      = 0000 0000 0000 0000 0000 0000 0000 0010 Invert All bits = 1111 1111 1111 1111 1111 1111 1111 1101 Add 1 = _____ +1 -2    1111 1111 1111 1111 1111 1111 1111 1110</p>  <p>-2&gt;&gt;1 = 1111 1111 1111 1111 1111 1111 1111 1111</p> <p>Sign bit. Since the sign bit is 1, the vacancy created after shifting is filled with . = -1 in decimal. ∴ O/P: -1</p> <p>Thus, -1 shifted to the right for several times will only lead to -1</p>	<p>This also shifts the bits to the right by the specified number.</p> <p>(eg) int a = 100; System.out.println(a &gt;&gt;&gt; 1); O/P: 50</p> <p>Always shifts zeros into the higher order bits (which are vacant), regardless of the sign of the number.</p> <p>(eg) -2 &gt;&gt;&gt; 1; -2 in binary is 1111 1111 1111 1111 1111 1111 1111 1110 ↓ Sign bit 0111 1111 1111 1111 1111 1111 1111 1110</p> <p>The vacant positions after shifting are always filled with 0. Sign bit will not be used for filling.</p> <p>= 2147483647</p> <p>Note: -2 &gt;&gt; 1 = -1          -2 &gt;&gt;&gt; 1 = 2147483647</p>

-2 >> 2 = -1	-2 >>> 2 = 1073741823
-2 >> 3 = -1	-2 >>> 31 = 1
...	...

Relational Operators:

==   !=   >   <   >=   <=

Outcome of these operators is the boolean value (unlike in C or C++, where '1' or '0' is returned).

(eg) int a = 4, b = 1;

boolean c = a > b;

System.out.println(c); → O/P: true

If any integer variable has to be used in a loop or branching statement, it must be written as

int n = 65;

if(n == 0)	(or)	if(n != 0)	while(n == 0)	(or)	while(n != 0)
{ ..... }		{ ..... }	{ ..... }		{ ..... }

It cannot be written as while(n) because in java, a positive value does not mean true and zero value does not mean false.

```

{
    ...
}

```

Boolean Logical Operators:

Operates only on boolean operands.

& logical AND      && short circuit AND

| logical OR      || short circuit OR

^ logical XOR      ?: Ternary Operator

!= logical NOT

&= AND Assignment

|= OR Assignment

^= XOR Assignment

= = Equal to

!= Not Equal to

Short-circuit Logical Operators:

||      &&

When these are used, Java will not bother to evaluate the right hand operand when the outcome of the expression can be determined by the left operand alone. It works on the following logic.

- For 'OR' operator, if left operand evaluates to true, the result is also true, regardless of the other operand.
- For 'AND' operator, if left operand evaluates to false, the result will be false.

Thus, '||' and '&&' operators make use of this property and predict the output in most of the cases, by evaluating the first few expressions itself.

(eg) if((10 < 5) || (7 > 3) || (100 > 1000) || (40 == 50))

```

{
    ...
}

```

In this condition,  $(10 < 5)$  will be false. When the 2<sup>nd</sup> expression is evaluated,  $(7 > 3)$ ; it turns out to be true. So, the remaining 2 conditions will not be checked at all and the result of the entire expression will be declared to be true and the 'if' block will be executed.

In contrast, the non-short circuit operators '|' and '&' will evaluate all the sub-expressions to predict the outcome of the entire expression.

To prove that short-circuit operators do not bother about the subsequent expressions, once the result can be predicted with the previous expressions, let us take the following example.

<pre>int a = 10, b = 50; System.out.println(a &gt; 10 &amp;&amp; b/0 &gt; 10); → O/P: false</pre> <p>Here there is no risk of causing a runtime exception when the denominator is zero.</p>	<pre>int a = 10, b = 50; System.out.println(a &gt; 10 &amp; b/0 &gt; 10);</pre> <p>As both the expressions are evaluated, this would cause a runtime exception – → DivisionByZeroException</p>
---	--

### Ternary Operator(?:)

```
intdenom = 0, num = 5;
```

```
ratio = denom == 0 ? 0 : num/denom; → ratio will be assigned 0
```

### instanceof Operator

It returns true when the object on the left hand side is an instance of the class on the right hand side. It is used to determine whether the object belongs to a particular class or not.

(eg) `e instanceof employee` will be true if 'e' object belongs to the class 'employee'; else false.

### Dot Operator:

Used to access the instance variables and methods of class object.

(eg) `e.ename` , `e.salary` , `e.increment()`

### Operator Precedence:

Note: All binary operators except assignment operators follow L → R associativity.

<pre>1) [ ] ( ) . 2) ++(postfix) --(postfix) 3) ++(prefix), --(prefix), ~, !, +(unary), -(unary), (typecast) 4) * / % 5) + - 6) &gt;&gt;, &gt;&gt;&gt;, &lt;&lt; 7) &gt;, &gt;=, &lt;, &lt;= instanceof 8) == != 9) &amp; 10) ^ 11)   12) &amp;&amp; 13)    14) ?: 15) = += -= *= etc</pre>	<pre>R → L R → L R → L R → L R → L R → L R → L R → L R → L R → L R → L R → L R → L R → L R → L</pre>
---	--

### Symbolic Constants:

- i) To modify a constant in the future (Easy, if the modification is done in the single statement in which the symbolic constant has been defined instead of doing the modification in all the places where the numeric values are used).
- ii) To facilitate easy understanding of the program.  
(eg) If the class strength and pass marks are both 50, it will be better to assign to these names, the values at the beginning of the program itself. Use of names will make the program clearer. Once defined, subsequent use of these names in the program cause their defined values to be automatically substituted at the appropriate points.

Syntax:

final type symbolic\_name = value;

- Symbolic constants are defined for types (unlike C, C++)  
(eg) final int MARKS = 50;  
final int CLASS\_STRENGTH = 50;  
final float PI = 3.14;
- It is only a convention to use CAPITAL letters, so as to distinguish them from variable names.
- Assigning any value to it later on, is illegal.  
CLASS\_STRENGTH = 60; → Invalid

### Java Statements:

A statement is an executable combination of tokens ending with a semicolon.

- i) Empty statement – used as a place holder
  - ii) Labeled statement
  - iii) Expression statements – 7 types
    - a) Assignment
    - b) Pre-increment
    - c) Post-increment
    - d) Pre-decrement
    - e) Post-decrement
    - f) Method call
    - g) Allocation expression
  - iv) Selection statement
  - v) Iteration statement
  - vi) Jump statement
- } Control statements
- vii) Synchronization statement → Handling issues with multithreading.
  - viii) Guarding statement → for safe handling of code that may cause exceptions. These statements use the keywords – try, catch, finally.

### Selection Statements:

if, switch

The expression in switch can also be of type String. But using strings in switch is expensive.

If we need to select among a large group of values, a switch statement will run much faster than the equivalent logic coded using a sequence of if-else's. This is because it knows that the case constants are all the same type and simply must be compared for equality with the switch expression. Compiler has no such knowledge of a long list of 'if' expressions.

Example:

Write a program to read the slot name in which the student has registered for Java and display whether his class will be in the morning or afternoon.

```
import java.util.*;
public class classtiming
{
    public static void main(String args[ ])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the Slot:");
        String slot = sc.nextLine( );
        switch(slot)
        {
            case "c1":
                System.out.println("Your class is in morning");
                break;
            case "c2":
                System.out.println("Your class is in the afternoon");
                break;
            default:
                System.out.println("Enter the slot registered for Java");
                break;
        }
    }
}
```

'for' loop:

- i) Simple for
- ii) Enhanced for (or) for-each loop → (see the Arrays Section)

goto:

Although a reserved keyword in Java, it is not supported as it is hard to understand and maintain and also prohibits compiler optimizations.

break:

Provides a 'civilized' form of goto. However, it cannot be used to exit from a deeply nested loop set, unlike 'goto'.

Labeled break:

Can be used to break out of one or more blocks of code.

Syntax:        break label;  
                    ↘ name that identifies a block of code

The labeled block must enclose the break statement, but it need not be the immediately enclosing block.

Example 1:

first:

```
{
```

second:

```
{
    third:
    {
        System.out.println("Before break");
        if(true)
            break second;
        System.out.println("After break");
    }
    System.out.println("Inside 2nd block");
}
System.out.println("Inside 1st block");
}
```

→ It helps to exit from the block named second

The control is transferred here

Output: Before break  
Inside 1<sup>st</sup> block

Example 2:

outer:

```
for(int i=0;i<3;i++)
{
    System.out.println("Pass" + i + ":");
    for(int j=0;j<100;j++)
    {
        if(j == 10)
            break outer;
        System.out.println(j + " ");
    }
    System.out.println("not printed");
}
```

Output:

Pass 0: 0 1 2 3 4 5 6 7 8 9

Labeled continue:

Continue statement may specify a label to describe which enclosing loop to continue.

outer:	Output:
for(int i=0;i<10;i++)	0
{	0 1
for(int j=0;j<10;j++)	0 2 4
{	0 3 6 9
if(j > i)	0 4 8 12 16
{	...
System.out.println();	0 9 18 27 36 ... 81
continue outer;	



```

    }
    System.out.println(" " + (i*j));
    }
    System.out.println();
}

```

→this terminates 'j' loop and continues with the next iteration of 'i' loop.

When i=0:

i) j=0: 'if' block will not be executed. only 'i\*j', ie., 0\*0 is printed.

ii) j=1: 'if' block will not be executed, where a new line is printed and then it continues for the next iteration of 'i' loop, omitting the other values of 'j'.

Same way, it proceeds for other values of 'i'.

#### Some more examples based on labeled break & continue:

1) Write a program to generate the Pythagorean triplets within a given range. Pythagorean triplets are a set of 3 numbers (i, j, k) which follow the relationship

$$i^2 + j^2 = k^2$$

For a particular value of 'i' only a single set has to be generated. Accomplish this using labeled-break and labeled-continue statements.

```

class Pythagorean
{
    public static void main(String args[ ])
    {
        int i, j, k;
        first:
        for(i=1;i<1000;i++)
        {
            second:
            for(j=1;j<1000;j++)
            {
                third:
                for(k=1;k<1000;k++)
                {
                    if(i*i + j*j == k*k)
                    {
                        System.out.println(i + " " + j + " " + k);
                        continue first; → (or) we can use 'break second;' to
                                          accomplish the same task
                    }
                }
            }
        }
    } // end of main()
} // end of class

```

2) Write a Java program to display all the prime numbers within a given range.

```

import java.util.*;
class Prime_break
{
    public static void main(String args[ ])

```

```

{
    int start, end, i, j, flag=0;
    Scanner s = new Scanner(System.in)
    System.out.println("Enter the starting and ending range:");
    start = sc.nextInt( );
    end = sc.nextInt( );
    first:
    for(i=start;i<=end;i++)
    {
        flag=0;
        second:
        for(j=2;j<i/2;j++)
        {
            if(i%j == 0)
                continue first;
            else
                flag=1;
        }
        if(flag == 1)
            System.out.println(i);
    }
}

```

Important:  
 In this scenario, using labeled continue will be better than labeled break. Had we used 'break second' the control would be transferred out of the 'second' block but still the unnecessary 'if' condition (flag == 1) checking will be done.

#### Difference between goto and labeled break:

'goto' is used to transfer the control to any desired location in the code.

'labeled break' is used to transfer the control only out of the enclosing blocks and not from any other block (ie., only in the backward direction and not in the forward direction).

(eg) first:

```

for(i=0;i<10;i++)
{
    ...
    second:
    for(j=0;j<10;j++)
    {
        ...
        break second; → valid
        (or)
        break first; → valid
        //but
        break third; → Invalid because the third block does not enclose this break statement
    }
    third:
    for(k=0;k<10;k++)
    {
        ...
    }
}

```

## Arrays

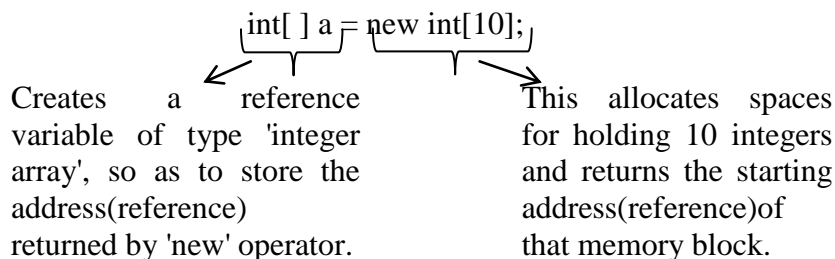
- Collection or group of similar data items, referred to by a common name.
- In Java, arrays are implemented as objects.
- Arrays are immutable (ie., its size cannot be changed once given while declaring it).
- As arrays are implemented as objects, they are also created using 'new' operator, so that they are created during run time (dynamically).

### Syntax:

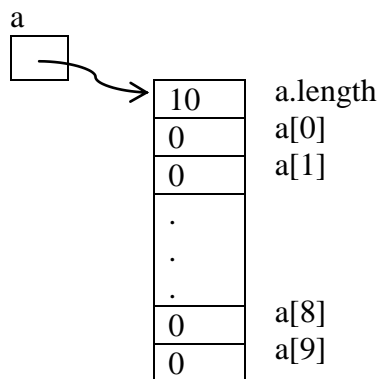
```
type[ ] ref_variable;
ref_variable = new type[size];
(or)
type[ ] ref_variable = new type[size];
```

### Example:

Suppose an integer array has to be created with 10 elements in it, the following statement has to be provided.



### Pictorial Representation:



### Remember:

The reference variable which stores the starting address of this array should also be of same array type.

Though array is an object in Java, there are a few differences between array object and other objects.

- Though all the array elements are basically instance variables in the array object, they are referred to by number, rather than by name.
- No array classes have to be defined explicitly to create array objects.
- The array object also contains 'length' instance variable in addition to the elements of the array. Thus, if an array object of size 10 is created, there will be 11 instance variables in total, including 'length'. 'length' gives the total number of elements that the array is designed to hold and not the number of elements that are actually in use.

Difference between static memory allocation and dynamic memory allocation:

Any variable created using 'new' operator is allocated space in the heap memory unlike other variables which are allocated in stack memory.

Stack	Heap
i) Memory allocated on stack stays in scope as long as it is on the stack. It is destroyed when it is popped off the stack (eg: local variables).	Variables allocated in heap memory are retained until they are specifically de-allocated. [But in Java, Garbage Collection feature takes care of that).
ii) All memory allocated on the stack is known at compile time. Consequently, this memory can be accessed directly through a variable.	As the precise location of the memory allocated is not known until runtime, the memory has to be accessed indirectly through a pointer.
iii) Since the stack is relatively small, large arrays, structures, classes and even heavy recursion → not recommended.	Since heap is a big pool of memory, large arrays, structures or classes can be allocated here.
iv) CPU does efficient use of memory – thus no fragmentation.	Inefficient use.
v) Fast access as it literally uses a single instruction to move the stack pointer down by 10 bytes and thus allocate those bytes for use by a variable.	Slower access.

For the previous example, the array declaration statement can also be written as follows (as in C)

```
int a[ ] = new int[10];
```

However, the former method gives a clear interpretation of the array object creation.

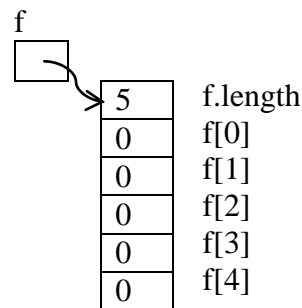
Note:

Thus, in 'public static void main(String[] args)' statement also, 'String[] args' bears better interpretation than 'String args[]'. However, the latter is also valid.

(eg)

2) To create a float array with 5 elements.

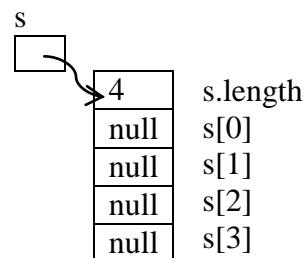
```
float[ ] f = new float[5];
```



(eg)

3) Array of strings with 4 elements.

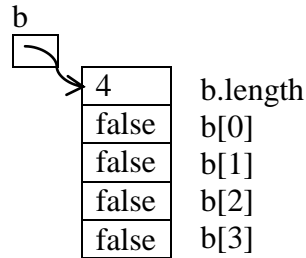
```
String[ ] s = new String[4];
```



(eg)

4) To create a boolean array with 4 elements.

```
boolean[ ] b = new boolean[4];
```



All array elements are auto initialized to zero (for numeric types), false (for boolean) and null (for reference types). Every string in Java is an object; thus the elements of String array are references and hence initialized to 'null'.

Write a program to create a String array of 4 elements and display their default values.

```
String[ ] s = new String[4];
```

```
for(int i=0;i<s.length;i++) → 'length' is an instance variable of the array object 's'.
```

```
System.out.println(s[i]);
```

So, it should be accessed as 's.length'

Output:

null

null

null

null

Note: Array elements are zero, false or null depending on the type, by default. But the default value for all types of reference variables is

- null, if it is a class variable.
- undefined, if it is a local variable in a method.

Example:

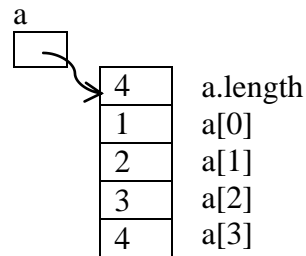
<pre>class Sample {     static byte[ ] b = new byte[2];     static Sample a1;     static int[ ] a;     double f;     public static void main(String args[ ])     {         S.o.p(a1); → null } since, those ref.         S.o.p(a); → null } variables are not                            assigned any values         S.o.p(b); → 1c1729854 (some addr)         S.o.p(new Sample(). f); → 0.0     } }</pre> <p>'f' is a non-static variable. So, should be accessed using an object or reference variable.</p> <p>Here 'a1' and 'a' are reference variables declared as class variables. Hence, initialized to 'null' by default.</p>	<pre>// In case of local variables of a method, no such default initialization is done. class Sample {     public static void main(String args[ ])     {         Sample a1;         int[ ] a;         float f;         S.o.p(a1);         S.o.p(a);         S.o.p(f);     } }</pre> <p>The three println statements above will produce error because the values of 'a1' and 'a' are undefined, when declared inside a method. They are not automatically initialized.</p>
--	---

Array Initialization:

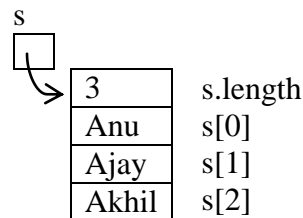
(eg)

`int[ ] a = { 1, 2, 3, 4};`

(or)

`int[ ] a = new int[ ] { 1, 2, 3, 4};`

When the two statements are executed the arrays will be automatically created large enough to hold the number of elements specified in the array initializer.

`String[ ] s = {"Anu", "Ajay", "Akhil"};`

Example:

Write a program to add two integers. Take the inputs as command line arguments. If they are not provided as command line arguments, assign some default values to them.

`class args_length`

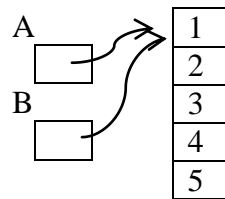
```
{
    public static void main(String[ ] args)
    {
        int a, b, sum;
        if(args.length == 0)
        {
            a = 0;
            b = 0;
        }
        else
        {
            int a = Integer.parseInt(args[0]);
            int b = Integer.parseInt(args[1]);
        }
        sum = a + b;
        System.out.println(sum);
    }
}
```

Copying Arrays:

```
int[ ] a = { 1, 2, 3, 4, 5};
```

```
int[ ] b;
```

```
b = a;
```



This copies the array reference variable 'a' into 'b'. i.e., both the reference variables have the same address which is the starting address of the array. Hence both point to the same array, resulting in only 1 array, being referred to by 2 variables. If you actually want to copy all the values of one array to another, you have to use the `arraycopy()` method, defined in the `System` class.

```
System.arraycopy(fromarray, fromindex, toarray, toindex, count);
```

where,

fromarray – indicates the source array

fromindex – indicates the index in source array from where copying is to be done

toarray – indicates the destination array. It must have sufficient space to hold the copied elements

toindex – indicates the index in destination array from where copying is to be done

count – indicates the number of elements to be copied.

(eg) `int[ ] a = new int[10];`

```
int[ ] b = new int[5];
```

```
for(int i=0;i<10;i++)
```

```
    a[i] = i+1;
```

`System.arraycopy(a, 0, b, 2, 2);` → copies 0<sup>th</sup> & 1<sup>st</sup> elements of `a[ ]` array into the

`for(int i=0;i<5;i++)` 2<sup>nd</sup> & 3<sup>rd</sup> locations of `b[ ]` array since start index

`System.out.println(b[i]);` of `a[ ]` is 0 and start index of `b[ ]` is 2 and number of elements to copy is 2.

Output:

```
0 0 1 2 0
```

→ except these 2 values, all other elements retain their original default values

Example:

Write a program to read an integer array and display the smallest and largest element in the array.

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

```
class minmax_array
```

```
{
```

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

```
    {
```

`int[ ] a=new int[10];` → The size of the array can be got from the user and the array can be constructed accordingly.

```
    int min,max;
```

```
    Scanner sc=new Scanner(System.in);
```

```
    for(int i=1;i<10;i++)
```

```
        a[i]=sc.nextInt( );
```

```
    max=min=a[0];
```

```
    for(int i=1;i<10;i++)
```

```
    {
```

```
        if(a[i]>max)
```

```
            max=a[i];
```

```

        if(a[i]<min)
            min=a[i];
    }
    System.out.println("Minimum is:" + min + " Maximum is:" + max);
}
}

```

#### Enhanced 'for' loop (or) for . . . each loop:

- eliminates the need to establish a loop counter, to specify the starting and ending values and to manually index the array.
- This loop is designed to cycle through a collection of objects like array, in a strictly sequential fashion, from start to finish (not only for arrays but for any collection of objects like vectors...).
- Cannot be substituted for a 'for' loop without collections (example: the 'for' loop used while finding the factorial).
- Syntax:

```

for(type itr_var : Collection)
{
    ...
}

```

→ must be compatible with the elements stored in the collection  
ie., type of the array, if array is being used.

- Example

Program using for-each loop to print all the array elements.

```
int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

```
for(int x : a) → 'x', the iteration variable is of the same type as that of the array
```

```
System.out.println(x);
```

→ O/P: 1 2 3 4 5 6 7 8 9 10

Example:

Find the sum of all the following float array elements using for-each loop.

```
float[] f = {2.4, 3.6, 3.9, 12.8, 10.5};
```

```
float sum = 0; // to assign 0.0, the statement should float sum=0.0f;
```

```
for(float i : f)
```

```
sum += i;
```

```
System.out.println(sum);
```

- A 'for-each' loop is used to iterate through the whole array.  
In the previous example, if the sum of only the first 3 elements has to be found, it can be modified as follows:

```
float[] f = {2.4, 3.6, 3.9, 12.8, 10.5};
```

```
float sum = 0;
```

```
int count=0;
```

```
for(float i : f)
```

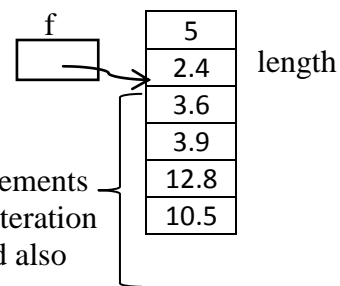
```
{
```

```
sum += i;
```

```
if(++count >= 3)
```

```
break;
```

```
}
```



As the array elements are 'float', the iteration variable should also be a float



```
System.out.println(sum);
```

Example:

The previous 'minmax\_array' program with for-each loop.

```
for(int i=1;i<10;i++)
a[i]=sc.nextInt( );
```

→ Note: Here, for-each loop cannot be used for the reason that a for-each loop processes the values in the array, not the elements (ie., the memory locations).

```
max=min=0;
for(int x : a)
{
    if(x > max)
        max=x;
    if(x < min)
        min=x;
}
```

Invalid:

```
int[ ] a = new int[10];
for(int x : a)
    x = 10;
```

→ The array elements will still have their default values '0'.

The iteration variable is 'read-only', and hence an assignment to the iteration variable has no effect on the underlying array. ie., The array contents cannot be changed by assigning the iteration variable, a new value.

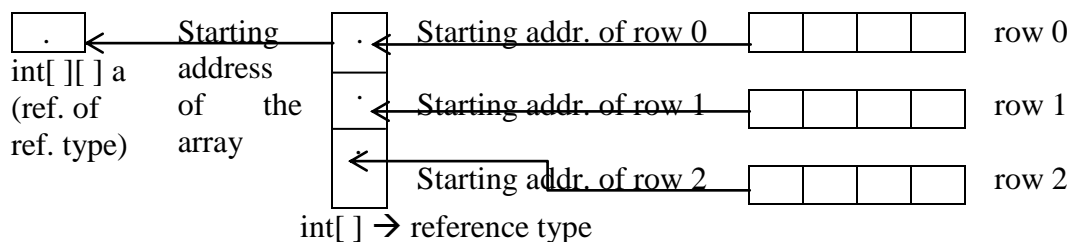
### Multi-Dimensional Arrays:

- Implemented as array of arrays.

For example, a 2-D is treated as an array of several 1-D arrays.

A 3-D array is treated as an array of several 2-D arrays.

(eg) If a 2-D array with 3 rows and 4 columns has to be created, Java does this by creating 3 1-D arrays each with 4 elements in it.



Java creates 3 rows with 4 elements each, using new operator. The starting address of each row, thus created, would be returned by the 'new' operator. So, totally 3 addresses would be returned. Therefore, an array of 3 reference variables is needed to hold these 3 addresses. And to access all the rows and columns through this array of reference variables, first of all, the starting address of this array is needed. So, the starting address of this reference array has to be stored in a variable (reference of reference variable).

→ `int[][] a`

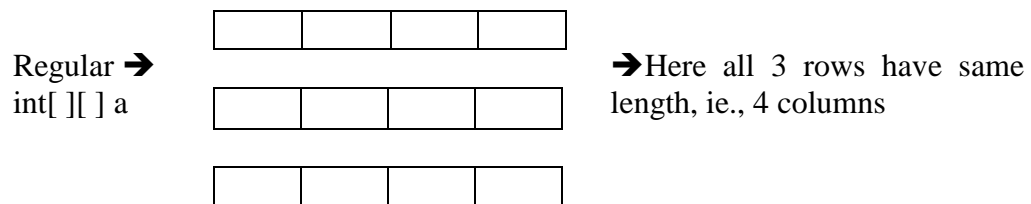
This is what happens internally whenever a 2-D array is created.

Creates a 2-D array with

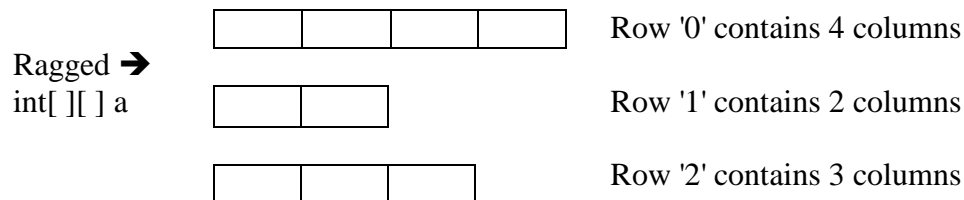
(eg) 1) `int[ ][ ] a = new int[3][4];` → 3 rows and 4 columns  
 2) `float[ ][ ] f = new float[3][3];` → 3 rows and 3 columns  
 3) `int[ ][ ] a = {{1, 2, 3},{4, 5, 6},{7, 8, 9}};`  
 ↓  
 The 2-D array 'a' is created with 3 rows and 3 columns.  
 Row '0' elements – 1, 2, 3    Row '1' – 4, 5, 6    Row '2' – 7, 8, 9

Multi-dimensional arrays are of 2 types

- Regular Arrays
  - For a N-dimensional array, all (N-1) dimensional arrays are of same length.
  - Example



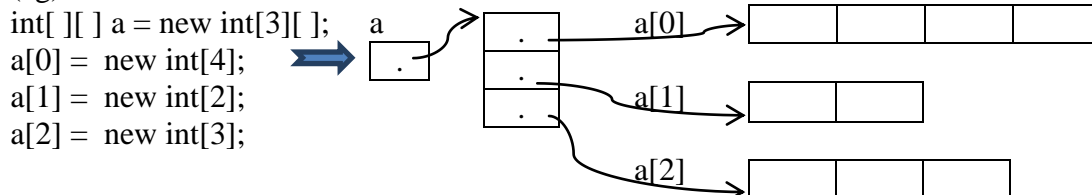
- Ragged Arrays (or Jagged array. But 'Jagged array' is the term used commonly in languages like C#)
  - They are of varying length.
  - Example



### How to create a Ragged Array?

When memory is allocated for a multi-dimensional array, only the first dimension (left most) must be specified compulsorily. Remaining elements can be allocated separately.

(eg)



It is inconvenient to declare the individual rows explicitly for large number of rows. It can also be written as follows:

```
int[ ][ ] a = new int[3][ ]; → statement A
for(int i=0; i<a.length; i++)
{
  // ...
}
```

or 3

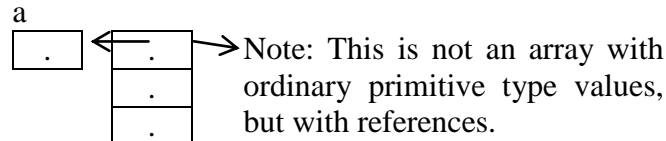
```

        System.out.println("Enter the number of columns for row " + i);
        int x = sc.nextInt();
        a[i] = new int[x]; → statement B
    }

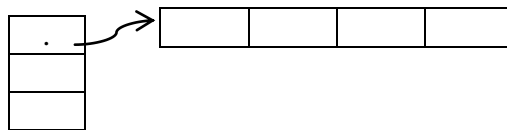
```

→ When user gives 4, 2, 3 value for each iteration, a 2-D array with 3 rows – one with 4 col, another with 2 col and last with 3 col will be created.

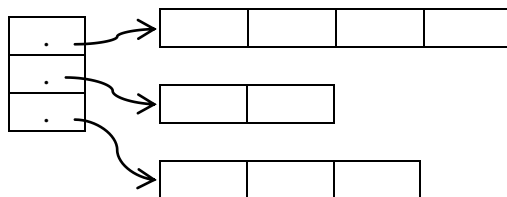
The statement marked A creates an array of 3 reference variables and its starting address is assigned to 'a'.



Now that we have created 3 reference variables, each one can store a reference or address. Statement B, when executed, creates an array (row) of 'x' elements and its starting address is stored in 0<sup>th</sup> element of the array of reference variables.



When statement B is repeated 3 times inside the loop, 3 arrays with 'x' columns will be created & their addresses stored appropriately inside the elements of the array of references.



Example:

Write a program to find the sum of the elements of a matrix.

```

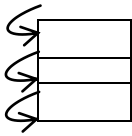
int[ ][ ] matrix = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
int sum=0;
for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        sum += matrix[i][j];
    }
}
System.out.println(sum);

```

for-each loop for the above program:

The outer loop is for traversing the rows.

Inner loop is for iterating through the columns.



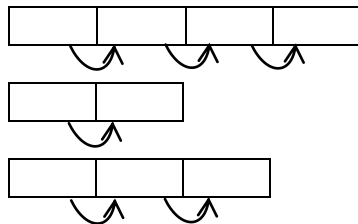
Outer loop should process the reference array which moves down the rows, starting from row 0 to row 2.

So, the iteration variable should be of the same type as that of the elements in this array (ie., `int[ ]`)

→ Reference type

`for(int[ ] x : matrix)` → outer loop

The inner loop should process the 3 integer arrays.



So, iteration variable should be of type integer, same as that of the elements in integer array.

→ 'x' gives the row to be traversed  
`for(int y : x)` → inner loop

Thus, the above code can be written as

```
for(int[ ] x : matrix)
{
    for(int y : x)
    {
        sum += y;
    }
}
```

Remember:

- Enhanced for loop cannot be used to move through an array in reverse.
- Cannot be used to add/change array elements.
- Cannot be used to track the index position of the current element in an array.
- Cannot be used to access any element other than the current element on each pass.
- A single for-each cannot be used to traverse 2 arrays.

## Math Class

- It is a predefined class in 'lang' package.
- All the methods are static methods. So, the class name has to be used to access it.  
`Math.log(one argument)` → natural logarithm of the given value  
`Math.sqrt(one argument)`  
`Math.pow(arg1, arg2)` →  $\text{pow}(x, y) \rightarrow x^y$

`Math.min(arg1, arg2)` → gives the smaller of 2 values of any type (numeric)  
`Math.max(arg1, arg2)` → gives the greater of 2 values of any numeric type  
`Math.floor(arg1)` → `floor(7.2) = 7` ; `floor(7.8) = 7`  
`Math.ceil(arg1)` → `ceil(7.2) = 8` ; `ceil(7.8) = 8`  
`Math.round(arg1)` → `round(7.2) = 7` ; `round(7.8) = 8`  
`Math.abs(arg1)` → `abs(-7) = 7` ; `abs(7) = 7`  
`Math.exp(arg1)` → `exp(x) = ex` ie., returns e raised to the power of 'x'  
`Math.sin(arg1)`  
`Math.cos(arg1)`  
 ....  
`Math.asin(arg1)` → returns arc sine of an angle  
`Math.acos(arg1)`  
 ....  
`Math.toRadians(double deg)` → converts degrees to radians  
`Math.toDegrees(double rad)` → converts radians to degrees  
`Math.random()` → returns a random number between 0 to 1 [0 , 1) → including 0 but less than 1)

1) Write a program to check whether the given number is a power of 2 or not.

i) By using the methods in Math class

ii) By using bit-wise operator (&) [Note: if 'n' is a power of 2,  $n \& (-n) = n$ ]

i) Let 'x' be the number to be checked. If it is a power of 2, then 'x' will be equal to  $2^n$ .

$$x = 2^n \Rightarrow \log x = \log 2^n$$

$$\log x = n \log 2 \text{ (Since, } \log a^x = x \log a \text{)}$$

$$\log x / \log 2 = n$$

ie., If  $(\log x / \log 2)$  gives a whole number, then 'x' is a power of 2.

```

int x = sc.nextInt();
int i; double d;
d=Math.log(x) / Math.log(2); → Since they are static methods, use the class
i = (int) (Math.floor(d));          name also
if(d - i == 0)
    System.out.println("The number is a power of 2");
else
    System.out.println("The number is not a power of 2");
  
```

ii) If 'n' is a power of 2, only 1 bit will be set.

(eg) 1 = 0 0 0 1	-1 = 1 1 1 1	∴ 1 & -1 = 0 0 0 1 = 1
2 = 0 0 1 0	-2 = 1 1 1 0	∴ 2 & -2 = 0 0 1 0 = 2
4 = 0 1 0 0	-4 = 1 1 0 0	∴ 4 & -4 = 0 1 0 0 = 4

...

Code snippet:

```

if((x & (-x)) == x)
    System.out.println("The number is a power of 2");
else
    System.out.println("The number is not a power of 2");
  
```

Some more examples:

1) Write a program to find the radius of a circle in feet and inches, given its area in square feet.

```
import java.util.*;
class circleradius
{
    public static void main(String[ ] args)
    {
        Scanner sc = new Scanner(System.in);
        double area = sc.nextDouble( );
        double rad = Math.sqrt(area/Math.PI);
        int feet = (int)Math.floor(rad);
        int inches = (int)Math.round(12.0 * (rad-feet));
        System.out.println("The radius is " + feet + " feet " + inches + " inches");
    }
}
```

Area of circle =  $\pi r^2$ . ' $\pi$ ' Value is a constant defined in Math class [PI]. Similarly Math.E is also a constant.

$$\therefore r = \sqrt{\text{area}/\text{Math.PI}}$$

If the answer is (say) 73.84, its integer part '73' is the feet value and the fractional part (84) should be converted to inches. To convert from feet to inches, the formula 12 inches = 1 foot can be used.

As per the program, 'rad' will be 73.84

'feet' will be 73

So, inches has to be calculated as (73.84 – 73) \* 12

2) Write a program to randomly generate 'n' number of uppercase characters. Count the number of vowels generated totally and the number of consonants. Read the number of characters as command line input.

```
class random_class
{
    public static void main(String[ ] args)
    {
        int count=0;
        int n=Integer.parseInt(args[0]);
        for(int i=0;i<n;i++)
        {
            int r=65+(int)(Math.random( )*(90-65+1));
            System.out.println(r);
            switch((char)r)
            {
                case 'A':
                case 'E':
                case 'I':
                case 'O':
                case 'U': count++;
                    break;
                default: break;
            }
        }
    }
}
```

```

    }
    System.out.println("nr of vowels"+count);
    System.out.println("consonants" +(n-count));
}
}

```

For our problem, we require 26 different random numbers (for all the uppercase letters) from 65 to 90. But `Math.random()` generates only a number between 0 and 1. So, to get a specified range of values, first multiply by the magnitude of the range of values to be covered. `Math.random() * (90-65)`. → This returns a value in the range  $[0, 90-65) = [0, 25)$ . Since, this still does not include 25, we add one to the range parameter (90-65).

Now, `Math.random() * (90-65+1)` will generate numbers from 0 to 25, including 25, making it possible for the generation of 26 different random numbers.

However, we want numbers from 65 to 90 instead of from 0 to 26. So, the starting number should be shifted from 0 to 65. This can be done by adding 65 to it.

`65+(Math.random() * (90-65+1))`. This will give a random value in the range  $[65-90]$ . As the value returned is double, truncate the decimal part by casting to an int.

→ `65+(int)(Math.random() * (90-65 + 1))`

i.e., To generate a random number within a specified range, use the following formula

`lowerlimit+Math.random()*(upper-lower+1)`

3) Write a program to define a static method `sum_Squares()` to find the sum of the squares of the first 'n' natural numbers and a non-static method `square_Sum()` to find the square of the sum of those 'n' numbers. Invoke these methods from `main()` method to evaluate the difference between the values returned by them.

ie,  $(1+2+3\dots+n)^2 \rightarrow \text{square\_Sum}()$   
 $(1^2+2^2+3^2\dots n^2) \rightarrow \text{sum\_Squares}()$

class `sum_squareclass`

```

{
    public static void main(String[] args)
    {
        int n=Integer.parseInt(args[0]);
        System.out.println(sum_squares(n));
        System.out.println(new sum_squareclass().square_sum(n));
    }
    static long sum_squares(int n) → no need of function prototypes in the beginning as in
    {                                     'C' language
        long sum=0;
        int i;
        for(i=1;i<=n;i++)
        {
            sum+=i*i;
        }
        return sum;
    }
    long square_sum(int n)
    {
        long sum=0;

```

```

        int i;
        for(i=1;i<=n;i++)
        {
            sum+=i;
        }
        return sum*sum;
    }
}

```

Output:

Z:\>java sum\_squareclass 3

O/P: 14

36

The difference of these 2 values can be printed

4) Write a recursive function recursiveMin that takes an integer array, a starting subscript and an ending subscript as arguments, and returns the smallest element in the array. The function should stop processing and return when starting subscript equals ending subscript.

class recursiveMinClass

```

{
    public static void main(String[ ] args)
    {
        int[ ] a = {32, 3, 47, 91, 32, 68};
        System.out.println(recursiveMin(a, 0, a.length-1); → starting index = 0
                                                                ending index = 5
    }
    static int recursiveMin(int[ ] a, int start, int end)
    {
        if(start == end)
            return a[start];
        else
        {
            int mid = (start + end) / 2;
            int min1 = recursiveMin(a, start, mid); → stmt A
            int min2 = recursiveMin(a, mid+1, end); → stmt B
            if(min1 < min2)
                return min1;
            else
                return min2;
        }
    }
}

```

Output: 3

Let us make a stack representation of the sequence of method calls to understand the recursive procedures. Follow three steps.

- i) Whenever there is a function call, the parameters (a, start, end) are pushed into the stack.
- ii) When a function returns a value, replace the function call in the stack, with that value.



- iii) If only one integer value has been returned to the stack it means that we have to process the second half of the array (statement **B**).
- iv) If 2 values have been returned and pushed into the stack, it means that both **A** & **B** statements have been executed, for that particular recursive call and hence the control will move to the next line, finding the minimum of those 2 values and returning it.  
 $\therefore$  If there are 2 values in the stack, return the minimum of 2 values (by popping them and pushing the minimum)  
 (eg) First call to the method is in main( )

recursiveMin(a, 0, 5);

$\therefore$  start = 0

end = 5

a, 0, 5

$0 \neq 5$  So, else part executed  $\text{mid} = (0+5)/2 = 2$

In the next line **A**, again there is a method call by passing (a, 0, 2)  $\rightarrow$  (So, push it)

a, 0, 2
a, 0, 5

mid=2

$0(\text{start}) \neq 2(\text{end})$

$\therefore$  Else Part:  $\text{mid} = (0+2)/2 = 1$

Method call by passing a, 0, 1  $\rightarrow$  push it

a, 0, 1
a, 0, 2
a, 0, 5

mid=1

mid=2

Now also  $0(\text{start}) \neq 1(\text{end})$

$\therefore$  Else Part:  $\text{mid} = (0+1)/2 = 0$

Again method call by passing a, 0, 1  $\rightarrow$  push it

a, 0, 0
a, 0, 1
a, 0, 2
a, 0, 5

mid=0

mid=1

mid=2

This time, start (0) = end (0)

$\therefore$  In the 'if' part,  $a[\text{start}] = a[0] = 32$  will be returned to the place, where the method was called.

This was the place where method was called by passing (a, 0, 0)

32
a, 0, 1
a, 0, 2
a, 0, 5

mid=0

mid=1

mid=2

As per the code, the left half of the sub array, of a particular recursive call is processed completely. So, statement **B** has to be executed.

So, method call must be done by passing (a, mid+1, end)

Looking at the stack, the last mid value was '0'.

$\therefore$  Method call by passing (a, 1, 1)

a, 1, 1
32
a, 0, 1
a, 0, 2
a, 0, 5

mid=0

mid=1

mid=2

Notice the 'end' parameter in the recent method call at the top of the stack is 1

start (1) = end (1)  $\therefore$  return a[1] = 3

Now that there are 2 values returned by left side and right side of the array.

Minimum of these two which is '3' is to be returned to the place where the method was last called. (ie., in the place where (a, 0, 1) is there).

3 (min2)	
32 (min1)	
a, 0, 1	mid=0
a, 0, 2	mid=1
a, 0, 5	mid=2

→

3
a, 0, 2
a, 0, 5

Looking at the stack, as there is only 1 value, we are yet to process the right half of the sub array, (stmt **B**). So, call the method by passing (a, mid+1, end). Recent 'mid' value is 1 and recent 'end' value is '2'. So, method call by passing (a, 2, 2)

a, 2, 2	
3	
a, 0, 2	mid=1
a, 0, 5	mid=2

start (2) = end (2). So, return a[2]

As there are 2 values, the control can now move to the next line of the code, to find the minimum of these 2 and return it to the place where it was called.

47	
3	
a, 0, 2	mid=1
a, 0, 5	mid=2

→

mid=2	
	3
	a, 0, 5

Next call by passing (a, mid+1, end) = (a, 3, 5)

→	
	a, 3, 5
	3
	a, 0, 5

→ start≠end  
 $\therefore$  mid=(3+5)/2=4  
Method call:

mid=2

a, 3, 4	mid=4
a, 3, 5	
3	
a, 0, 5	mid=2

→

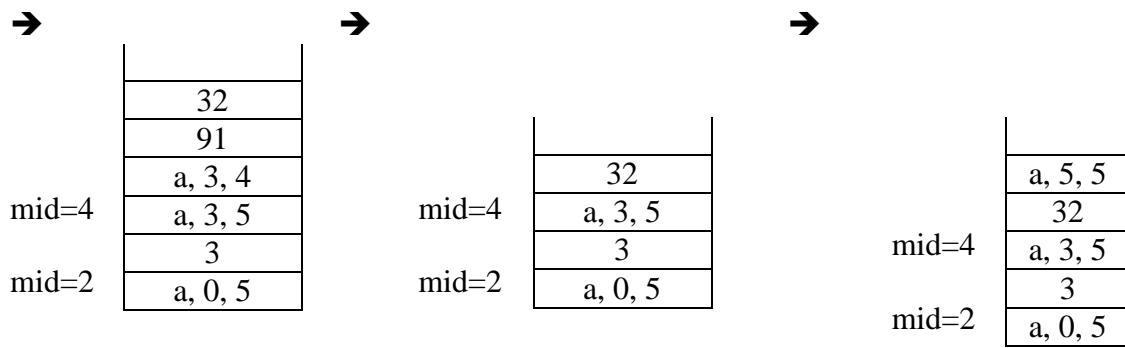
mid=3	
mid=4	a, 3, 3
	a, 3, 4
	a, 3, 5
	3
mid=2	a, 0, 5

→

mid=3	
mid=4	91
	a, 3, 4
	a, 3, 5
	3
mid=2	a, 0, 5

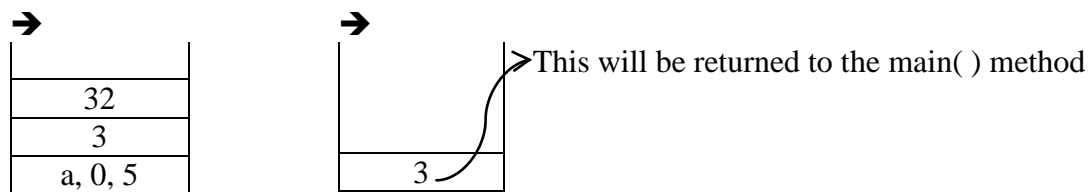
→

mid=3	
mid=4	a, 4, 4
	91
	a, 3, 4
	a, 3, 5
	3
mid=2	a, 0, 5



∴ 68 will be returned and minimum of

68 & 32 which is 32 will be returned



## Classes

Class:

Defines a new data type which is used to create objects of that type. It is a template for an object and is a logical construct which defines the relationship between its members.

Object:

It is an instance of a class (similar to a variable declared for a data type, object is declared for a class type). It has physical reality (ie., an object occupies space in memory).

Syntax for class declaration:

class className

```
{
type instance_variable1;
type instance_variable2;
...
typeinstance_variableN;
type method1(parameter_list)
```

Data or variables defined within a class are instance variables. However, static variables are called class variables.

```
{
//body of the method → code is contained within methods
}
```

```
...
typemethodN(parameter_list)
```

```
{
//body of the method
}
```

} → class definition ends here

The methods and variables defined within a class are called members of the class. This way of binding the data & the methods accessing them, together is called encapsulation.

### Example:

Let us create a class named 'Computer' which should contain the following as its instance variables.

- RAM size, hard disk capacity, frequency, make (company name).

Also, it should have 2 methods:

- i) getdata() – to read the values for the instance variables
- ii) display() – to display the values of the instance variables

### Class Definition:

class Computer

```
{
int RAM;
int hdisk;
float freq;
String make;
void getdata( )
{
    Scanner sc = new Scanner(System.in);
    RAM = sc.nextInt( );
    hdisk = sc.nextInt( );
    freq = sc.nextFloat( );
    make = sc.nextLine( );
}
void display( )
{
    System.out.println("RAM Size:" + RAM);
    System.out.println("Hdisk Capacity:" + hdisk);
    System.out.println("Frequency:" + freq);
    System.out.println("Brand:" + make);
}
}
```

(or) we can hard code these values in the program instead of getting input from the user

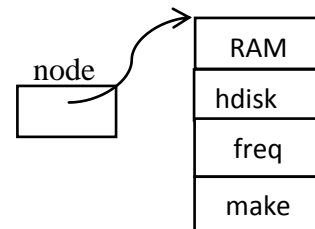
Now to work on a variable, which would have the same instance variables and methods as that of the class 'Computer', an object of this class type should be created.

As Java follows dynamic memory allocation, 'new' operator should be used, for creating objects.

Computer node = new Computer( );

'new' operator does 3 things

- i) Allocates memory space for the object of type 'Computer'.
- ii) Invokes the constructor of the class (will be discussed later).
- iii) Returns the starting Address (Reference) of the object, thus created.



In order to store this reference, a reference variable of same type 'Computer' should be created.

Here, node is just a reference variable which refers to the actual object. It is not an object, by itself.

The above statement can also be written as

Computer node; → After this line executes, 'node' contains the value 'null',  
 node = new Computer( ); indicating that it does not yet point to an actual object.

Each instance (object) of the class contains its own copy of these variables. So, data for one object is separate & unique from data for another.

Computer node1 = new Computer( );



Computer node2 = new Computer( );



All the methods inside the class 'Computer' – [getdata( ) & display( )] can access the instance variables RAM, hdisk, . . . directly. Apart from these 2 methods, if these variables are tried for access from any other method, it can be done only with the help of an object. No direct access is allowed from outside.

To write code which can access this class members, the main( ) method should be defined, where object for this class can be declared and used for accessing the class members.

Note: Though the object reference is similar to a pointer in C or C++, we cannot make this object reference to point to an arbitrary memory location (or) manipulate it like an integer.

class ComputerMain → The main( ) method should not be written independently. It should { be enclosed within a class & this class is generally called the main public static void main(Strin[ ] args) class as the execution starts from here.

```
{
  Computer node1 = new Computer( );    → Statement A
  Computer node2 = new Computer( );    → Statement B
  node1.getdata( );                    → Statement C
  node2.getdata( );                    → Statement D
  node1.display( );                    → Statement E
  node2.display( );                    → Statement F
}
```

The source file should be names as 'ComputerMain.java'. It should not be named after the class 'Computer', as 'Computer' is not the main class.

But still, when the program is compiled, two .class files will be created, one for each class.

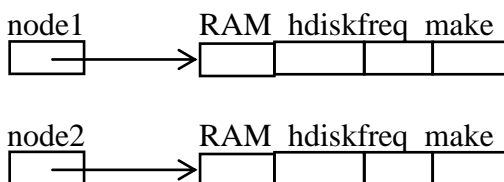
→Computer.class

→ComputerMain.class

The two class definitions can also be stored in 2 different source files.

Statements A & B:

Allocates space for 2 objects of type 'Computer' and stores their references in 'node1' & 'node2' variables.



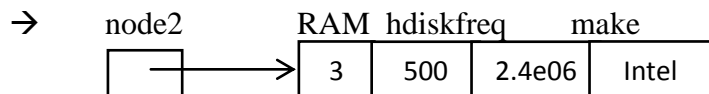
Statements C & D:

To get the input values for these objects, `getdata( )` method should be called. But, as it is a member of the class 'Computer' and has to be called from another class 'ComputerMain', it cannot be called directly. Object of the class 'Computer' should be used to access it. In `getdata( )` method, the values provided by the user are assigned to RAM, hdisk, freq & make. The method accesses the instance variables of the class directly, without reference to an object & without dot operator. A method is always invoked relative to some object of its class. Once this invocation has occurred, the object is known.

There are 2 copies of all the instance variables, one for `node1` & another for `node2`. However, as, in statement C, the method is invoked using `node1`, the input values obtained in the first method call, get stored in the instance variables of the object referred to by 'node1'.



Similarly `node2.getdata( );` makes the input values to be stored in the instance variables of the object referred to by 'node2'.



If the values of object referred to by `node1` have to be displayed, invoke the `display( )` method using `node1`. So, inside the `display( )` method, RAM, hdisk, freq & make refer to RAM, hdisk, freq & make of node1.

To display the values of `node2`,

`node2.display( );`

Output:

```
2
400
1.2e6 (in std notation)
AMD
3
500
2.4e06
Intel
```

In the above program, the `main( )` method, can access the instance variables also just like accessing the class methods.

This is because, all the instance variables & methods have not been specified any access mode. Thus they enjoy default access mode, where in, they can be accessed by any other class, in the same package.

(eg) `node2.hdisk = 100;`  $\rightarrow$  is possible from main class.

To protect the data from outside access or from accidental misuse, it is better to make the instance variables 'private'.

Private members can only be accessed by the class methods, `getdata( )` & `display( )`.

`main( )` method which is outside the 'Computer' class will not be able to access the private members.

```

class Computer
{
    private int RAM;
    private int hdisk;
    private float freq;
    private String make;
    public void getdata( )
    {
        ...
    }
    public void display( )
    {
        ...
    }
}
class ComputerMain
{
    public static void main(String args[ ])
    {
        Computer node1 = new Computer( );
        Computer node2 = new Computer( );
        node1.getdata( );
        node1.RAM = 1; → is not possible
        node2.getdata( );
        node1.display( );
        node2.display( );
    }
}

```

As the data members are declared 'private', they can only be accessed only within this class.

Methods should generally be defined with 'public' or 'default' access mode, so that they act as an interface to the class, from the outside world.

'public' members of a class can be accessed by any other code.

'protected' access mode will be discussed in 'Inheritance' concept.

### Methods taking Parameters:

Parameter : A variable defined by a method that receives a value when the method is called (eg) in 'int square(int i)', 'i' is a parameter.

Argument: The value passed to a method when it is invoked (eg) square(100) passes 100 as an argument.

The getdata( ) method can be modified in such a way that it takes parameters. The input values can be read in the main( ) method and passed to getdata( ).

```

class Computer
{
    ...
    public void getdata(int r, int h, float f, String s)
    {
        RAM = r;
        hdisk = h;
        freq = f;
        make = s;
    }
}

```

→ parameters

```

    }
    ...
}
class ComputerMain
{
    public static void main(String args[ ])
    {
        Computer node1 = new Computer( );
        Computer node2 = new Computer( );
        node1.getdata(1, 200, 1.2e06, "Intel"); → The values received by the parameters in the
        Scanner sc = new Scanner(System.in);      getdata( ) method & eventually assigned to
        int r = sc.nextInt( );                      the instance variables.
        int h = sc.nextInt( );
        float f = sc.nextFloat( );
        String s = sc.next( );
        node2.getdata(r, h, f, s);
        node1.display( );
        node2.display( );
    }
}

```

Constructors:

Since all the instance variables of all the objects have to be initialized, it will be convenient, if this initialization is done at the time, the object is first created. This automatic initialization is performed through the use of a constructor.

Constructor:

- Initializes an object immediately upon creation.
- Has the same name as the class in which it resides & syntactically similar to a method.
- Once defined, it is automatically called immediately after the object is created, before the 'new' operator completes.
- Constructor should not have any return type (not even void) as the implicit return type of a class constructor is the class type itself.
- If a constructor is not defined for the class explicitly, Java creates a default constructor for the class, which initializes all the instance variables to their default values (0, false, null for numeric, boolean and reference variables respectively).

Thus, in the previous example, when the objects are being created using 'new', default constructor supplied by Java, is invoked. This makes RAM = 0, hdisk = 0, freq = 0, & make = null. When the getdata( ) method is invoked later on, it changes the values of these instance variables.

Let us modify the previous program by including a constructor for assigning values to the instance variables.

Instead of getdata( ) method

```

Computer( )
{
    RAM = 2;           //It is a constructor as it takes the name of the class. It need
    hdisk = 200;        not be called explicitly. Once the object is created, it will
    freq = 1.2e06;      be automatically called.
}

```



```
        make = "IBM";
    }
    Computer node1 = new Computer( ); ➔ calls the constructor Computer( )
                                automatically
```

Whereas, if `getdata( )` method is defined for initializing, it must be explicitly invoked, after creating the object.

`node1.getdata( ); ➔` only then, the method is invoked.

### Parameterized Constructor:

- The constructor defined above always initializes all the objects with the same values mentioned inside it.

So, `node1` will also have `{2, 200, 1.2e06, "IBM"}`. `node2` will also have `{2, 200, 1.2e06, "IBM"}`.

To supply the values which the user wants, a parameterized constructor can be written, instead of the default constructor. (Or, Input values can be read from the user within the default constructor and assigned to the instance variables).

`Computer(int r, int h, float f, String s)`

```
{
    RAM =r;
    hdisk = h;
    freq = f;
    make = s;
```

`}` ➔ such a parameterized constructor will be invoked only when the object is created as follows:

```
    Computer node1 = new Computer(2, 200, 1.2e06, "IBM");
```

(or)

```
    Computer node1 = new Computer(r, h, f, s);
```

Where `r`, `h`, `f`, & `s` variables are assigned with some constant values or obtained as input from the user.

Once the parameterized constructor is defined, no object can be created by simply writing a statement like the one below.

```
    Computer node3 = new Computer( ); ✕ This will cause an error, as there is no
    matching constructor without any parameters (default constructor).
```

**It is important to remember that the default constructor will be supplied by Java only if there is no constructor defined in the class. Once the user starts to define any constructor, he must also take care of defining the default constructor by himself, to handle the situation mentioned above.**

### Overloading Constructors:

#### What is Overloading?

The name of the method together with the type and sequence of parameters form the signature of the method.

Two or more methods within the same class can share same name as long as their parameter declarations are different (ie., signature is different. Those methods are said to be overloaded and the process is referred to as method overloading.

(eg) `void method1(int x) { . . . } ➔ A`

`void method1(int x, int y) { . . . } ➔ B`

```
void method1(float x, float y) { . . . } → C
```

All the above 3 methods have the same name 'method1'. However, they differ by the number of parameters, it receives (statements A & B) or they differ at least by the type of the arguments (B & C though they have 2 arguments, 'B' has integer parameters and 'C' has float parameters).

When an overloaded method is invoked, Java uses the type &/ number of arguments to determine which version of the overloaded method to actually call.

(eg) If there is a call like method1(2, 3);, method1 in statement 'B' will be invoked.

If there is a call like method1(5);, method1 in statement 'A' will be invoked.

method1(2, 3.5); → method1 in statement 'C' will be invoked. Though the 1<sup>st</sup> argument is an integer, it will be type promoted to float, to match the method which can take float. Even though the overloaded methods may have different return types, the return types alone are insufficient to distinguish 2 versions of a method.

```
(eg) void method1(int x, int y) { . . . }
      int method1(int x, int y) { . . . } } cannot be overloaded as they differ only by
                                         their return types.
```

A constructor can also be overloaded as we overload other methods provided their number of arguments or type of arguments differs.

Let us rewrite the previous program by overloading the constructor. Let us also try to overload the method – 'calculateSpeed()'.

```
class Computer
```

```
{
```

```
private int RAM, hdisk
```

```
private float freq;
```

```
private String make;
```

```
Computer() → will be invoked whenever an object is created as
```

```
{ Computer node1 = new Computer();
```

```
    RAM = 2;
```

```
    hdisk = 200;
```

```
    freq = 1.2e06;
```

```
    make = "Intel";
```

```
}
```

```
Computer(int r, int h, float f, String s)
```

```
{
```

```
    RAM = r;
```

```
    hdisk = h;
```

```
    freq = f;
```

```
    make = s;
```

```
} → will be invoked when object is created as Computer c = new Computer(3, 100,
    //2.4e06, "IBM");
```

```
Computer(int r, int h, float f)
```

```
{
```

```
    RAM = r;
```

```
    hdisk = h;
```

```
    freq = f;
```

```
    make = "AMD";
```

```
} → will be invoked when object is created as
```

```
    Computer c = new Computer(3, 100, 2.4e06); All objects created by invoking
    this constructor, assigns "AMD" for the 'make' instance variable.
```

```

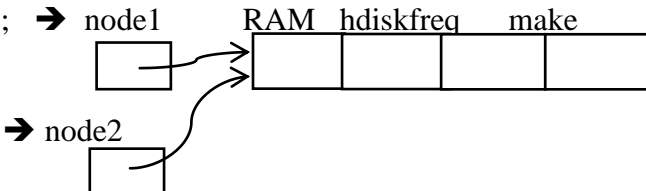
void calculateSpeed( ) //will be invoked for a single core processor
{
    System.out.println("The speed is: " + freq + " cycles per second");
}
void calculateSpeed(Boolean ismulticore) //will be invoked for a multicore processors
{
    System.out.println("Enter the number of cores:");
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt( );
    System.out.println("The speed is: " + freq*n + " cycles per second");
}
void display( )
{
    System.out.println(RAM + " " + hdisk + " " + freq + " " + make);
}
}
Class ComputerMain
{
    public static void main(String args[ ])
    {
        Computer node1 = new Computer( ); → invokes default constructor
        Computer node2 = new Computer(4,400,2.4e06,"Intel");→calls constructor with 4 args
        Computer node3 = new Computer(4, 400, 2.4e06); → calls constructor with 3 args
        System.out.println("Is node2 a multicore processor?");
        Scanner sc = new Scanner(System.in);
        boolean ismulticore = sc.nextBoolean( );
        if(ismulticore) → only if the user gives 'true' if block will be executed.
            node2.calculateSpeed(true); //invokes method that takes a boolean argument
        else
            node2.calculateSpeed( ); //invokes method that doesn't take any argument
        node1.display( );
        node2.display( );
        node3.display( );
    }
}

```

#### Assigning one object reference to another:

When one object reference variable is assigned to another object reference variable, a copy of the object is not made instead only a copy of the reference variable is made.

(eg) Computer node1 = new Computer( ); → node1



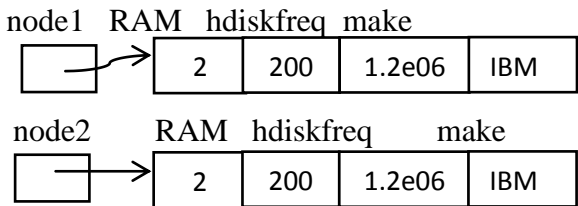
Computer node2 = node1; → node2

As node2 is assigned node1's value, which is nothing but the address of the object referred to by node1, node2 will also point to the same object.

Thus we have not created 2 objects with identical values but we have created 2 reference variables for the same object.

To create new object which is identical to the already existing one, the statement should be as follows:

```
Computer node2 = new Computer(node1);
```



//This will create a new object referenced by node2 & its instance variables will take the values as that of node1.

The object's reference variable node1 itself is passed as an argument.

So, a constructor which takes reference variable as its parameter should be defined to handle this.

```

    Computer(Computer C)
    {
        RAM = C.RAM;
        hdisk = C.hdisk;
        freq = C.freq;
        make = C.make;
    }

```

If 'final' Computer C is given, 'C' cannot be modified within the constructor  
 //as the argument is a reference variable, the parameter receiving it should also be a reference variable of type 'Computer'. node1's instance variable's values are copied to the 'C' reference variable which are later assigned to the instance variables of the object (in our case, node2), used to invoke the constructor.

This is called Copy Constructor.

### Creating an Array of Objects:

In case, 10 objects for the class 'Computer' have to be created, it will be inconvenient to declare 10 different objects and assign them to 10 reference variables.

```
(eg) Computer node1 = new Computer( );
```

...

```
Computer node10 = new Computer( );
```

To make the task simpler, an array of 10 objects can be created as follows:

```
Computer[ ] node = new Computer[10];
```

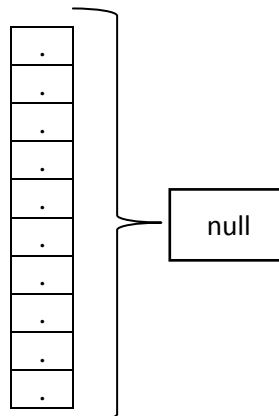
→ allocates an array of references which are null.

```

for(int i=0;i<10;i++)
{
    //Read values for r, h, f, s from the user
    node[i] = new Computer(r, h, f, s);
}

```

Instantiates objects of type 'Computer' and assigns their reference to the elements in this array →



(or)

```
Computer[ ] node = new Computer[10];
for(int i=0;i<10;i++)
node[i] = new Computer( ); //invokes default constructor
```

**Note: Whenever we try to create an array of objects in Java, JVM allocates space for ‘n’ references for your objects but not for the objects themselves.**

Eg., student[] s=new student[10]; creates only an array of 10 reference variables. Then, we should create individual objects explicitly and store their references in this array, as follows

```
for(i=0;i<10;i++)
s[i]=new student(); → parameterized constructor can also be invoked, depending on the type
of object that we want. Address(reference) of ith object is assigned to s[i].
```

In addition to constructor, other methods of the class can also take objects as parameters.

(eg) a method isMemoryEqual( ) is to be written to check whether the memory size of 2 nodes are equal and return true or false.

(eg) If we want to check for the objects, node1 & node2, node1's instance variables will be implicitly available for the method, as it is invoked using node1. But it cannot access the instance variables of node2, unless node2 is passed as an argument to the method.

```
boolean isMemoryEqual(Computer node2)
{
    if((RAM == node2.RAM) && (hdisk == node2.hdisk))
        return true
    else
        return false;
}
...
```

This should be invoked as

```
System.out.println(node1.isMemoryEqual(node2));
(or)
boolean result = node1.isMemoryEqual(node2);
```

### Returning Objects from a Method:

A method can also be made to return an object. Assume a method update( ) is defined for changing the memory configuration & frequency of a given node. If this method does not return any value, still the changes will be retained in the node with which it is invoked.

```
void update( )
{
    RAM = 5;   hdisk = 500;   freq = 3.3e06;
}
↘
```

Can be invoked by

```
node2.update( );
node2.display( ); → will display only the modified values as changes are done
for only node2, in the update( ) method.
```

But if the modified object has to be assigned to another object (like node5 = node2.update( )) or if the update( ) method works on a 'temp' reference variable & makes the modifications in that, finally 'temp' has to be returned.

To handle these, the method should be made to return an object reference.

<pre>Computer update( ) {     RAM = 5;     hdisk = 500;     freq = 3.3e06;     return this; }</pre>	(or)	<pre>Computer update( ) {     Computer temp = new Computer( );     temp.RAM = RAM+1;     temp.hdisk = hdisk+100;     temp.freq = freq+0.1;     temp.make = "Motorola";     return temp; }</pre>
---	------	---

ie., the current object being processed, is returned.

These methods can be invoked as

```
node5 = node2.update( );
(or)
node2.update( ); // In this case, the returned reference is not assigned to any other
                  variable and hence is of no use.
```

'this' Keyword: → Reference variable

- Every instance method has a variable with the name, this, which refers to the current object for which the method is being called. This is used implicitly by the compiler when the method refers to an instance variable of the class.  
(eg) if update( ) method refers to the instance variable RAM, the compiler will insert the 'this' object reference, so that the reference will be equivalent to this.RAM.
- ∴ The method can be written in either of the following 2 ways:

<pre>void update( ) {     RAM = 5;     hdisk = 500;     freq = 3.3e06;     return this; }</pre>	(or)	<pre>void update( ) {     this.RAM = 5;     this.hdisk = 500;     this.freq = 3.3e06;     return this; }</pre>
---	------	--

2 uses of 'this':

- Useful in situations where a local variable has the same name as an instance variable. Here the local variable hides the instance variable. But 'this' can be used to overcome the instance variable hiding.

(eg) Computer (int RAM, int hdisk, float freq, String make)

```
{
    RAM = RAM; → local variables also have the same name as instance variables
    this.RAM = RAM;
    this.hdisk = hdisk;
    this.freq = freq; → it refers to local variable 'freq' of this method.
}
```

→ it refers to the 'freq' instance variable of the object invoking this method

→ This will not create an error. But local variable RAM's value will be assigned to itself and not to the instance variable, RAM.

ii) 'this' can be used to call one constructor from another, in the same class.

- One class constructor can call another constructor in the same class, in its first executable statement.
- This avoids duplicating a lot of code.
- To refer to another constructor in the same class, use 'this' as the method name (constructor name) followed by the appropriate arguments, between the parentheses.

(eg)

Computer(int r, int h, float f) → Stmt A

```
{  
    RAM = r;  
    hdisk = h;  
    freq = f;  
}
```

Computer(int r, int h, float f, String s) → Stmt B

```
{  
    this(r, h, f); → This constructor call should be the first statement and it  
    can only call the constructor of similar type, within the  
    same class.  
}
```

make = s;

Computer node1 = new Computer(2, 200, 2.4e6); -> will invoke constructor A

Computer node2 = new Computer(2, 200, 2.4e6, "AMD");

will invoke constructor B which in turn will invoke constructor A to initialize RAM, hdisk & freq. At last, it initializes 'make'.

Had 'this' not provided the facility of a constructor calling another, the 3 statements in constructor A have to be repeated in constructor B too.

Thus, by making use of 'this', code redundancy is eliminated.

The 'Computer' class in its entirety.

class Computer

```
{  
    private int RAM, hdisk;  
    private float freq;  
    private String make;  
    Computer()  
    {
```

```
        RAM = 2;  
        hdisk = 200;  
        freq = 1.2e06;  
        make = "Intel";  
    }
```

Computer(int RAM, int hdisk, float freq)

```
{  
    this.RAM = RAM;  
    this.hdisk = hdisk;  
    this.freq = freq;  
}
```

Computer(int RAM, int hdisk, float freq, String make)

```

        {
            this(RAM, hdisk, freq);
            this.make = make;
        }
void calculateSpeed( ) //will be invoked for a single core processor
{
    System.out.println("The speed is: " + freq + " cycles per second");
}
void calculateSpeed(booleanismulticore)
{
    System.out.println("Enter the number of cores:");
    Scanner sc = new Scanner(System.in);
int n = sc.nextInt( );
    System.out.println("The speed is: " + freq*n + " cycles per second");
}
void display( )
{
    System.out.println(RAM + "    " + hdisk + "    " + freq + "    " + make);
}
Computer(Computer c)
{
    RAM = c.RAM;
    hdisk = c.hdisk;
    freq = c.freq;
    make = c.make;
}
boolean isMemoryEqual(Computer node)
{
    if((RAM == node.RAM) && (hdisk == node.hdisk))
        return true;
    else
        return false;
}
Compuer update( )
{
    RAM = 5;
    Hdisk = 500;
    freq = 3.3e06;
    return this;
}
}
Class ComputerMain
{
public static void main(String[] args)
{
    Scanner sc = new Scanner(System.in);
    Computer node1 = new Computer( );

```



```

Computer node2 = new Computer(2, 200, 2.4e06);
Computer node3 = new Computer(node1);
System.out.println("Is node2 a multicore processor?");
Computer[ ] server = new Computer[3];
for(int i=0;i<3;i++)
{
    int r = sc.nextInt( ); int h = sc.nextInt( );
    float f = sc.nextFloat( ); String s = sc.next( );
    server[i] = new Computer(r, h, f, s);
}
Computer node5 = new Computer( );
node5 = node1.update( );
node1.display( ); } both will have modified
node5.display( ); } contents
System.out.println("Is server[0] multicore?");
boolean ismulticore = sc.nextBoolean( );
if(ismulticore)
    server[0].calculateSpeed(true);
else
    server[0].calculateSpeed( );
}
}

```

### Some more examples:

1) Stack Class: First in – last out ordering.

Controlled by 2 operations (push & pop)

push – to insert an item on top of the stack

pop – to remove an item from the top of the stack

The following stack class implements a stack for up to 10 integers.

```

class Stack
{
    int[ ] s = new int[10];
    int top;
    Stack( )
    {
        top = -1;
    }
    void push(int item)
    {
        if(top== 9)
            System.out.println("full");
        else
            s[++top]=item;
    }
    int pop()
    {

```

```

if(top<0)
{
    System.out.println("underflow");
    return 0;
}
else
    return s[top--];
}
}
Class StackDemo
{
public static void main(String[ ] args)
{
    Stack stack1=new Stack( );
    for(int i=0;i<10;i++)
        stack1.push(i);
    for(int i=0;i<10;i++)
        System.out.println(stack1.pop( ));
}
}

```

Since access to the stack is through push( ) & pop( ), the fact that the stack is held in an array is actually not relevant to using the stack (ie.,) the stack could be held in a linked list also. Yet the interface defined by push( ) & pop( ) would remain the same.

2) Write a program to define a class named 'ProjectInfo'. It should contain an array of 4 float elements to store the marks obtained by the student in 0<sup>th</sup>, 1<sup>st</sup>, 2<sup>nd</sup> and 3<sup>rd</sup> reviews respectively. It should also contain a variable 'f\_rev' to store final review marks & another variable 'total'. The variable 'grade' should hold the grades scored by the student in the project. The zeroeth review mark (out of 5 marks) should be decided while creating the objects itself. ie., if the student submits the abstract of his project, he will get full 5 marks else 0.

Review 1, 2, & 3 marks should be within the range [0 to 25]. Final review mark should be within [0 to 50]. As long as any of these marks entered is not within the specified range, repeat asking the user to enter the mark again. All 3 review marks (1, 2, 3) should be converted for 15 & added up with zeroeth review mark to get the internal marks.

An array of 60 ProjectInfo objects should be created & their grades should be evaluated.

Also, count the number of 'S' grades & display the highest marks scored in the batch.

```

import java.util.*;
class ProjectInfo
{
private float[ ] review=new float[4];
private float f_rev;
private float total;
char grade;
Scanner sc=new Scanner(System.in);
ProjectInfo()
{
    for(int i=0;i<4;i++)

```

```

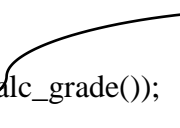
        review[i]=0;
        f_rev=0;
    }
    ProjectInfo(final float mark)
    {
        this();
        review[0]=mark;
    }
    public void getMarks()
    {
        System.out.println("enter the review 1,2 and 3 marks-less than 25 ");
        first:
        while(true)→ This loop will continue until a value within the range [1-25] is
        {
            //entered
            System.out.println("enter the mark within the range 1-25");
            review[1]=sc.nextFloat();
            if(review[1]>=0 && review[1]<=25)
                break first;
        }
        second:
        while(true)
        {
            System.out.println("enter the mark within the range 1-25");
            review[2]=sc.nextFloat();
            if(review[2]>=0 && review[2]<=25)
                break second;
        }
        third:
        while(true)
        {
            System.out.println("enter the mark within the range 1-25");
            review[3]=sc.nextFloat();
            if(review[3]>=0 && review[3]<=25)
                break third;
        }
        fourth:
        while(true)
        {
            System.out.println("enter the mark within the range 1-50");
            f_rev=sc.nextFloat();
            if(f_rev>=0 && f_rev<=50)
                break fourth;
        }
    }
    private float calc_internal()→ This method cannot be called from main class. It should
    {
        be called from within one of the methods of this class.
        System.out.println("the review marks are");
    }

```

```

        System.out.println(review[0]+" "+review[1]+" "+review[2]+" "+review[3]);
        float internal=review[0]+review[1]*3/5+review[2]*3/5+review[3]*3/5;
        System.out.println("internal "+internal);
        return internal;
    }
    private char calc_grade()
    {
        float internal=calc_internal();
        System.out.println("final review "+f_rev);
        total=internal+f_rev;
        System.out.println("total "+total);
        if(total>=90)
            grade='S';
        else if(total>=80)
            grade='A';
        else if(total>=70)
            grade='B';
        else if(total>=60)
            grade='C';
        else if(total>=50)
            grade='D';
        else
            grade='F';
        return grade;
    }
    public void displayresult()
    {
        System.out.println(calc_grade());
    }
    static int count_Sgrade(ProjectInfo[] p)
    {
        int count=0;
        for(ProjectInfo x: p)
        {
            if(x.grade=='S')    (or)
                count++;
        }
        return count;
    }
    static ProjectInfo highest(ProjectInfo[] p)
    {
        ProjectInfo max=p[0];
        for(int i=1;i<3;i++)
        {
            if(p[i].total>max.total)
                max=p[i];
        }
    }

```


 This method is declared 'static' because already the array of objects is being passed to this method. Again using 1 of the objects for invoking it is unnecessary. Being static it can be called using class name. A counting should be done using all objects and not for a single object.

```

        for(int i=0;i<60;i++)
        {
            if(p[i].grade == 'S')
                count++;
        }
    }

```

```

        return max;
    }
}
class ProjectPortal
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        ProjectInfo[ ] p=new ProjectInfo[60];
        for(int i=0;i<60;i++)
        {
            System.out.println("is the abstract submitted on time?");
            boolean b=sc.nextBoolean();
            if(b)
            p[i]=new ProjectInfo(5);
            else
            p[i]=new ProjectInfo();
            p[i].getMarks();
        }
        for(int i=0;i<60;i++)
            p[i].displayresult();
        System.out.println(ProjectInfo.count_Sgrade(p));
        ProjectInfo temp;
        temp=ProjectInfo.highest(p);
        System.out.println("the highest mark is");
        temp.displayresult();
    }
}

```

Creates an array of 60 objects

(or) the code can be replaced by the following

ProjectInfo.highest(p).displayresult( );

← This returns a reference & using that reference, displayresult( ) method is invoked.

### Argument Passing:

Java uses call-by-value to pass all arguments; however the effect differs between whether a primitive type or a reference type is passed.

i) When a primitive type is passed to a method, it is passed by value. Thus a copy of the argument is made & what occurs to the parameter that receives the argument has no effect outside the method.

class sample

```

{
    void modify(int i, int j)
    {
        i *= 2;
        j /= 2;
        System.out.println("i = " + i + " j = " + j);    → 10 5
    }
}
class samplemain
{

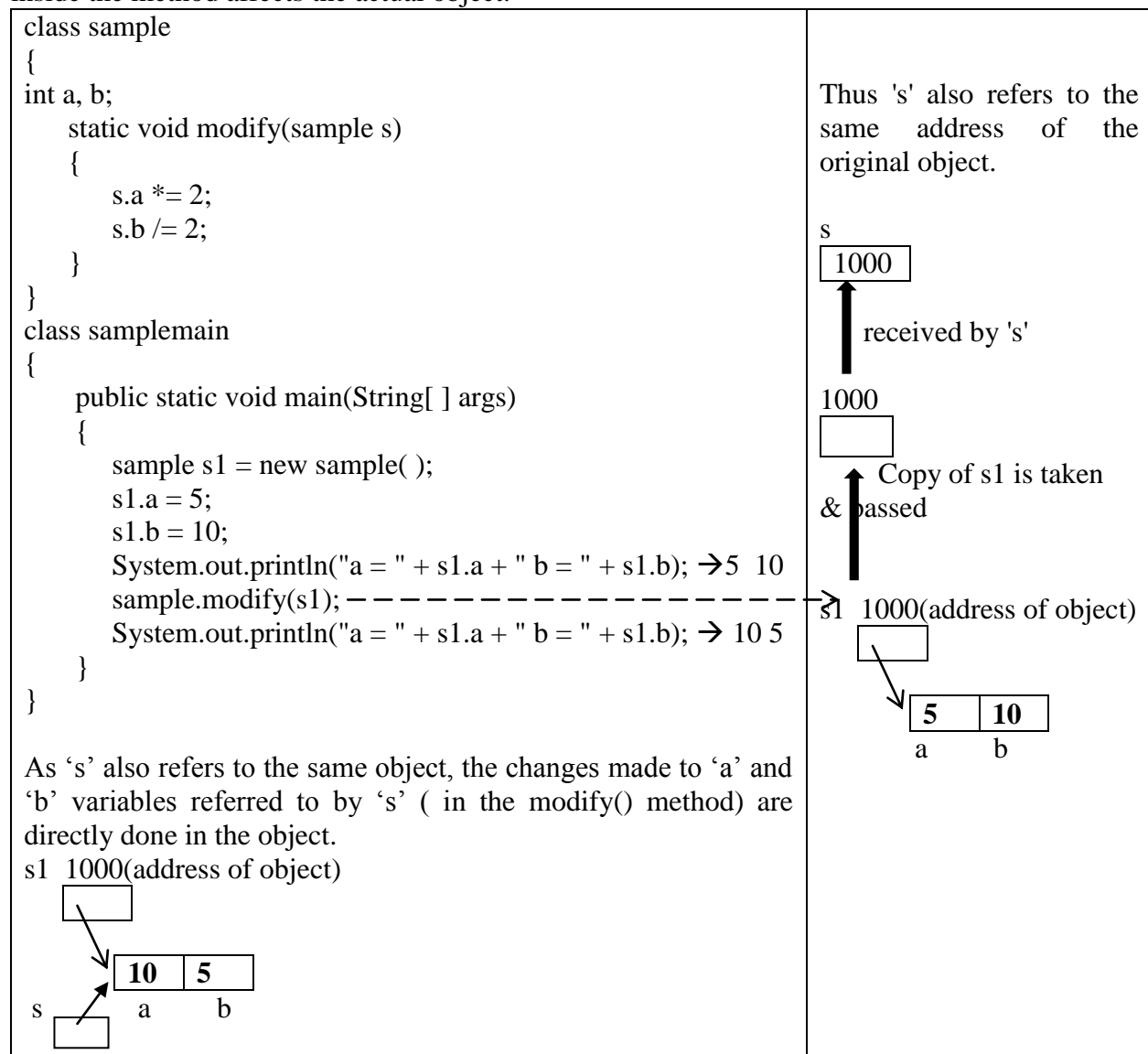
```

```

public static void main(String[ ] args)
{
    sample s = new sample( );
    int a = 5, b = 10;
    System.out.println("a = " + a + " b = " + b); → 5 10
    s.modify(a,b);
    System.out.println("a = " + a + " b = " + b); → 5 10
}

```

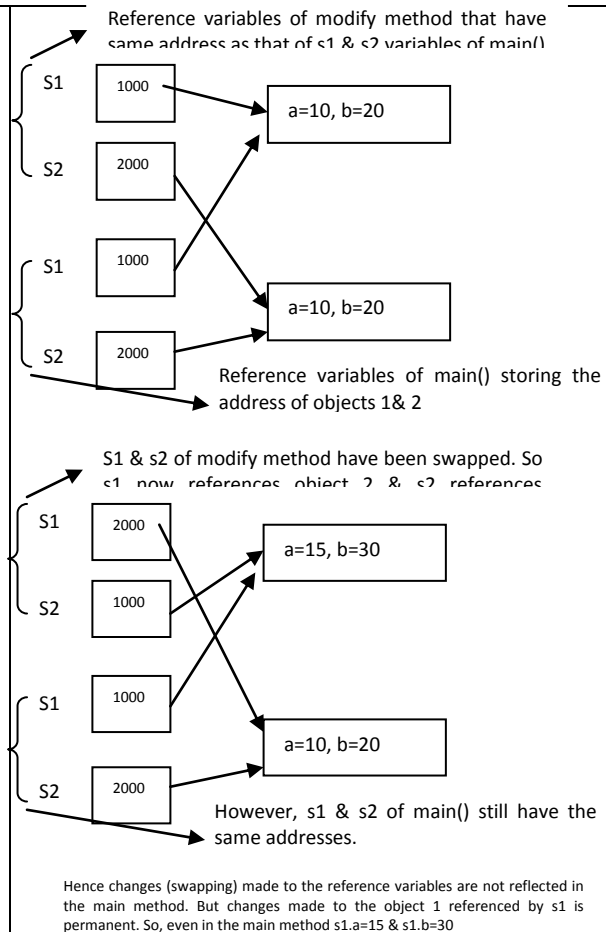
ii) When an object is passed to a method, it works similar to call-by-reference. This is because, when you pass the reference to a method, a copy of that reference is taken and passed. Thus the parameter that receives it will refer to the same object as that referred to by the argument. Thus changes made inside the method affects the actual object.



Point to ponder while passing reference variables to a method: Changes made in the objects referred to by the reference variables, within the called method are reflected even in the calling method and hence **changes made to the objects are permanent**.

On the other hand, **changes made to the reference variables themselves** are not permanent and so not reflected in the calling method.

```
class Sample
{
    int a,b;
    static void modify(Sample s1, Sample s2)
    {
        s1.a=s1.a+5;
        s1.b+=10;
        Sample temp;
        temp=s1;
        s1=s2;
        s2=temp;
        System.out.println(s1.a+ " "+s1.b);
        System.out.println(s2.a+ " "+s2.b);
    }
    public static void main(String[] args)
    { Sample s1=new Sample();
      s1.a=10; s1.b=20;
      Sample s2=new Sample();
      s2.a=100; s2.b=200;
      modify(s1,s2);
      System.out.println(s1.a+ " "+s1.b);
      System.out.println(s2.a+ " "+s2.b);
    }
}
```

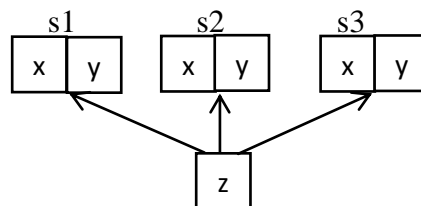


static:

- Static member belongs to the class as a whole rather than the objects created from the class.
- When objects of its class are declared, no copy of a static variable is made. Instead, all instances of the class share the same static variable.

```
(eg) class sample
{
    int x, y;
    static int z;
    ...
}
```

```
...
sample s1 = new sample( );
sample s2 = new sample( );
```



Only one copy of static variable 'z' is created & shared by all 3 objects referred to by s1, s2 & s3.

```
sample s3 = new sample( );
```

- Static instance variables are the controlled version of global variables in Java.

#### Static Methods:

- Can only directly call other static methods.
- Can only directly access static data.
- Cannot refer to 'this' or 'super' in any way.

#### Static Block (or) Static Initializer:

- Not a method.
- A static block can be declared inside a class which will get executed exactly once, when the class is first loaded into the JVM.
- Executed in the order in which they appear textually in the source file.
  - i) Used to initialize all static members, at once.
  - ii) Used to load drivers & other items into the namespace.  
(eg) class.forName("org.Driver"); → will be studied later
  - iii) Useful for some database operations like creating prepared SQL statements.

```
class sample
{
    static int a=3, b;
    static void method1(int x)
    {
        System.out.println(x + "    " + a + "    " + b);
    }
    static → static block
    {
        System.out.println("Static block initalized");
        b = a * 4;
    }
    public static void main(String[ ] args)
{
    method1(10);
}
}
```

As soon as the 'sample' class is loaded into the JVM, all of the static statements are run first.

(i) a is set to 3. (ii) static block executes which sets b to a\*4. (iii) method1( ) although static, should be invoked from elsewhere. Thus, main( ) method is called third, which in turn calls method1( ), by passing 10 to x.

Example 2:

```
public class sample
{
    Static                                → Invoked 1st
    {
        System.out.println("Static block1");
    }
    public sample( )                    → Invoked 6th from main()
}
```



```

    {
        System.out.println("Constructor");
    }
public static String s1 = "static variable"; → executed 2nd
static                                     → Invoked 3rd
    {
        System.out.println("Static block2");
        System.out.println(s1);
    }
public static void main(String args[ ])
    {
        sample s = new sample( );
        s.statmethod2( );
    }
static
    {
        statmethod1( );
        System.out.println("Static block3"); → executed 5th
    }
public static void statmethod1( ) → invoked 4th because of executing the above static
{
    //block
    System.out.println("Stat method1");
}
public static void statmethod2( )          → Invoked 7th
    {
        System.out.println("Stat method2");
    }
}

```

Static variables are rare. But static constants are more commonly used.

(eg) In Math Class

```

public class Math
{
    ...
    public static final double PI = 3.141592633...;
}

```

If PI has not been declared static, we would need an object of the class to access PI and every Math object will have its own copy of PI.

In System Class,

```

public static final PrintStream out = ...;
public static final PrintStream err = ...;
public static final PrintStream in = ...;

```

Example:

Define a class 'Theater' with the following instance variables: nr\_adults, nr\_children, family\_total, class\_type. Also define the following static constants – I\_row\_nr, I\_seat\_nr, I\_seat\_count, II\_row\_nr, II\_seat\_nr, II\_seat\_count, I\_class\_adult, I\_class\_child, II\_class\_adult, II\_class\_child.

Create 'n' number of objects. For each object, read the number of adult tickets & children tickets needed and the class type (I class or II class). If only 1 ticket is needed, invoke the default constructor to set the class type to II. Else invoke the parameterized constructor.

Define a method to allot the seat numbers for the tickets bought.

I\_class\_Row 1 to Row 10; each row 10 seats;

II\_class\_Row 11 to Row 20; each row 10 seats;

If the seats are full, ticket amount should not be calculated & "Housefull" message should be displayed. Else calculate

I\_class Adult ticket – Rs.100, child – Rs.50

II\_class Adult ticket – Rs.60, child – Rs.30

Also calculate the total income.

import java.util.\*;

class theater

```
{
private int family_total;
private int nr_adults;
private int nr_children;
private int class_type;
static int I_row_nr, I_seat_nr;
static int II_row_nr, II_seat_nr, total;
static final int I_CLASS_ADULT, I_CLASS_CHILD;
static final int II_CLASS_ADULT, II_CLASS_CHILD;
static int I_seat_count, II_seat_count;
static
{
    I_CLASS_ADULT=100; // in Rs.
    I_CLASS_CHILD=50;
    II_CLASS_ADULT=60;
    II_CLASS_CHILD=30;
    I_row_nr=1;
    I_seat_nr=0;
    II_row_nr=11;
    II_seat_nr=0;
    total=0;
    I_seat_count=0;
    II_seat_count=0;
}
theater()
{
    nr_adults=1;
    nr_children=0;
    class_type=2;
    family_total=II_CLASS_ADULT;
}
theater(int n1,int n2,int x)
{
    nr_adults=n1;
```

```

        nr_children=n2;
        class_type=x;
    }
    static void calc_total(theater[ ] t)
    {
        for(int i=0;i<t.length;i++)
        {
            total+=t[i].family_total;
        }
        System.out.println("total= "+total);
    }
    private void calc_familytotal()
    {
        if(class_type==1)
        family_total=nr_adults*I_CLASS_ADULT+nr_children*I_CLASS_CHILD;
        else if(class_type==2)
            family_total=nr_adults*II_CLASS_ADULT+nr_children*II_CLASS_CHILD;
        System.out.println("famil total amount="+family_total);
    }
    void get_seat_nr()
    {
        boolean flag=true;
        if(class_type==1)
        {
            System.out.println("nr of seats"+I_seat_count);
            if(100-I_seat_count<nr_adults+nr_children)
                return;
            for(int i=0;i<nr_adults+nr_children;i++)
            {
                if(I_row_nr<=10)
                {
                    if(I_seat_nr==10)
                    {
                        I_row_nr++;
                        I_seat_nr=1;
                        System.out.print("row_nr"+I_row_nr);
                        System.out.println("seat_nr"+I_seat_nr);
                    }
                    else
                    {
                        I_seat_nr++;
                        System.out.print("row_nr"+I_row_nr);
                        System.out.println("seat_nr"+I_seat_nr);
                    }
                    I_seat_count++;
                }
                else

```

I\_class: 10 rows & 10 seats each = 100 seats total  
 I\_seat\_count:gives the number of seats in I class  
 (allotted as of now)  
 (100-I\_seat\_count > (nr\_adults+nr\_children) is checked  
 for every purchase. Likewise, for II\_class also.

```

        {
            System.out.println("cannot be allotted");
            I_seat_nr=nr_adults+nr_children-i;
            flag=false;
            break;
        }
    }
}
else if(class_type == 2)
{
    System.out.println("nr of seats"+II_seat_count);
    if(100-II_seat_count<nr_adults+nr_children)
        return;
    for(int i=0;i<nr_adults+nr_children;i++)
    {
        if(II_row_nr<=20)
        {
            if(II_seat_nr == 10)
            {
                II_row_nr++;
                II_seat_nr=1;
                System.out.print("row_nr "+II_row_nr);
                System.out.println("seat_nr "+II_seat_nr);
            }
            else
            {
                II_seat_nr++;
                System.out.print("row_nr "+II_row_nr);
                System.out.println("seat_nr"+II_seat_nr);
            }
            II_seat_count++;
        }
        else
        {
            System.out.println("cannot be allotted");
            II_seat_nr=nr_adults+nr_children-i;
            flag=false;
            break;
        }
    }
}
if(flag)
    calc_familytotal( );
}
}
class theatermain
{

```

```
public static void main(String[ ] args)
{
    Scanner sc=new Scanner(System.in);
    theater[ ] t=new theater[3];
    for(int i=0;i<3;i++)
    {
        System.out.println("How many adults & how many children?");
        int a=sc.nextInt( );
        int c=sc.nextInt( );
        if((a == 1)&&(c == 0))
            t[i]=new theater( ); //invokes default constructor
        else
        {
            System.out.println("enter the class type 1 or 2");
            int ct=sc.nextInt( );
            t[i]=new theater(a,c,ct); //invokes parameterized constructor
        }
        t[i].get_seat_nr( );
    }
    theater.calc_total(t);
}
```

Output:

How many adults & children?

5

3

Enter I/II class:

1

row nr 1 seat nr 1

row nr 1 seat nr 2

row nr 1 seat nr 3

row nr 1 seat nr 4

row nr 1 seat nr 5

row nr 1 seat nr 6

row nr 1 seat nr 7

row nr 1 seat nr 8

family total =Rs. 650

...

(repeated for three times for the 3 objects in the array)

Total = . . .

#### Var-args Method:

- Varargs – variable length arguments.
- A method that takes a variable number of arguments is called a variable-arity method or var-args method.

- A good example of var-args method:  
`printf( )` method takes a variable number of arguments, which it formats & then outputs.  
 (eg) `printf("hello");` → 1 argument  
`printf("%d", n);` → 2 arguments  
`printf("%d%d", m, n);` → 3 arguments

- A variable-length argument is specified by 3 periods ( . . . )  
 (eg) `void method1(int . . . a)`

There should be a gap here

By seeing such a method signature, the compiler understands that `method1( )` can be called with 0 or more arguments.

'a' is implicitly declared as an array of type `int[ ]` & hence inside `method1( )`, 'a' is accessed using the normal array syntax.

(eg)

class varargs

{	Output:
void method1(int . . . a)	1 (length of the array when the method
{	is called first time)
System.out.println(a.length);	10
for(int x : a)	3 (length)
System.out.println(x);	1
}	2
public static void main(String[] args)	3
{	0 (length)
method1(10);	
method1(1, 2, 3);	
method1( );	
}	

- If there is more than one parameter for the method, then the varargs parameter must be the last.
- There must be only one varargs parameter.
- Varargs method can also be overloaded.

Write a program to define a method '`varargs_method( )`' which should be overloaded for the following:

Version 1 of the method should do the following:

If "rescale", provide only CAT-I or CAT-II marks as argument.

If "regular", provide both CAT-I or CAT-II marks as arguments.

Version 2 of the method should do the following:

Provide 'true' or 'false' values for clearing or not clearing all the subjects in a particular semester. The varying number of semesters will be taken care of, by varargs array.

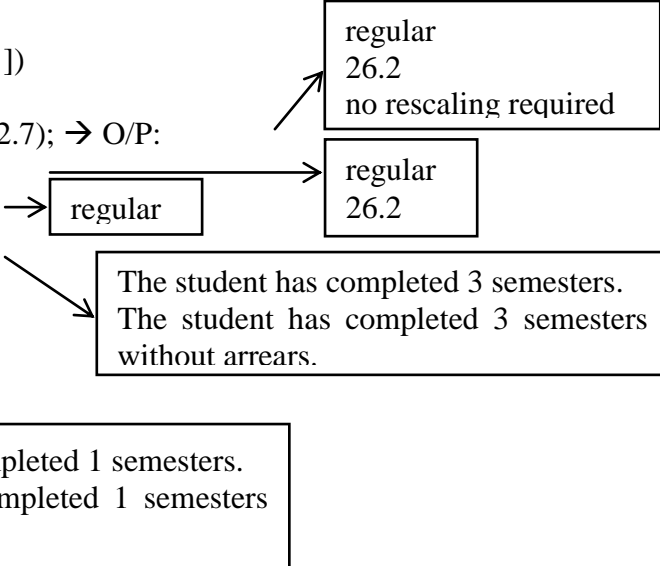
class varargs

```
{
static void varargs_method(String s, float . . . f)
{
    System.out.println(s);
```

```

        if(f.length == 1)
        {
float cat2 = f[0];        float sum = f[0] + cat2;
        }
        else if(f.length == 2)
        {
float sum = f[0] + f[1];
System.out.println(s + "no rescaling required");
System.out.println(sum);
        }
        static void varargs_method(boolean . . . b)
        {
int count = 0;
if(b.length != 0)
{
System.out.println("The student has completed " + b.length + "semesters");
for(int i=0;i<b.length;i++)
{
if(b[i] == true)
count++;
}
System.out.println("The student has successfully completed " + count +
                    "semesters without arrears");
}
}
    }
    public static void main(String ars[ ])
    {
varargs_method("regular", 13.5, 12.7); → O/P:
varargs_method("rescale", 12);
varargs_method("regular");
varargs_method(false, true, true);
varargs_method(true);
    }
}

```



regular  
26.2  
no rescaling required

regular  
26.2

The student has completed 3 semesters.  
The student has completed 3 semesters without arrears.


The student has completed 1 semesters.  
The student has completed 1 semesters without arrears.

### Ambiguity related to varargs method:

I) (eg)

```
class varargs
{
    static void method1(int ... i) → statement A
    {
        ...
    }
    static void method1(boolean ... b) → statement B
    {
        ...
    }
    public static void main(String args[ ])
    {
        method1(1, 2, 3); → invokes A
        method1(true, false); → invokes B
        method1( ); → This call could be translated into a call to
                        statement A or B. Both are equally valid.
    }
}
```

→ This program would give compilation error because



II) (eg)

```
static void method1(int ... v) → statement A
{
    ...
}
static void method1(int n, int ... v) → statement B
{
    ...
}
```

If there is a function call like method1(5), the compiler cannot resolve the problem of which function should be called, because this call can be translated into method1(int ... v) with 1 varargs argument (or) can be translated into statement B with 1 integer argument & zero varargs argument.

### Nested Classes:

- Defining a class within another class.

### Why nested classes?

- 1) A way of logically grouping classes that are only used in one place.

If a class is useful to only one other class, then it is logical to embed in it, that class & keep the two together. Nesting makes their package more streamlined.

- 2) It increases encapsulation.

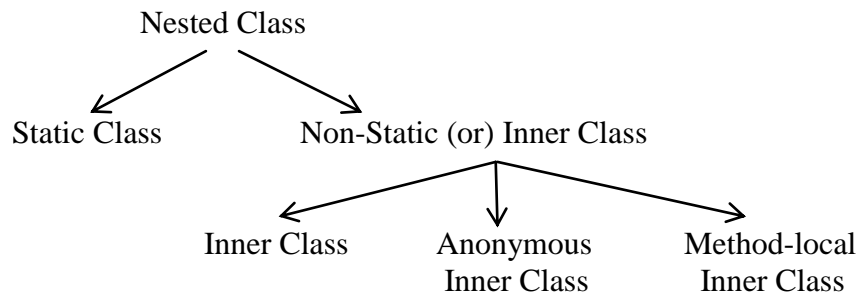
Consider 2 distinct classes A & B, where B is the only class that needs access to members of A, that would otherwise be declared private. By hiding class B within class A, A's members can be declared private & B can access them. In addition, B itself can be hidden from the outside world.



3) Nested classes can lead to more readable & maintainable code.

Nesting places the code closer to the place where it is used. If class B is nested within class A, we can write another class B as a top-level class (ie., visible to the entire world). This has nothing to do with the one already present inside class A.

- If class B is nested within class A, then B is known to A, but not outside of A.
- A nested class can even access private members of the class in which it is nested.
- But enclosing class has no access to the members of the nested class.

Static Class:

- Since it is static, it cannot access the non-static members of its enclosing class directly (can be accessed only through an object). Thus, it is seldom used.

Inner Class:

- Non-static nested class, declared outside of any method of the outer class.
- Can access all the variables & methods of the outer class directly as other non-static methods of the outer class do.

Method-local Inner Class:

- Defined within a method of the enclosing class.
- For the inner class to be used, you must instantiate it & that instantiation must happen within the same method, but after the class definition code.
- Cannot use variables declared within the method (including parameters) unless those variables are marked final.
- Only modifiers that can be applied to a method-local inner class are abstract & final (but not both at the same time).

Anonymous Inner Classes:

- Inner classes declared without a name.
- Always created as part of a statement.

Instantiation of an inner class and a static nested class

```

class outclass1 → outer class
{
    class inclass1 → inner class
    {
        ...
    }
}
  
```

```

class outclass2 → outer class
{
    static class inclass2 → static nested class
    {
        ...
    }
}
  
```

Now the inner class and static nested class can be instantiated in 2 ways.

i) From within its outer class

Inner class	Static nested class
<code>inclass1 ic1 = new inclass1( );</code>	<code>inclass2 ic2 = new inclass2( );</code>

Thus, if an object for the nested class has to be created, from within its outer class, the syntax is same, regardless of whether it is inner or static nested.

ii) From outside its outer or enclosing class

Inner class	Static nested class
<p>As this class is a non-static member of this class, it should be accessed only through an object of the outer class. So, first create an object for the outer class &amp; use it to create object for the inner class. Syntax:</p> <pre>outclass1 oc1 = new outclass1( ); outclass1.inclass1 ic1 = oc1.new inclass1( );</pre> <p style="text-align: center;">↓ object of outer class</p> <p style="text-align: center;">(or)</p> <p>Instead of writing these 2 statements, a single statement can be written as  <code>outclass1.inclass1 ic1 = <u>new outclass1( )</u> . new inclass1( );</code>  <span style="margin-left: 150px;">↓</span>          This first creates object for the outer class which is then used to create object for the inner class.</p>	<p>As this class is defined as a static member of the outer class, it <u>can be access only by making use of the class name</u>. No need of any object. Syntax:  <code>outclass2.inclass2 ic2 = new outclass2.inclass2( );</code>  <span style="margin-left: 150px;">↓</span>          The static class inclass2 is accessed by using the outer class, outclass2.</p>

#### Example for inner class:

<pre>class Outer { int x=100; void test( ) {     Inner ic=new Inner();     ic.display(); } class Inner {     int y=10;     void display()</pre>	<p>→ When this program is compiled, 3 class files would be created.</p> <ul style="list-style-type: none"> <li>i) Outer.class</li> <li>ii) Outer\$Inner.class → the inner class</li> <li>iii) Innerclass.class      will always be tied to its outer class, although it is distinct</li> </ul>
---	--

```

    {
System.out.println(x);
    }
    }
    void showy()
    {
System.out.println(x);
    }
}
class innerclass
{
    public static void main(String[] args)
    {
        Outer oc=new Outer();
        oc.test();
    }
}

```

→ Any code in class 'Inner' can directly access 'x' variable

✗ This will produce an error, as the member 'y' of the inner class is known only within the scope of the inner class and may not be used by the outer class.

Example for Method-local inner class:

```

class Outer
{
    int x;
    void test( )
    {
        class Inner
        {
            void display()
            {
                System.out.println(x);
            }
        }
    }
}

```

Inner ic=new Inner(); ➔ Note: Object also should be created within the same scope (ie., within test( ) method) where the class has been defined.

```

ic.display();
}
}
class innermain
{
    public static void main(String[] args)
    {
        Outer oc=new Outer();
        oc.test();
    }
}

```

More into the intricacies of inner & static nested classes:

Assume there is an instance variable with the same name 's' in both classes.

Example 1: Inner Class

```

class outer
{
private String s="Outer string";
int x = 10;
inner ic=new inner();
class inner
{
    private String s="Inner string";
    public void display()
    {
System.out.println(x); → O/P: 10
        System.out.println(s);→ O/P: Inner string
        System.out.println(new outer().s);→ O/P: Outer string (To access the 's' variable
            (or)                                of outer class, create an object for the
        System.out.println(outer.this.s);        outer class.)
    }
}
void display()
{
    System.out.println(s);→ O/P: Outer string
    System.out.println(ic.s);→ O/P: Inner string
}
}
class inner_demo
{
public static void main(String[] args)
{
    outer oc=new outer();
    oc.display();
    outer.inner oic=oc.new inner();
    oic.display();
}
}

```

### Example 2: Static Nested Class

Assume there are both static & non-static members in both the outer as well as inner classes.

```

class nested
{
static String s1="outer static";
static int n=10;
private static int x=5;
String s2="outer nonstatic";
nestedinner ni=new nestedinner();
void display()
{
System.out.println("outer class members from outer class");
    System.out.println(s1);→ O/P: outer static
}
}

```

```

System.out.println(s2); → O/P: outer non-static
System.out.println("inner class members from outer class");
System.out.println(nestedinner.s1);
System.out.println(ni.s2);
System.out.println("a="+a);
}

```

As 's1' is a static member of 'nestedinner', use the class name itself to access it.  
O/P: inner static

Error since enclosing class cannot access members of nested class, directly.

's2' is a non-static member. So, use object to access  
O/P: inner static

```

static class nestedinner
{
static String s1="inner static";
int a=100;
String s2="inner nonstatic";
void display()
{
System.out.println("inner class members from inner class");
System.out.println(s1); → O/P: inner static
System.out.println(s2); → O/P: inner non-static
System.out.println("outer class members from inner class");
System.out.println(nested.s1);
System.out.println(new nested().s2);
System.out.println(n);
System.out.println("x="+x);
}
}
}
class nested_demo
{
public static void main(String[] args)
{
nested.nestedinner ni=new nested.nestedinner();
ni.display();
nested n=new nested();
n.display();
}
}

```

's1' is static member of 'nested' class. So, use the class name to access it.  
O/P: outer static

's2' is a non-static member of 'nested' class. So, create object for 'nested' class & then use it to access 's2'.

Use the outer class name to access the nested class name, as it is a static member.

→ Creates 3 .class files

- i) nested.class
- ii) nested\$nestedinner.class
- iii) nested\_demo.class

## Garbage Collection

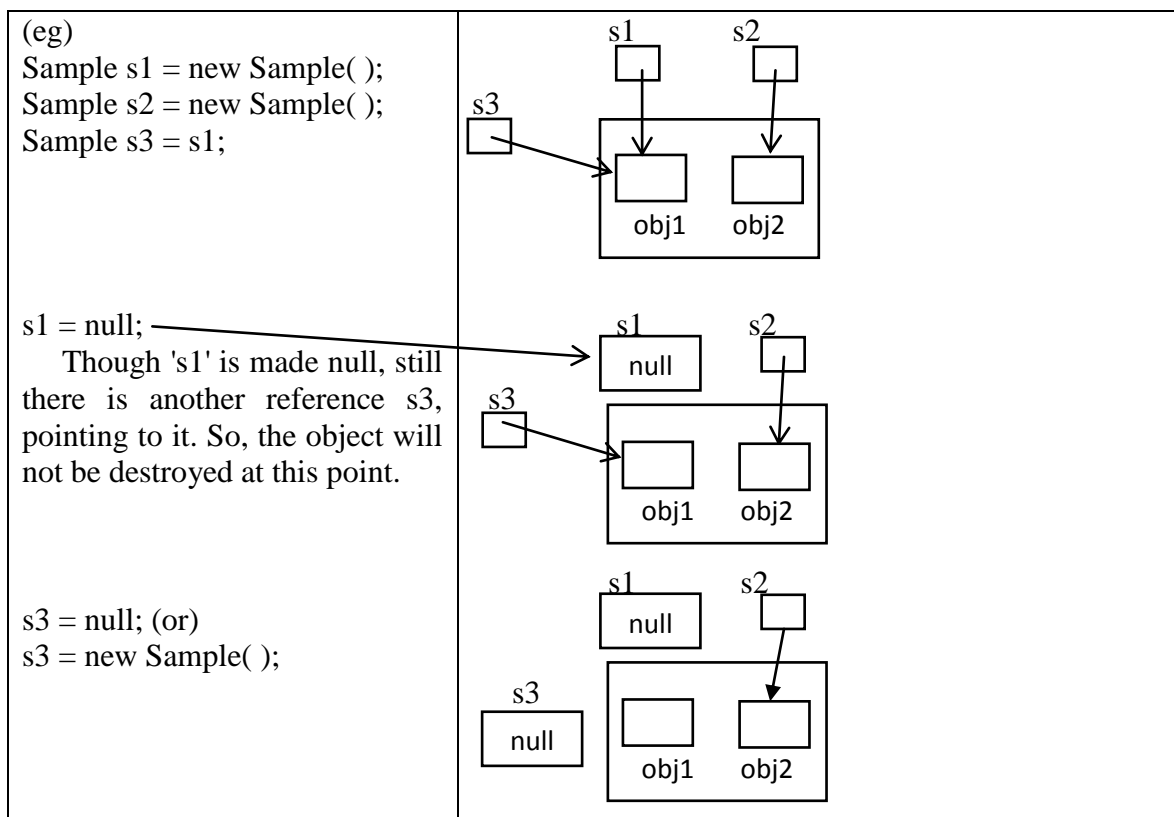
A technique in Java which handles de-allocation of memory allotted for objects, automatically.

As the program runs, the execution engine of the JVM creates objects of the classes. These objects are maintained in the form of a heap (a special type of tree) & stored in the area called Java Objects Heap.

Any object which is no longer required for execution is called 'garbage'. To optimize the available memory usage, it is necessary to collect and destroy the garbage so that the heap space it occupies can be recycled and thus the space is made available for subsequent new objects. This automatic cleaning of memory by destroying the unwanted objects is called garbage collection. In the JVM, a daemon thread runs as a background process, all the time, for the task of automatic garbage collection, which takes place sporadically (irregular intervals).

### When an object becomes eligible for garbage collection?

- If there are no reference variables pointing to the object, this object can be garbage collected, ie., removed from memory.

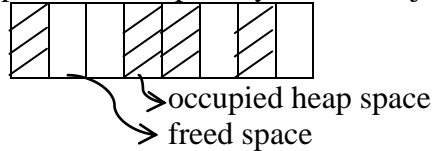


Now the object 'obj1' does not have any references. So, it will be destroyed by the garbage collector in the next round, automatically.

In addition to automatic de-allocation, garbage collector also moves the objects within the heap memory as the application runs to reduce heap fragmentation.

### What is Heap Fragmentation?

When unreferenced objects are freed, it will result in free portions of heap memory, left in between the portions occupied by the live objects.



Thus, even though there is enough total unused space in the existing heap, to allocate new objects, they are not contiguous to fit in the new object. Thus the size of the heap has to be expanded to allocate space for new objects. This ever-growing heap can degrade the performance of the executing program (or) can cause the virtual machine to run out of memory. Garbage collector helps to avoid this problem by moving the objects within the heap.

#### Advantages of automatic garbage collection:

- i) Programmers can spend effort in the core logic of the program and be more productive rather than analyzing the execution part and focusing on housekeeping activities such as how to free the memory.
- ii) Increases the security and integrity of the Java program by protecting it from accidentally (or purposely) crashing the JVM, by incorrectly freeing memory.

#### Disadvantages:

- i) JVM has to keep track of all the objects created and used. This affects the performance of the program and impacts the speed.
- ii) Programmers in a garbage-collected environment have less control over the scheduling of CPU time devoted to freeing objects that are no longer needed.

#### finalize( ):

Finalization is a mechanism to define the specific actions that will occur, when an object is just about to be reclaimed by the garbage collector.

ie., if an object is holding some non-Java resources such as a file handle or graphics resource like character font, then these resources must be freed before the object is destroyed. Otherwise, if we use enough of these resources, our program & possibly other programs, may stop working.

Thus to free the resources, the finalize( ) method must be defined. When the garbage collector encounters an unused object, just before destroying the object, it calls the finalize( ) method on the object.

Java has a class called 'Object' which is the parent class of all other classes in the Java Class Hierarchy. The 'Object' class has a method finalize( ). This method in the 'Object' class can be overridden in the user defined classes as follows:

<pre><u>protected</u> void finalize( ) {     //finalization code here }</pre>	<p>'protected' access specifier prevents access to finalize( ) method by code defined outside its class, except its derived class, but still subclasses can provide an implementation of it, if needed.</p>
---	---

Java's garbage collector runs periodically to recycle unused objects. Sometimes, if we wish to collect the discarded objects prior to the collector's next appointed rounds, we can request garbage collection by calling the gc( ) method.

2 ways to invoke gc( ) method:

- i) gc( ) is a static method of System class. So, it can be invoked by writing the following statement.

System.gc( );

- ii) 'Runtime' class of *java.lang package* can be used (gc( ))

- Runtime class encapsulates the runtime environment.
- A Runtime class cannot be instantiated. But a reference to the current Runtime object can be obtained by calling the static method Runtime.getRuntime( ). Once a reference is obtained, the gc( ) method in Runtime class can be invoked, using that. The following methods are useful for memory management.

long freeMemory( ) → returns the number of bytes of free memory available to the application program.

void gc( ) → Initiates garbage collection.

long totalMemory( ) → returns the total heap memory in JVM.

Totalmemory=memory used by objects + free memory

These methods can be used to determine how much space is needed for one instance, of the class & how much of object heap memory is free. Thus we can approximate how many more objects of a certain type can be instantiated.

Example:

class memorydemo

```
{
public static void main(String args[ ])
{
    Runtime r = Runtime.getRuntime( );
    long beginning, ending;
    System.out.println("Total Memory=" + r.totalMemory( ));
    r.gc( ); //to destroy any unused objects
    beginning = r.freeMemory( );
    memorydemo[ ] m = new memorydemo[10000];
    for(int i=0;i<10000;i++)
    {
        m[i] = new memorydemo( );
    }
    r.gc( ); //to destroy unused objects
    ending = r.freeMemory( );
    int objsize = (int)((beginning – ending) / 10000);
    System.out.println("A single object occupies " + objsize + "bytes");
}
}
```

Before creating a large number of objects, invoking gc( ) is advisable for 2 purposes.

- i) We can start with as much free memory as possible.
- ii) Reduce the likelihood of the garbage collector running during the task, especially in a time-critical application that might be affected by garbage collection overhead.

public void assureMaxFreememory( )



```

{
    Runtime rt = Runtime.getRuntime( );
    long before;
    long after = rt.freeMemory( );
    do
    {
        before = after;
        rt.gc( );
        after = rt.freeMemory( );
    }while(before<after);
}

```

→ This loop repeatedly calls gc( ) method, till there are no more objects to be destroyed. In such a situation, the amount of free memory before and after calling gc( ) method, will be the same.

## String

String is a sequence of characters. Java does not implement strings as character arrays but as objects of class String. Even string literals are actually string objects.

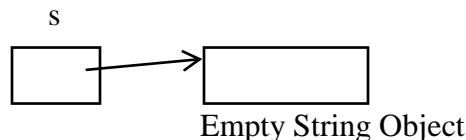
(eg) in System.out.println("Hello World");

→ This string literal is also treated as String object. This makes string handling convenient as Java has predefined methods to compare two strings, concatenate, change the case of the string, search for a substring, etc.

### Basic 9 String Constructors

i) To create an empty string

String s = new String( ); → creates an instance of String with no characters in it.  
's' is the reference variable which stores the reference to the object, thus created.



Objects of type String are immutable (ie., its size or the contents cannot be changed).

Each time we need an altered version of an existing string, a new String object is created that contains the modifications. Original string is left unchanged.

(eg) System.out.println("Hello World" + "Program");

This creates one string object with these characters as its data members

H  
e  
l  
l  
o  
w  
o  
r  
l  
d

A

H  
e  
l  
l  
o  
w  
o  
r  
l  
d  
P  
r  
o  
g  
r  
a  
m

This creates another string object

B

P  
r  
o  
g  
r  
a  
m

When these 2 are concatenated, the result is neither stored in String object labelled A nor in B. But a new object, which is entirely different from A & B is created with these data members

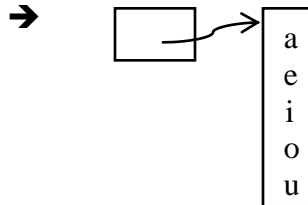
- Fixed immutable strings can be implemented more efficiently than changeable ones.
- If modified string is desired, String Buffer and String Builder classes can be used.

ii) To create strings with character array

String (char[ ] chars)

(eg) char [ ] vowels = {'a','e','i','o','u'};

String s = new String(vowels);



iii) To create a String from a portion of a character array

String(char[ ] chararray, int startindex, int numchars)

Index at which the  
subrange should start

number of characters to  
use

(eg) char [ ] vowels = {'a','e','i','o','u'};

String s = new String(vowels,2,3);

→ s="iou"

From 2<sup>nd</sup> character use 3 characters to build the string

iv) To create a String from a byte array

Though Java's char type uses 16 bits to represent the basic Unicode char set, the typical format for strings on the Internet uses arrays of 8-bit bytes constructed from ASCII char set.

Thus if the programmer is assured that the characters in a string are from ASCII char set, it is advisable to use byte data type rather than char.

So, String class provides constructors that initialize a string when a byte array is given.

String (byte[ ] bytearray)

➤ Which contains integers within the range of -128 to +127

v) To create a String from a portion of the byte array

String(byte[ ] bytearray, int startindex, int numchars)

(eg) byte[ ] bytearray = {65,66,67,68,69,70};

➤ Even if negative numbers are given, the equivalent characters are displayed

String s1 = new String(bytearray);

System.out.println(s1); ➔ ABCDEF

String s2 = new String(bytearray,3,3)

System.out.println(s2); ➔ DEF

Note:

In both (iv) and (v), the byte to character conversion is done by using default character encoding of the platform.

- vi) To construct a String object that contains the same character sequence as another object

String(strObj) → string object

```
(eg) char[] c = {'J','A','V','A'};
      String s1 = new String(c);
      String s2 = new String(s1);
      System.out.println(s1); → JAVA
      System.out.println(s2); → JAVA
```

- vii) To build a String from a StringBuffer object

String(StringBuffer sbuff)

- viii) To construct a String from a StringBuilder object

String(StringBuilder sbuild)

- ix) To construct a String from an array that contains Unicode code points

The array should be an **integer** array to store the Unicode sequences.

String(int[] codepoints, int startindex, int numchars)

String, StringBuffer and StringBuilder classes are defined in java.lang - Hence, available to all programs, by default.

- These classes are final i.e., none of these can be subclassed

- All these classes implement CharSequence interface.

But a string reference variable of one object can be changed to point to some other string object, at a later point of time.

### Important points regarding Strings:

- i) Automatic creation of String instances from String literals

(eg) String s1 = "abc";

This automatically calls the constructor type (vi) through the new operator. This is implicitly done.

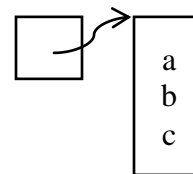
Thus a String object with the data members 'a, b, c' is created and its reference is stored in s1.

Hence, a string literal can be used in any

place, a String object is used.

i.e., All the methods of the String class can be called directly on a quoted string, as if it were an object reference.

(eg) s1.length();                      (or)    "ABC".length();  
       s1.charAt(0);                    (or)    "ABC".charAt(0); } are valid

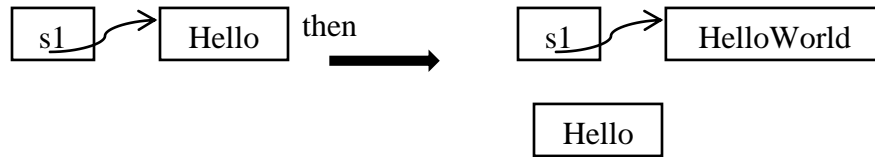


- ii) What is meant by immutable?

(eg) String s1 = "Hello";

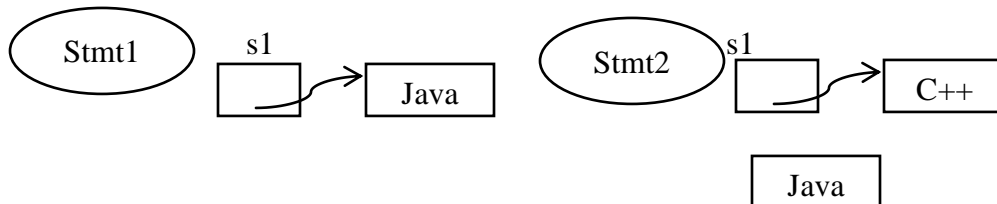
s1 = "Hello" + "World";    (or)    s1 = s1 + "World";

⇒ Creates a new String object.



Initially 's1' was pointing to the object containing "Hello". Then 's1' is referencing the object containing "HelloWorld". For each modification, the original string is not modified. But a new string is constructed, including those modifications.

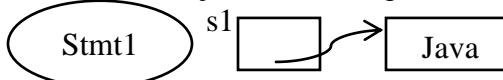
Let String s1 = "Java";  
s1 = "C++"; } valid because the actions that are performed, for these statements are



Now, 's1' is the reference of another object, containing "C++".

However,

```
String s1 = "Java";
String s1 = "C++";
```



→ This is an error because statement 2 tries to create another reference variable with the same name, which is wrong.

Consider an object of StringBuffer class.

To the original string, some characters can be appended, inserted or removed, using the methods defined in the class.

```
(eg) StringBuffer sb = new StringBuffer("Hello");
sb.append("world");
```

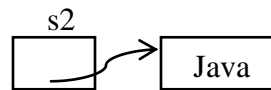
→ to the same String Buffer object, "world" is appended.

This is not possible with a String object. Thus, we say that it is immutable.

### iii) Difference between implicit constructor call for creating the string and explicit call

<u>Implicit</u>	<u>Explicit</u>
<pre>String s1 = "Java"; String s2 = "Java"; if(s1 == s2) S.o.p("Both reference are same");</pre>	<pre>String s1 = "Java"; String s2 = new String("Java"); if(s1 == s2) S.o.p("They are equal");</pre>
<p>O/P: Both reference are same</p> <p>This is because, when JVM implicitly calls the String class constructor, it first searches the heap area to ascertain whether any memory chunk of the String object</p>	<p>⇒ No output is shown</p> <p>When the constructor is explicitly called through the new operator, a new separate memory chunk is allocated.</p> <pre> graph LR     s1[s1] --&gt; Java[Java]   </pre>

has the same content. *That object also should have been created using implicit constructor call.* If so, it uses the existing one. In the example, there is already an object in the heap containing "Java" and so, no new object will be created. Instead the existing object's reference is returned and stored in 's2'. Hence, s1 = s2.



Thus s1 & s2 are not equal and so 'if' block does not execute.

Note:

String s1 = new String("Java");  
String s2 = "Java"; Even for this example, a new object will be created for 2<sup>nd</sup> statement as the 1st object has not been created through implicit call.

### Only operator allowed for String manipulations – '+'

'+' is the only operator that can be applied for String objects.

This concatenates 2 strings and produces a new String object as a result.

(eg) String s = "Java";

s = s + "Programming";

s = s + "is easier";

System.out.println(s); → Java Programming is easier

'+=' operator can also be used.

String s = " "; s += "hello";

### Basic methods of String Class:

#### i) toString( )

- It is basically a method of 'object' class – the parent class of all other classes. Therefore, every class implements toString( ).
- The toString( ) method returns a string that contains the description of the object with which it is called.
- It is automatically called when an object is output using println( ) or when the object is used in a concatenation expression.

(eg) class StringClass

```

{
    String s = new String("Java");
    void display( )
    {
        System.out.println(s);
    }
}
  
```

class StringMain

```

{
    public static void main(String[ ] args)
    {
        StringClass sc = new StringClass( );
        System.out.println(sc); → StringClass@1815F59
    }
}
  
```

```
    }
}
```

Since the object of a class 'sc' is used in `println( )` it automatically invokes the `toString( )` method of 'Object' class.

This method returns a string consisting of the name of the class of which the object is an instance, the at-sign character, '@' and the hexadecimal representation of the hashcode of the object. In other words, this method returns a string equal to the value of:

```
getClass( ).getName( ) + '@' + Integer.toHexString(hashcode( ))
```



Hence, the output - `StringClass@1815F59`


Many classes override this method. Even for the user defined classes, we can override this method, to tailor the object description specifically for the type of object.

The previous program can be modified to override the `toString( )` method.

```
classStringClass
```

```
{
    String s = new String("Java");
    void display( )
    {
        System.out.println(s);
    }

    public String toString( )  return type is 'String'
    {
        String temp;
        temp="The instance variable is the object" + s;
        return temp;  the method should return a 'String'
    }
}

class StringMain
{
    public static void main(String[ ] args)
    {
        StringClass sc = new StringClass( );
        System.out.println(sc);  invokes the toString( ) method which is
                                overridden in the class 'StringClass'
    }
}
```

Output:

The instance variable is the object Java

Note:

- Any description for the object can be given in that method.
- Even for 'projectPortal' class or 'Theatre' class of any other user-defined class, `toString( )` method can be redefined.
- `System.out.println(sc.toString())` too will display the contents of the current String object and not the description of the String class

ii) `length( )`

`int length( )` – gives the number of characters in the string.

```
(eg) char[] chars = {'a','b','c'};
      String s = new String(chars);
      System.out.println(s.length()); ➔ O/P: 3
```

#### Difference between 'length' and 'length()'

length – An instance variable of array object which gives the maximum number of elements that an array can store.

length() – A method in String class which gives the number of characters in a string.

#### Character Extraction Methods:

##### iii) charAt()

```
char charAt(int pos)
```

- returns the character at the specified index or position.
- Gives an error if negative index is given.

```
(eg) char c;
      c = "abc".charAt(1); ➔ b
```

This method can be used to read a character type of input through command line arguments or using Scanner class.

```
(eg)
class argsDemo
{
    public static void main(String[] args)
    {
        if(args[0].charAt(0) == 't')
            System.out.println("true");
        else
            System.out.println("false");
    }
}
```

If the input is given as

```
Z:\> java argsDemo t a f s
```

The 0<sup>th</sup> argument is the String "t". In the String "t", if the 0<sup>th</sup> character is retrieved, it means that a character type of input is read.

```
import java.util.*;
class scannerChar
{
    public static void main(String[] args)
    {
        Scanner sc=new
                        Scanner(System.in);
        if(sc.next().charAt(0) == 't')
            System.out.println("true");
        else
            System.out.println("false");
    }
}
```

sc.next() method returns a String. In that String obtain the 0<sup>th</sup> character using charAt(0) method.

##### iv) getChars()

- To extract more than one character at a time.

```
void getChars(int sourcestart, int sourceend, char target[], int targetstart)
```

where,

sourcestart – indicates the beginning of the substring

sourceend – one past the end of the substring. (substring contains characters from start through sourceend – 1)

target[] – array that will receive the characters

targetstart – index within target at which the substring will be copied

```
(eg)
class demo
{
    public static void main(String[ ] args)
    {
        String s = "We learn String Class";
        int start = 9, end = 15;
        char[ ] buff = new char[end-start]; → creates a char array of size '6' to
        s.getChars(start, end, buff, 0);      accommodate "String"
        System.out.println(buff);
    }
}
```

O/P: String

The method starts copying from 9<sup>th</sup> character upto 14<sup>th</sup> character into the buff[ ] array. This copying starts from 0<sup>th</sup> position of the buff array and not somewhere in the middle of the array.

#### v) toCharArray( )

- To convert all the characters in a String object into a character array.

char[ ] toCharArray( )

(eg) String s = "Java Program";

char c = s.toCharArray( );                      → c[ ]

J	a	v	a		P	R	o	g	r	a	m
---	---	---	---	--	---	---	---	---	---	---	---

Difference between getChars( ) and toCharArray( )

<u>getChars( )</u>	<u>toCharArray( )</u>
i) Return type is void	i) Return type is char Array
ii) Has arguments	ii) No arguments
iii) All characters or a portion of a string can also be obtained.	iii) Converts the entire string into character array.
iv) The character array should be created large enough to hold the characters. (It must be large enough to accommodate the longest substring).	iv) The array will be implicitly created with the size equal to the number of characters in the string.
(eg) String s = "Java Program"; char[ ] target = new char[12]; s.getChars(0,13,target,0);	(eg) String s = "Java Program"; char[ ] target = s.toCharArray( );

#### vi) getBytes( )

- Alternative to getChars( ) which stores the characters in an array of bytes.
- But syntactically similar to 'toCharArray( )' having no arguments and returning byte array.  
byte[ ] getBytes( )
- Useful when the string value has to be exported into an environment that does not support 16-bit Unicode characters. (ie., most Internet Protocols & text file formats that use 8-bit ASCII).

(eg)

String s1 = "Hello";

Much simpler way of using it

String s1 = "Hello";



```
int i = s1.length( );
byte[ ] b = new byte[i];
b=s1.getBytes( );
for(int j=0;j<i;j++)
System.out.println(b[j]);
```

O/P: 72 101 108 108 111

```
byte[ ] b = s1.getBytes( );
for(int i=0;i<b.length( );i++)
System.out.println(b[i]);
```

O/P: 72 101 108 108 111

String Comparison Methods:

vii) equals( )

boolean equals(Object)

Object to be compared with the  
invoking String object

- Case sensitive & returns true if the 2 strings contain the same characters in the same sequence.

viii) equalsIgnoreCase( )

boolean equalsIgnoreCase(String)

- To make case insensitive comparison.

(eg) String s1="Hello", s2="Hello", s3="Hi", s4="HELLO";

S.o.p(s1.equals(s2)); → true → 2 strings have to be compared. One string is

S.o.p(s1.equals(s3)); → false used to invoke the method. So, it is implicitly

S.o.p(s1.equals(s4)); → false available to the method. But the other string

S.o.p(s1.equalsIgnoreCase(s4)); → true 's2' should be supplied as an

S.o.p(s1.equalsIgnoreCase(s3)); → false argument.

Difference between equals( ) method and == operator

equals( ) → compares the characters inside a String object

== operator → compares 2 object references to see whether they refer to the same instance.

(eg) String s1="Hello";

String s2=new String(s1);

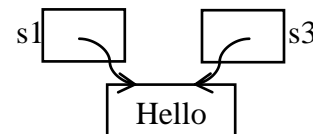
String s3="Hello"; (or) String s3=s1;

System.out.println(s1.equals(s2)); → true

System.out.println(s1 == s2); → false

System.out.println(s1 == s3); → true

System.out.println(s1.equals(s3)); → true



Thus, s1 == s3 and s1 != s2

But, s1 equals s2 and s1 equals s3

ix) compareTo( )

int compareTo(String str)

→ The string to be compared with the invoking string

- Returns the difference of ASCII values of the 2 given characters.

- Return value

If &lt; 0 → Invoking string &lt; str

If &gt; 0 → Invoking string &gt; str

If zero → 2 strings are equal

(eg) This method can be used to sort the array of strings

classsortMain

```

{
static String[ ] names={"Vino","Anu","Prasad","Sunny","Benny"};
public static void main(String[ ] args)
{
    for(int i=0;i<names.length;i++)
    {
        for(int j=i+1;j<names.length;j++)
        {
            if(names[i].compareTo(names[j])>0)
            {
                String temp=names[i];
                names[i]=names[j];
                names[j]=temp;
            }
        }
        System.out.println(names[j]);
    }
}
}

```

x) compareToIgnoreCase( )

- To ignore case differences when comparing 2 strings.

xi) startsWith( )

- To find whether a given string begins with a specified string.

boolean startsWith(String str)

(eg) String s="Hello";

System.out.println(s.startsWith("Hell"));     ➔ True

xii) startsWith(String str, int startIndex)

boolean startsWith(String str, int startIndex)

➔ index into the invoking string at  
which the search will begin

(eg) "treat".startsWith("eat",2); ➔ true

↙  
Check for the letter 'e' in the index 2 of the invoking string

xiii) endsWith( )

boolean endsWith(String str)

(eg) "Welcome".endsWith("come"); ➔ true

"newcomer".endsWith("come"); ➔ false

xiv) regionMatches( )

- Compares a specific region inside a string with another specific region in another string.

boolean regionMatches(int startIndex, String str2, int str2startIndex,  
int numChars)

where,

startIndex – represents the index at which the region begins within the invoking string

str2 – String being compared

str2startIndex – index at which comparison will start within string 2

numChars–length of substring to be compared

boolean regionMatches(boolean ignoreCase, int startIndex, String str2,  
int str2startIndex, int numChars)

←  
If true, case is ignored

### Methods for Searching Strings

xv) indexOf( )


- Searches for the 1<sup>st</sup> occurrence of a character or a substring.
- Returns the index at which the character was found or returns -1, if the character is not found.

xvi) lastIndexOf( )

- Searches for the last occurrence of a character or a substring.
- Returns the index at which the character was found or returns -1, if the character is not found.
- Would not create any error even if negative index is given.

4 variations of this method

Searches for Character	Searches for Substring
<p>A)</p> <pre>int indexOf(int ch) int lastIndexOf(int ch) (eg) int n="buffer".lastIndexOf(' f '); n=3 n="buffer".indexOf(' f '); n=2</pre> <p>Note: The datatype for the 1<sup>st</sup> parameter is int. So, either a character or its ASCII value can be given, to search for.</p>	<p>B)</p> <pre>int indexOf(String str) int lastIndexOf(String str) (eg) String s="to be or not to be"; S.o.p(s.indexOf("to")); → 0 S.o.p(s.lastIndexOf("to")); → 13</pre>
<p>C &amp; D,</p> <p>The search runs from startIndex to the end of the string in case of indexOf( )</p> <p>In case of lastIndexOf( ), the search runs from startIndex to zero</p>	
<p>C)</p> <pre>int indexOf(int ch, int startIndex) int lastIndexOf(int ch, int startIndex) (eg) String s="Be good do good"; S.o.p(s.indexOf('o',7)); → 9 S.o.p(s.lastIndexOf('o',11)); → 9</pre>	<p>D)</p> <pre>int indexOf(String str, int startIndex) int lastIndexOf(String str, int startIndex) (eg) String s="Be good do good"; S.o.p(s.indexOf("good",7)); → 11 S.o.p(s.lastIndexOf("good",10)); → 3</pre>

 <p>This will search in the string starting from 11<sup>th</sup> location and go towards the beginning of the string.</p>	
--	--

### Methods for Modifying a String:

If a string has to be modified, it must be copied into a StringBuffer or StringBuilder or a String method which constructs a new string with the modifications, must be used.

xvii) substring( )

a) String substring(int startIndex)

Extracts all characters from startIndex till the end

b) String substring(int startIndex, int endIndex)

The substring contains characters from the beginning index, up to, but not including the end index.

(eg) To replace all occurrences of a substring by another string.

```
String orig="This is a String. This is an object too";
```

```
String search="is", replacestr="was", result=" ";
```

```
int i;
```

```
do
```

```
{
```

```
i = orig.indexOf(search);
```

→stmt A

```
if(i!=-1)
```

```
{
```

```
result=orig.substring(0,i);
```

→stmt B

```
result+=replacestr;
```

→stmt C

```
result+=orig.substring(i+search.length( ));
```

→stmt D

```
orig=result;
```

→stmt E

```
System.out.println(orig);
```

```
}
```

```
}while(i!=-1);
```

Iteration 1:

The substring 'is' is searched for, in the original string.

Therefore, i=2 (Since, 'this' contains 'is')

A new string is constructed by copying all characters from the beginning till (i=2).

Therefore, result="Th" according to stmt B.

"is" is to be replaced with "was". So stmt C appends "was" to result. Therefore, result="Thwas".

In stmt D, leaving the 2 characters "is", the remaining part of the original string is appended to result.

This is copied to orig. therefore, O/P: Thwas is a String. This is an object too

In the next iteration, this orig string is again searched for the occurrence of "is". Now, i=6

After replacing, O/P: Thwas was a String. This is an object too

Iteration 3: Thwas was a String. Thwas is an object too

Iteration 4: Thwas was a String. Thwas was an object too

Iteration 5: No Output since (I == -1), this time

xviii) concat( )

- Similar to '+'  
String concat(String str)  
→ Creates a new object that contains the invoking string with the contents of the string appended to the end.  
(eg) String s1="one";  
String s2="two";  
String s3=s1.concat(s2); → onetwo  
Note: s1 will not have the concatenated String.

xix) replace( )

- Replaces all occurrences of a given char by another char.  
a) String replace(char original, char replacement)  
b) String replace(CharSequence original, CharSequence replacement)

←  
It is an interface which is implemented by String, StringBuffer and StringBuilder classes. So, any of these can be passed as argument to this method.

(eg) String result="Hello".replace('l','w'); → Hewwo

String result="to be or not to be".replace("to","too");

→ too be or not too be

The previous example program to replace all occurrences of a substring by another, can be done using replace( ) method.

String result = orig.replace(search,replacestr);

xx) trim( )

- Returns a copy of the invoking String from which any leading and trailing whitespaces are removed.

String trim( )

(eg) String str=" Hello world Program ";

str=str.trim( );

O/P: Hello world Program

xxi) changing the case

- a) String toLowerCase( ) – converts all characters in a string from upper to lowercase
- b) String toUpperCase( ) – converts all lowercase characters to uppercase characters
- (eg) String upper="Hai".toUpperCase( ); → HAI  
String lower="Hai".toLowerCase( ); → hai

xxii) split( )

- a) String[ ] split(String regexp)

Decomposes the invoking string into parts delimited by the regular expression passed in 'regexp'. It returns an array of String objects containing these parts.

(eg) String s="to be or not to be";

String[ ] tokens=s.split(" "); → Here the delimiter is space character

for(int i=0;i<tokens.length;i++)

```
System.out.println(tokens[i]);
```

Output: to

be

or

not

to

be

b) `String[ ] split(String regexp, int max)`

If max is negative or zero, the invoking string is fully decomposed. If it is positive, then the string has to be split up into 'max' number of parts, where the last part contains the remaining portion of the string.

(eg) `String[ ] tokens="to be or not to be".split(" ", 3);`

`for(String s : tokens)`

```
System.out.println(s);
```

Ouput: to

be

or not to be



Now the invoking string is split up into exactly 3 parts and stored in the `String[ ]` array. The last part contains the remaining portion of the array.

Note: For example, if an IP address has to be split up with '.' as the delimiter, give the regular expression as `"\\."` instead of `"."`. This is because '.' is a meta character.

A meta character is a character with special meaning interpreted by the matcher. The metacharacter "." means "any character". Other meta characters are: `<([\\^-= $! | ] ) ? * + >`

There are two ways to force a metacharacter to be treated as an ordinary character:

- precede the metacharacter with a backslash, or
- Use `Pattern.quote( )` method to enclose it within `\Q` (which starts the quote) and `\E` (which ends it).

Why double backslash?

From the above list, it can be seen that `"\"` (backslash) is itself a meta-character used to specify that the next character is either a special character, a literal, a back reference, or an octal escape. i.e. `\.` denotes that dot is a meta character. Now in our example we do not want '.' to be treated as metacharacter. So, we have to escape the backslash. To escape any character in general, we use `\"`. Thus, `\"` escapes the metacharacter `\` which would otherwise make '.' character to be used as a metacharacter.

Eg., `String[ ] tokens="10.10.131.132".split("\\.");`

`for (String s : tokens)`

```
System.out.println(s);
```

(Or)

`import java.util.regex.*;` → as `Pattern` class is defined in 'regex' sub-package of 'util' package

`String[] s="10.10.131.132".split(Pattern.quote("."));`

.....

## xxiii) valueOf( )

- A static method in String class that is overloaded for all of Java's built-in types, so that each type can be converted to its String representation.

static String valueOf(double num)

static String valueOf(long num)

...

static String valueOf(Object ob)

static String valueOf(char[ ] chars)

Difference between valueOf( ) and toString( )

valueOf( )	toString( )
i) – For primitive data types, it gives the String representation of those values. – For objects, it invokes the toString( ) method and gives the result returned by it.	Gives the String representation of <u>objects</u> passed to it.
ii) Static method	Non-static method

For Primitive Data Types:

Directly gives the string representation of the data.

(eg) String s=String.valueOf(5.5);

System.out.println(s);      ➔ 5.5(now it is string "5.5")

For Non-Primitive Data Types:

Consider the example program below

class StringClass

```
{
    String s = new String("Java");
    void display( )
    {
        System.out.println(s);
    }
    public String toString( )
    {
        String temp;
        temp="The instance variable is the object" + s;
        return temp; ➔ the method should return a 'String'
    }
}
```

class toStringMain

```
{
    public static void main(String[ ] args)
    {
        StringClass sc = new StringClass( );
        String ss=String.valueOf(sc);
        System.out.println(ss); ➔ O/P: The instance variable is the object Java
        toStringMain ts=new toStringMain( );
    }
}
```

```
String sss=String.valueOf(ts);
System.out.println(sss);    O/P: toStringMain@18fb65
    }
}
```

For the class 'StringClass', the toString( ) method is overridden. For the main class 'toStringMain', the toString( ) method is not redefined.

As there is a overridden toString( ) method for this class, only that will be invoked, by the valueOf( )

As there is no overridden toString( ) method for the main class, it invokes the one that is in Object class.

## **StringBuffer Class**

- Represents growable character sequences.
- A character or substring may be inserted in the middle or appended to the end.
- StringBuffer will automatically grow to make room for such additions.

### Constructors:

#### i) StringBuffer( )

- Reserves space for 16 characters, without reallocation.
- StringBuffer does not allocate space for each character (as it comes) individually. Rather, it allocates space for a few extra characters and thus reduces the number of reallocations that take place.
- Reallocation should not happen quite frequently for 2 reasons
  - a) It is a costly process in terms of time.
  - b) Frequent reallocations can fragment memory.
- For each reallocation, the capacity is increased to twice the current capacity +2.

StringBuffer sb=new StringBuffer( );

    ↘ This creates a StringBuffer object which can hold 16 characters and will go for reallocation, if more characters are added. Its reference is stored in the reference variable 'sb'.

#### ii) StringBuffer(int size)

- Sets the default size specified by 'size' instead of 16.

StringBuffer sb=new StringBuffer(10);

    ↘ The initial capacity of the object is 10 characters

#### iii) StringBuffer(String str)

- StringBuffer object is created from the existing string 'str' and reserves room for 16 more characters without reallocation.

#### iv) StringBuffer(CharSequence chars)

- Creates an object that contains the character sequence contained in chars and reserves room for 16 more characters.



Methods similar to those in String class

- 1) length( ) – gives the current length of a StringBuffer.
- 2) charAt(int index) – gives the character in the location specified by index.
- 3) void getChars(int sourceStart, int sourceEnd, char[ ] target, int targetStart)
- 4) String substring(int startIndex)  
String substring(int startIndex, int endIndex)
- 5) int indexOf(String str)  
int indexOf(String str, int startIndex)  
int lastIndexOf(String str)  
int lastIndexOf(String str, int startIndex)

Other Methods

- 6) capacity( ) – gives the total number of characters that can be stored in the StringBuffer object, at present.
- 7) setLength( ) – to set the length of the StringBuffer object.  
void setLength(int len)  
If 'len' increases the size of the String, null characters are added to the end.  
If 'len' is less than the current length of the String, the characters stored beyond the new length will be lost
- 8) setCharAt( )  
void setCharAt(int index, char ch)  
index – specifies the position of the character to be set.  
ch – specifies the new value of the character  
(eg) StringBuffer sb=new StringBuffer("Hello");  
sb.setCharAt(1,'i');     → Hillo  
sb.setLength(2);             → Hi  
System.out.println(sb);     → O/P: Hi
- 9) append( )
  - Concatenates the string representation of any other data type to the end of the invoking StringBuffer object.  
Some overloaded versions:  
StringBuffer append(String str)  
StringBuffer append(Object obj)  
StringBuffer append(int num)
  - Subsequent calls can be chained together.  
(eg 1) StringBuffer sb=new StringBuffer(10);  
sb.append("a=").append("Hello").append("?").toString( );  
System.out.println(sb);  
O/P: a=Hello?
  - (eg 2) StringBuffer sb=new StringBuffer(10);  
sb.append("Java");  
System.out.println(sb.length( )); → 4  
System.out.println(sb.capacity( )); → 10  
sb.append("Programming");

```
System.out.println(sb.length( )); ➔ 15 ➔ count of the characters
System.out.println(sb.capacity( )); ➔ 22 ➔ as the original capacity is
    not sufficient to append the string "Programming",
    the capacity will be increased to twice the original
    capacity + 2 ➔  $10 * 2 + 2 = 22$ 
sb.append("Laboratory");
System.out.println(sb.length( )); ➔ 25
System.out.println(sb.capacity( )); ➔ Original capacity (22) would not have
    been sufficient. So,  $22 * 2 + 2 = 46$  will
    be the new capacity
```

#### 10) ensureCapacity( )

- To set the size of the buffer.  
void ensureCapacity(50);

➔ It is the minimum size of the buffer. But a larger  
Buffer will be allocated (current capacity \* 2 + 2)

ie., if already the capacity is more than 50, it will not be reduced to 50. But if the capacity was 30 earlier, upon executing this statement, it will be changed to 50. It will be ensured that the capacity is at least 50.

#### 11) insert( )

- To insert one string into another.
  - StringBuffer(int index, String str) ➔ inserts the String str at position 'index'
  - StringBuffer(int index, char ch) ➔ inserts the char ch at position 'index'
  - StringBuffer(int index, Object obj)

```
(eg) StringBuffer sb=new StringBuffer("I like Programming");
sb.insert(2,"always");
System.out.println(sb); ➔ O/P: I always like Programming
```

#### 12) reverse( )

```
(eg) StringBuffer sb=new StringBuffer("abcdef");
sb.reverse();
System.out.println(sb); ➔ O/P: fedcba
```

#### 13) delete( )

- StringBuffer delete(int startIndex, int endIndex)  
All characters from startIndex upto endIndex-1 will be deleted.
  - StringBuffer deleteCharAt(int index)  
Deletes the character at the location specified by index.
- ```
(eg) StringBuffer sb=new StringBuffer("I always like Programming");
sb.delete(2, 8); ➔ I like Programming
sb.deleteCharAt(3);
System.out.println(sb); ➔ O/P: I lie Programming
```

## 14) replace( )

StringBuffer(intstartIndex, intendIndex, String str)

The substring from startIndex upto endIndex – 1 is to be replaced by 'str' String.

(eg) StringBuffer sb=new StringBuffer("I like Programming");

sb.replace(2, 6, "love");

System.out.println(sb);      → O/P: I love Programming

## **StringBuilder Class**

- Similar to StringBuffer class
- But not synchronized (so, not thread-safe).
- Faster performance.
- In situations where mutable strings will be accessed by multiple threads and no external synchronization is provided, it is better to use StringBuffer than StringBuilder.
- Constructor
  - i) StringBuilder( )
  - ii) StringBuilder(String str)
- Methods
  - append( ), capacity( ), charAt( ), getChars( ), indexOf( ), lastIndexOf( ), insert( ), replace( ), reverse( ), toString( ), setCharAt( ), substring( ).

### Example Programs related to Strings:

#### 1) Removing duplicates from a String.

```
class Remove_duplicates
{
    public static void main(String[] args)
    {
        String s=args[0];
        int len=s.length();
        int c;
        String org=s, s1="";
        for(int i=0;i<(len-1);i++)
        {
            s1=s.substring(0,i+1); → Store only distinct characters in s1. For the
            c=0;                      1st iteration, 'c' alone is stored. In the inner
            for(int j=i+1; j<len;j++)   for loop, characters other than 'c', get
            {                          accumulated in s1. So, at the end of 1st
                if(s.charAt(i)==s.charAt(j)) iteration of inner 'for' loop, there
                {                          won't be any duplicates of zeroeth
                    c++;                  character.
                    continue;            (eg) character
                }                        In iteration 2, of 'i' loop, s1="ch"
                else                      & inner loop copies all characters
                    s1=s1+s.charAt(j);   other than 'h' into s1. This
            }                            continues.
            s=s1;
        }
    }
}
```

```

        s1="";
        if(c>0)
            len -=c;
        }
        System.out.println("Original String: "+org);
        System.out.println("String after removing duplicates: "+s);
    }
}
Output: Z:\> java Remove_duplicates character
        Original String: character
        String after removing duplicates: charte

```

2) Finding the number of occurrences of a given character in a string.

```

Scanner sc = new Scanner(System.in);
String s = sc.next( );
char c = sc.next( ).charAt(0);
int count = 0;
s.toLowerCase( );
for(int i=0;i<s.length( );i++)
{
char x=s.charAt(i);
if(x== c)
count++;
}
System.out.println("Frequency of the letter "+c+" in the sentence "+s+" is "+count);

```

3) Program which will display only the initials, given first name, middle name & surname.

(eg) AmitParashuramKesarkar → A.P.K

```

import java.util.*;
class Surname
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the name");
        String s=sc.nextLine();→ If next( ) method is used, only the first name
        String s1 = "";           will be read.
        s = " " + s; //to insert a space char in the beginning
        int len = s.length( );
        char a;
        for(int i=0;i<len;i++)
        {
            a = s.charAt(i);
            copy the character next to space

```

```

        if(a == ' ')                character
            s1 = s1 + s.charAt(i+1) + ".";
    }
    System.out.println("The initials are" + s1);
}
}

```

- 4) Program to print surname first, followed by the first name & middle name.

```

Scanner sc=new Scanner(System.in);
String s = sc.nextLine( );
int first=s.indexOf(' ');
int last= s.lastIndexOf(' ');
String sur=s.substring(last+1);→ extracts from 'last+1' position till the end
String middle= s.substring(first+1, last);→upto last-1 char
String firstname= s.substring(0, first);→upto first-1 char
System.out.println(sur+" "+firstname+" "+middle);

```

- 5) Reversing a String.

```

String s = args[0], backward = " ";
for(int i=s.length( ) - 1; i>=0; i++)
    backward += s.charAt(i);
System.out.println(backward);

```

- 6) Program to remove vowels from a String.

```

String s = args[0];
String s1 = " ";
for(int i=0;i<s.length( );i++)
{
    ch = s.charAt(i); // can next use ch = Character.toLowerCase(ch);
    switch(ch)
    {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': break;
        default: s1 = s1 + ch;
    }
}
System.out.println(s1);
Input: remove Output: rmv

```

- 7) Program to print the pattern

```

String s = args[0];
int len = s.length( );
for(int i=len;i>0;i--)

```

```

→ SUPER
  SUPE
   SUP
    SU

```

```
System.out.println(s.substring(0,i));
```

 S

8) Program to print the pattern

```
String s = args[0];
int len = s.length( );
for(int i=0;i<len;i++)
{
    char a = s.charAt(i);
    for(int j=0;j<=i;j++)
    {
        System.out.print(a);
    }
    System.out.println( );
}
```

→ SSSSS  
 UUUU  
 PPP  
 EE  
 R


8) Program to print the index of all occurrences of 'b' after 'a'.

```
Scanner sc = new Scanner(System.in);
String s = sc.nextLine( );
int aind, bind;
aind = s.indexOf('a');
while(aind != -1)
{
    bind = s.indexOf('b', aind+1);
    System.out.println(bind);
    aind = s.indexOf('a', bind+1);
}
```

Output: Cab lobby slab → 2  
 13

9) Count the number of "and" and "the" words in a given string.

```
int andcount = 0, thecount = 0;
String text = "Time and tide wait for no man"
            + "The thirsty crow and silly goat"
            + "Stories are the stories repeated in"
            + "std1 and std2";
int index = text.indexOf("and");
while(index != -1)
{
    andcount++;
    index = text.indexOf("and", index + "and".length( ));
}
index = text.lastIndexOf("the");
while(index != -1)
{
    thecount++;
    index = text.lastIndexOf("the", index - "the".length( ));
}
```

move 3 char ahead & start searching from there  


```
System.out.println(andscount + " " + thecount);
```

10) Program to remove all the occurrences of a given character.

```
String s = args[0];
char rem = 'n';
char[] chars = s.toCharArray();
int len = s.length(), uniquecount = 0;
for(int i=0; i<len; i++)
{
    if(chars[i] != rem)
        chars[uniquecount++] = chars[i];
    else
        continue;
}
System.out.println(new String(chars, 0, uniquecount));
```

↙ Builds the string from CharArray

Output: entertainment → etertaimet

11) Replace all consonants in a given string with the next alphabet.

```
Scanner sc = new Scanner(System.in);
String s = sc.nextLine();
String ss = " ";
for(int i=0; i<s.length(); i++)
{
    char a = s.charAt(i);
    switch(a)
    {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': ss += a; break;
        default: ss = ss + (char)(a+1);
    }
}
System.out.println(ss);
```

12) Read a mail id and store the substrings in an array of strings.

```
class mailid
{
    public static void main(String[] args)
    {
        String s = "anu@vit.ac.in";
        String[] ss = new String[4];
        String temp = " ";
```

```

        int index1 = s.indexOf('@');
int index;
temp += s.substring(0, index1);
ss[0] = temp;
int i = 1;
    do
    {
        index = s.indexOf('.', index1+1);
        if(index == -1)
            break;
        ss[i++] = s.substring(index1+1, index);
        index1 = index;
    } while(index != -1);
    ss[i] = s.substring(index1+1);
    for(int i=0; i<4; i++)
        System.out.println(ss[i]);
    }
}

```

## Wrapper Classes

- Defined in java.lang package.
- Primitive data type, although not implemented as objects, for performance considerations, their corresponding object representations are needed for the following reason:  
There are collection classes that deal only with objects. So, to store a primitive data type in one of these classes, we need to wrap the primitive type in a class.
- Thus, for all the 8 primitive data types in Java, equivalent Wrapper Classes are there.

| Data type  | Wrapper Class     |                                                                                                                                                                     |
|------------|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1) boolean | <u>B</u> oolean   | Note that the class names start with capital letter.<br>Except Boolean & Character, all other classes are subclasses of an abstract class called ' <u>N</u> umber'. |
| 2) byte    | <u>B</u> yte      |                                                                                                                                                                     |
| 3) char    | <u>C</u> haracter |                                                                                                                                                                     |
| 4) short   | <u>S</u> hort     |                                                                                                                                                                     |
| 5) int     | <u>I</u> nteger   |                                                                                                                                                                     |
| 6) long    | <u>L</u> ong      |                                                                                                                                                                     |
| 7) float   | <u>F</u> loat     |                                                                                                                                                                     |
| 8) double  | <u>D</u> ouble    |                                                                                                                                                                     |



- To build an object for a given primitive data type, all these classes have appropriate constructors defined. There are 2 types of constructors for all the classes except Character.

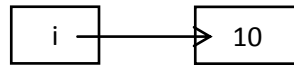
#### I) Constructor – I

Takes the corresponding type of numeric or boolean (for Boolean class) values as arguments and creates the object.

```
Double dd = new Double(double);
Float ff = new Float(float);
Short ss = new Short(short);
Long ll = new Long(long);
Integer ii = new Integer(int);
Byte bb = new Byte(byte);
Boolean bool = new Boolean(boolean);
(eg)
```

1) Integer I = new Integer(10);

Now 'i' is a reference to the object containing the integer value 10.



2) Double d1 = new Double(10.4f);

Double d2 = new Double(10.4);

3) Boolean bool = new Boolean(true);

#### II) Constructor – II

It constructs the object from the string representation of a numeric data or boolean data.

```
Double dd = new Double("110.5");
Long ll = new Long("150678924");
...
```

Byte bb = new Byte(args[0]);

→ If "100" is given as the command line argument, which is in the form of a String, it can be converted to an object.

#### Character Class Constructor:

This class provides only one constructor which takes a character as the argument.

```
Character cc = new Character(char ch);
(eg) Character cc = new Character('E');
```

All the 7 classes (excluding Character class) have overloaded the following abstract methods of 'Number' class.

Most basic methods.

|   | Methods                                      | Usage                                                                                                                                                  | Description                                                                                                                                                              |
|---|----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | static valueOf( )<br>return type<br>→ object | Integer.valueOf(int n)<br>Float.valueOf(float f)<br>Double.valueOf(double d)<br>Long.valueOf(long l)<br>Short.valueOf(short s)<br>Byte.valueOf(byte b) | Static methods. So, invoke them using the corresponding class names. <u>They covert the numeric or boolean value passed to them into their corresponding objects and</u> |

|   |                                                                                                             |                                                                                                                                                                                                 |                                                                                                                                                                        |
|---|-------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   |                                                                                                             | Boolean.valueOf(boolean bool)<br>The arguments can even be a String representation of the relevant data.                                                                                        | <u>return those objects.</u>                                                                                                                                           |
| 2 | xxxValue( )<br>where,<br>xxx refers to a data type<br><br>Return type is the respective primitive data type | ii.intValue( ) where, 'ii' is an integer object<br>dd.doubleValue( ) → 'dd' is an object<br>ff.floatValue( )<br>ss.shortValue( )<br>ll.longValue( )<br>bb.byteValue( )<br>bool.booleanValue( )  | Non-static methods. So, invoked using objects.<br>They return the value of the invoking objects in the form of their equivalent primitive data type.                   |
| 3 | static parseXXX( )<br>Return type<br>→ Respective primitive data types                                      | Integer.parseInt(s)<br>Long.parseLong(s)<br>Short.parseShort(s)<br>Byte.parseByte(s)<br>Float.parseFloat(s)<br>Double.parseDouble(s)<br>Boolean.parseBoolean(s)<br><u>where 's' is a String</u> | Static methods. So, invoked using the corresponding class names.<br>Convert the String representation of the numbers into their equivalent primitive data type values. |
| 4 | a)<br>String toString( )<br>Return type → is a String                                                       | dd.toString( )<br>ll.toString( )<br>ii.toString( )<br>ff.toString( )<br>ss.toString( )<br>bb.toString( )<br>bool.toString( )                                                                    | Returns the String representation of the invoking objects.                                                                                                             |
|   | b)<br>static String toString(datatype)<br>Return type → is a String                                         | Integer.toString(int i)<br>Long.toString(long l)<br>Short.toString(short s)<br>Byte.toString(byte b)<br>Float.toString(float f)<br>Double.toString(boolean bool)                                | Returns the String equivalent of the numeric or boolean values passed as arguments. Static methods, so invoked using their corresponding class names.                  |

Important Note:

Primitive into object conversion → valueOf( )

Object into primitive conversion → xxxValue( )

String representation of primitive into primitive → parseXXX( )

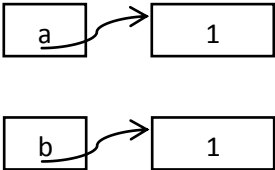
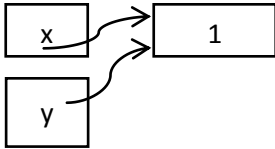
Primitive (4b) (or) object (4a) into String → toString( )

We have seen that the constructors of these various Wrapper Classes, also create objects from their primitive type values.

Then, why should there be a method valueOf( ) to convert primitive to object?

Though both the constructor and the valueOf( ) method convert the primitive type values into objects, there is a subtle difference between these two.

|                                     |                                        |
|-------------------------------------|----------------------------------------|
| Constructor<br>(eg) new Integer(50) | valueOf( )<br>(eg) Integer.valueOf(50) |
|-------------------------------------|----------------------------------------|

|                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Creates a new object for each call.<br/>         (eg) <code>Integer a = new Integer(1);</code><br/> <code>Integer b = new Integer(1);</code><br/> <code>System.out.println(a == b);</code> → false</p> <p>This is because 2 different objects will be created though both have the same content.</p>  | <p>From JDK 5+ onwards,<br/>         Integer classes caches (stores)<br/>         Integer objects from (-127 to 128)<br/>         Byte objects from (-127 to 128)<br/>         Boolean objects for 'true' and 'false', etc.<br/>         Thus this method gives back the same exact object every time instead of wasting an object construction on a brand new identical object.</p> <p><code>Integer x = Integer.valueOf(1);</code><br/> <code>Integer y = Integer.valueOf(1);</code><br/> <code>System.out.println(x == y);</code> → true</p>  <p>However, for<br/> <code>Integer c = Integer.valueOf(1000);</code><br/> <code>Integer d = Integer.valueOf(1000);</code><br/> <code>System.out.println(c == d);</code> → false<br/>         Because only up to limit +127, cached objects are used. Beyond that, new objects will be created.</p> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### Autoboxing and Auto-Unboxing

#### Autoboxing:

Converting a primitive type into the object of the corresponding wrapper class, automatically.

The Java compiler applies autoboxing when a primitive type is

- i) Passed as a parameter to a method that expects an object of the corresponding wrapper class
- ii) Assigned to a variable of the corresponding wrapper class

Eg., 1) `Integer iob=100;`

`iob` → Object of Integer class    `100` → primitive type data

The compiler autoboxes the primitive type integer value '100' into an Integer object, so that it can be assigned to the object 'iob'.

2) class wrapper

```
{
void display(Integer ii)
{
System.out.println(ii);
}
public static void main(String[] args)
{
wrapper.display(100); → An integer is passed but the parameter in the method
int j=25;              signature is an Integer object.
}
```

```
wrapper.display(j);
}
```

Thus integer value '100' and integer variable 'j' will be automatically converted, into the Integer object and so will compile successfully

Note:

In both the scenarios, autoboxing invokes the valueOf( ) method, for doing this conversion.

#### Auto-Unboxing:

Converting an object of wrapper type to its corresponding primitive value, automatically.

Unboxing is done during the following situations:

- i) When an object is passed as a parameter to a method which expects a primitive type value.
- ii) When an object is assigned to a variable of the primitive type.

(eg) Integer iob = Integer.valueOf(50);

int i = iob; → 'iob' is an object of Integer class. When it is assigned to an integer variable, it is unboxed to primitive data type, before getting assigned to the variable.

xxxValue( ) method is used implicitly when auto-unboxing takes place. (intValue( ), floatValue( ), shortValue( ), etc).

Use of the wrappers should be restricted to only those cases in which object representations of a primitive type is required, in spite of having autoboxing/auto-unboxing facility.

Difference between parseXXX( ) & valueOf( ) methods:

| parseXXX( )                                                                                                    | valueOf( )                                                                                                                         |
|----------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| Returns a <u>primitive type data</u> from the given string.<br>(eg) int x = Integer.parseInt("10");<br>↓<br>10 | Returns an <u>object</u> from a primitive type data or a string representation of the data.<br>Integer ii = Integer.valueOf("10"); |

Apart from the 4 methods discussed earlier, there are other methods which may be useful while working with the objects of wrapper classes.

Following 3 methods are overloaded in all the 7 classes.

- 5) int compareTo(object of Wrapper class) – Returns '0', if the numeric value of the corresponding objects are equal. Returns '-ve' value if the invoking object has a greater value than that of the object passed as argument.

For Boolean class, it returns a '+ve' value if the invoking object is true & the argument object is false. Otherwise it returns a false.

(eg) 1) Short ss1 = new Short(120);

      Short ss2 = new Short(150);

      System.out.println(ss1.compareTo(ss2)); → negative value

      2) Boolean b1 = new Boolean(true);

      Boolean b2 = new Boolean(false);

      System.out.println(b1.compareTo(b2)); → positive value

- 6) boolean equals(obj) – returns true, if the invoking object is equivalent to argument object.  
Otherwise, false.  
(eg) ss1.equals(ss2); → false
- 7) int hashCode() – returns the hash code for the invoking object.

Methods in Integer, Long, Float & Double:

- 8) static String toHexString(num)  
→ Primitive type data
- Returns a string containing the value of 'num' in hexadecimal format.
- (eg) System.out.println(Integer.toHexString(10)); → a  
System.out.println(Float.toHexString(10.12f)); → 0x1.43d70ap3  
System.out.println(Double.toHexString(10.12)); → 0x1.43d70ad70a3dp

Methods only in Integer & Long:

- 9) static String toBinaryString(int num)  
static String toBinaryString(long num)
- Returns the String that contains the binary equivalent of num.
- (eg) System.out.println(Integer.toBinaryString(4)); → 100
- 10) static String toOctalString(int num)  
static String toOctalString(long num)
- Returns the string that contains the octal equivalent of num.
- (eg) System.out.println(Integer.toOctalString(8)); → 10

Methods only in Float & Double:

- 11) a) boolean isInfinite()  
- Returns true if the invoking object contains an infinite value. Else, returns false.  
(eg) Double d1 = new Double(1 / 0.0); → do not give (1/0) as it will perform integer Division & will end up in Exception.  
System.out.println(d1.isInfinite()); → true  
This does not happen in 'Float' or 'Double' as there are equivalent constants defined for Infinity in these classes.
- b) static boolean isInfinite(float num)  
static Boolean isInfinite(double num)
- Checks if the num specifies an infinite value. If so, returns true. Else, false.
- 12) a) Boolean isNaN()  
- true if the invoking object contains a value that is not a number.  
(eg) Double d = new Double(0 / 0.0);  
System.out.println(d.isNaN()); → true
- b) static Boolean isNaN(float num)  
static boolean isNaN(double num)
- true if the num is a value that is not a number.
- (eg) System.out.println(Double.isNaN(0 / 0.0)); → true

Defined Constants in these Classes:

|                |                                                                  |
|----------------|------------------------------------------------------------------|
| Float & Double | MAX_EXPONENT → Maximum Exponent<br>MAX_VALUE → Maximum +ve value |
|----------------|------------------------------------------------------------------|

|                            |                                                                                                                                                                                                        |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                            | MIN_EXPONENT → Minimum Exponent<br>MAX_EXPONENT → Maximum Exponent<br>MIN_VALUE → Minimum Positive Value<br>NAN → Not a Number<br>POSITIVE_INFINITY → +ve Infinity<br>NEGATIVE_INFINITY → -ve Infinity |
| Byte, Short, Integer, Long | MIN_VALUE → Minimum -ve Value (-2,147,483,648)<br>MAX_VALUE → Maximum +ve Value                                                                                                                        |
| Boolean                    | TRUE → true<br>FALSE → false                                                                                                                                                                           |

## Character Class

- Only one constructor – Character(char ch)  
 Character cc = new Character('x');
- To obtain the char value contained in a Character object,  
 → char charValue( ) – returns a character  
 (eg) System.out.println(cc.charValue( )); → x
- All methods of Character class specified below are static methods.
  - i) static boolean isDigit(char ch)
  - ii) static boolean isLetter(char ch)
  - iii) static boolean isLetterOrDigit(char ch)
  - iv) static boolean isWhiteSpace(char ch)
  - v) static boolean isLowerCase(char ch)
  - vi) static boolean isUpperCase(char ch)
  - vii) static boolean isTitleCase(char ch) → returns true if ch is a Unicode TitleCase
  - viii) static char toLowerCase(char ch)                      character
  - ix) static char toUpperCase(char ch)
  - x) static char toTitleCase(char ch) → returns the title case equivalent of ch
  - xi) static char forDigit(int num, int radix) → returns the number/character equivalent of num, in the specified radix.  
 (ie., decimal to specified radix(base))
  - xii) static int digit(char ch, int radix) → returns the integer value associated with the char specified by the radix  
 (ie., specified radix to decimal)

(eg) Character.forDigit(15, 16); → f (character equivalent of 15 in radix (base) 16 is f)  
 System.out.println(Character.forDigit(1, 10)); 1(number equivalent 1 in radix 10 is 1)  
 System.out.println(Character.digit('e',16)); → 14(the decimal equivalent of hexadecimal character 'e' is 14)  
 System.out.println(Character.digit('1', 10)); → 1
- Write a program to count the number of consonants, letters & spaces in the given text.  
 String text = "Java Programming has to be learnt for its most important feature of"  
                   + "platform independence";  
 int spaces = 0, vowels = 0, letters = 0;  
 int len = text.length( );  
 for(int i=0;i<len;i++)

```

{
char ch = Character.toLowerCase(text.charAt(i));
if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
    vowels++;
if(Character.isLetter(ch))
    letters++;
if(Character.isWhiteSpace(ch))
    spaces++;
}
System.out.println("Vowels=" + vowels + " Consonants=" + letters-vowels
    + " Spaces=" + spaces);

```

Note:

Primitive Wrapper classes are **immutable** like String class. If you attempt to change the value of (say) an Integer object, the original object will not be modified; instead a new object will be created.

Example to demonstrate that:

```
Integer i=new Integer(5);
```

```
Integer j=i;
```

```
System.out.println(j==i); // true, as at present, both store the same reference
```

```
i=6;
```

```
System.out.println(j==i); // if the same object is modified to hold 6, the reference of
                          ↙ 'i' should still remain the same. Hence, the result should be
                          True. But as the result is false, it proves that a new object
                          False is created. So, primitive wrapper classes are immutable.
```

## Vector Class

- Defined in java.util package. So, import java.util.Vector;
  - A vector is a collection of elements of type Object.
  - Works like an array but can grow automatically when more capacity is needed.
  - In a vector, any type of Object can stored.
- Objects that are instances of a variety of classes can also be stored.

### 3 basic types of Constructors:

i) Vector v = new Vector( );

A vector of default size 10 is created and its size is doubled, if an object is added when the vector has become full.

ii) Vector v = new Vector(100);

↘ int size

A vector of size 100 is created and its size will be doubled when the vector becomes full.

iii) Vector v = new Vector(100, 10);

The size of the vector, whose initial capacity is 100, will be increased by 10, each time when the vector becomes full. Thus, it is efficient than doubling the capacity each time.

Note:

i) `Vector<String> v=new Vector<String>();`

ii) `Vector<Computer>v1=new Vector<Computer>();`

The above statements can also be used to create Vector objects. But in the first case, v can store only String objects and in the second case, v1 can store only Computer objects.

So, `Vector v=new Vector();` will allow us to store any type of object. But this will show 2 warning messages while compiling, as it is deprecated. Still, the program can be executed.

#### 1) get() Method

- Used to retrieve the reference from the Vector.
- Should type cast the object reference returned by this method, to the type, equivalent to the object type to which this will be assigned. However, in `println()`, conversion is implicitly done.

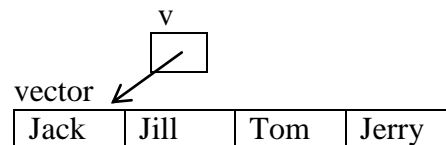
`get(int index)` → the index of the element to be retrieved from the Vector

2) `void add(obj)`  
`void addElement(obj)` } → adds the object reference 'obj' to the vector, after any other existing objects in V & increases the size by 1

3) `size()`

- Gives the number of elements that are occupied by the objects in a vector.

```
(eg) String[] s = {"Jack","Jill","Tom","Jerry"};
      Vector v = new Vector();
      for(int i=0;i<s.length;i++)
        v.add(s[i]);
      for(int i=0;i<v.size();i++)
        System.out.println((String)v.get(i));
```



←  
 Type cast the reference to String. This casting is especially needed when the return value of `get()` is assigned to the object.

4) `capacity()`

- Gives the maximum number of objects, the vector can hold at any instant.

Example:

Write a program to add 20 Integer objects containing values from 1 to 20. Find the capacity of the vector before and after adding these objects. At last, add another Integer object containing the value '100' & find the size and capacity of the vector.

class vector

```
{
public static void main(String[] args)
{
    Vector v = new Vector();
    System.out.println("Initial Capacity" + v.capacity()); → default 10
    for(int i=0;i<20;i++)
    {
        Integer io = new Integer(i+1); → creates 20 Integer objects within the loop
    }
}
```



```

        v.add(io); → adds the objects to the vector as & when created
    }
A   System.out.println("After adding 20 objects capacity=" + v.capacity()); → 20
    v.addElement(100);
B   System.out.println("Final Capacity " + v.capacity( ) + "Size=" + v.size( ));
    }                                     → 40                               → 21
    }

```

In statement A, `v.capacity( )` gives 20 because the original capacity would not have been sufficient to store the 11<sup>th</sup> object within the 'for' loop. So, the vector capacity would have been doubled (to 20). At this point, both `size( )` and `capacity( )` would return the same value.

Next, an object with value '100' is added. As the vector is full, its capacity will once again be doubled to 40 and the object will be inserted. So, capacity is 40 & out of 40, only 21 elements are inserted. So, size is 21.

Why is the capacity of the vector increased by many elements and not in steps?

By allocating more than just the required memory, the vector reduces the number of reallocations that must take place as the vector grows, as reallocations are time consuming. (∴ For each reallocation, all the elements of the vector should be copied to a bigger vector with the new capacity). `get( )` method can be used to obtain one or more elements of the Vector.

To obtain all the elements of the Vector object, an Iterator object can be used.

Obtain a reference to an Iterator by calling the `iterator( )` method for the Vector object.

(eg) For the previous program, Iterator can be used to retrieve the data stored in the Vector.

```

    Iterator iv = v.iterator( ); → returns an Integer object which can be used for iterating
                                through the vector.

    while(iv.hasNext( ))
    System.out.println((String)iv.next( ));

```

To store an object at a particular index position:

```

v.add(2, obj);
    → inserts the object at the index 2. The index of all other objects following it will be increased
    by 1.
    → No negative value or index less than or greater than the capacity of the vector, should be
    given.

```

To change an element:

```

v.set(2, obj1);
    → returns the reference of the object previously stored, after replacing it by obj1.

```

To add all the objects from a collection:

```

v.addAll(list1);
    → adds all the objects in the collection 'list1' to the vector

```

To insert all the objects from a collection, at a particular index in the Vector:

```

v.addAll(2, list1);
    → objects of 'list1' are inserted from index 2 in the vector

```

To remove an object at a particular index:

`v.remove(3);`

→ removes the 3<sup>rd</sup> reference from 'v' and returns the reference to the object being removed.

`Class1 cobj = (Class1)v.remove(3);`

Typecasting is needed as the reference is returned as type Object.

To get the elements back as an array:

`Object[ ] data = v.toArray( );`

→ returns array of type Object

To remove a particular object:

`boolean removed = v.remove(obj);`

true if the object is found and removed.

To discard all the elements in a Vector:

`v.clear( );`

`isEmpty( )` method will return true if Vector object has zero size else false.

Setting the Vector elements to null, does not mean that the size is zero.

To determine whether the give object is in the Vector:

`boolean b = v.contains(obj1);`

→ checks whether the obj1 object is present in the vector or not.

To obtain the first element in the Vector:

`v.firstElement( );`

→ the returned object should be type casted.

To find the index of the given object:

`int p = v.indexOf(obj);`

`int p = v.indexOf(obj, startindex);`

To set the minimum capacity:

`v.ensureCapacity(150);`

→ sets the minimum capacity to 150

If the capacity is already less than 150, it will increase to 150.

If the capacity is already 150 or more than that, it will be unchanged.

To set the size:

`v.setSize(50);`

If v is less than 50 elements, the additional elements up to 50 will be filled with null references.

If v has more than 50 elements, these extra object references will be discarded.

To change the capacity to match the current size:

`v.trimToSize( );`

→ if size is 50, the capacity will be also 50. So, memory wastage is avoided.

Example: Write a program to store 5 complex numbers in a vector.

```
import java.util.*;
class vectorcomp
{
    private int r, i;
    vectorcomp( )
    {
        r = 1; i = 1;
    }
    vectorcomp(int r, int i)
    {
        this.r = r;
        this.i = i;
    }
    public String toString( )
    {
        return (r + "+" + i + "i");
    }
}
class vector_ex
{
    public static void main(String[ ] args)
    {
        Vector v = new Vector( );
        for(int i=0;i<5;i++)
        {
            vectorcomp vc = new vectorcomp(i+1, i+3);
            v.add(vc);           → 5 complex number objects are added
        }
        v.addElement(Integer.valueOf(10));
        v.addElement(Integer.valueOf(20)); } — 2 Integer objects are added
        Iterator it = v.iterator( );
        while(it.hasNext( ))
        {
            System.out.println(it.next( ));
        }
    }
}
```

Or

```
for(Object o: v)
    System.out.println(o);
As all elements in v are of type
'Object', iteration variable type
should also be the same
```

Output:

```
1+3i
2+4i
3+5i
4+6i
5+7i
10
20
```

## Calendar class

Example:

Write a program to store objects for representing the dates, in a vector. (Assume joining dates of employees have to be stored).

- defined in java.util package

'Calendar' class can be used for storing dates and time.

- Calendar is an abstract class. So, cannot instantiate it.
- But a reference to the current Instance can be obtained as follows:

```
Calendar c = Calendar.getInstance( );
```

↘ Represents the current date & time.

### Important Methods:

1) int get(int calendarfield)

where the calendar fields are DATE, MONTH, YEAR, HOUR, MINUTE, SECOND. All are static constants. So, access them as Calendar.DATE, Calendar.Hour, etc.

Example:

Calendar c = Calendar.getInstance( ); → represents the system's current date & time

```
System.out.println(c.get(Calendar.DATE)); → 22
System.out.println(c.get(Calendar.MONTH)); → 9
System.out.println(c.get(Calendar.YEAR)); → 2014
System.out.println(c.get(Calendar.HOUR)); → 9
System.out.println(c.get(Calendar.MINUTE)); → 51
System.out.println(c.get(Calendar.SECOND)); → 24
```

} Current System  
Date (date when this pgm  
was last executed)  
} Current System  
Time

2) The values can also be set, using the set( ) method.

```
void set(Calendarfield, value)
```

Example:

```
c.set(Calendar.DATE, 3);
c.set(Calendar.YEAR, 2001);
c.set(Calendar.MONTH, 1);
```

} only date, month & year are set

Now:

```
c.get(Calendar.DATE); → 3
c.get(Calendar.SECOND); → 24 → SECOND will have the current 'second' value
only as we have not set it.
```

3) boolean after(Calendar obj)

- Returns true if the invoking object contains a date that is later than the one specified by Calendar obj.

Example:

```
C1.after(C2); → true    if C1=21.12.2012 and
                        C2=21.10.2010
```

4) boolean before(Calendar obj)

- Returns true if the invoking object contains a date that is earlier than the one specified by Calendar obj.

Example:

C1.before(C2);  $\rightarrow$  true      if C1=10.10.2010 and  
C2=11.10.2011

5) boolean equals(Calendar obj)

- Returns true if the invoking object and Calendar obj have the same data (including HOUR, MINUTE & SECOND).

Program:

```
import java.util.*;
class CalendarMain
{
    public static void main(String[] args)
    {
        Vector v = new Vector( );
        Calendar c1 =Calendar.getInstance( );
        c1.set(Calendar.DATE, 3);
        c1.set(Calendar.MONTH, 1);
        c1.set(Calendar.YEAR, 2001);
        v.add(c1);
        Calendar c2 =Calendar.getInstance( );
        c2.set(Calendar.DATE, 10);
        c2.set(Calendar.MONTH, 3);
        c2.set(Calendar.YEAR, 2002);
        v.add(c2);
        Calendar c3 =Calendar.getInstance( );
        c3.set(Calendar.DATE, 13);
        c3.set(Calendar.MONTH, 3);
        c3.set(Calendar.YEAR, 2005);
        v.add(c3);
        for(int i=0;i<v.size( );i++)
        {
            Calendar c = (Calendar)v.get(i);
            System.out.println(c.get(Calendar.DATE) + "/"
                               + c.get(Calendar.MONTH) + "/"
                               + c.get(Calendar.YEAR))
        }
    }
}
```

the object retrieved should be type casted to the appropriate type while assigning

Output:

3/1/2001  
10/3/2002  
13/3/2005

Write a program to define a class student with RegNo, name, dob as its instance variables. Create an array of 3 objects and store them in the vector. Find the eldest student in the group.

```

import java.util.*;
class student
{
    private String name;
    private String rno;
    private Calendar dob;
    student(String sn, String rn, Calendar d)
    {
        name = sn;
        rno = rn;
        dob = Calendar.getInstance( );
        dob = d;
    }
    public String toString( )
    {
        return ("name" + name + "regno" + rno + "dob" + dob.get(Calendar.DATE)
                + "/" + dob.get(Calendar.MONTH) + "/" + dob.get(Calendar.YEAR));
    }
    static void elder(Vector v)
    {
        student s = (student)v.get(0);
        Calendar max = s.dob;
        for(int i=1;i<3;i++)
        {
            student temp = (student)v.get(i);
            Calendar c1 = temp.dob;
            if(c1.before(max))
                max = c1;
        }
        System.out.println(max.get(Calendar.DATE) + "/" + max.get(Calendar.MONTH)
                            + "/" + dob.get(Calendar.YEAR));
    }
}
class studentmain
{
    public static void main(String[] args)
    {
        Vector v = new Vector( );
        Scanner sc = new Scanner(System.in);
        Calendar c;
        Student[] s=new Student[3];
        for(int i=0;i<3;i++)
        {
            c=Calendar.getInstance( );
            System.out.println("Enter the date, month and year:");
            intdd = sc.nextInt( );
            int mm = sc.nextInt( );

```

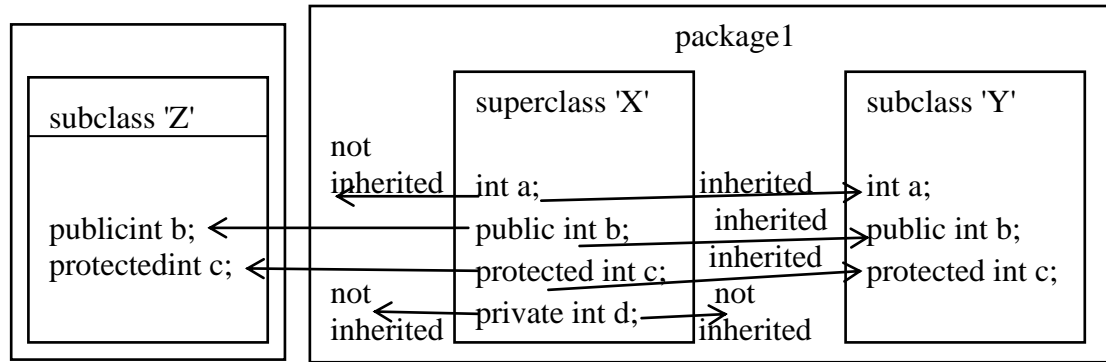
```

        intyy = sc.nextInt( );
        c.set(Calendar.DATE, dd);
        c.set(Calendar.MONTH, mm);
        c.set(Calendar.YEAR, yy);
        System.out.println("Enter the name and regno");
        String sn = sc.nextLine( );
        sc.nextLine( );
        String rn = sc.nextLine( );
        s[i] = student(sn, rn, c);
        v.add(s[i]);
    }
    Iterator it = v.iterator( );
    while(it.hasNext( ))
    {
        System.out.println(it.next( ));
    }
    student.elder(v); → invoking the static method elder( ) using the class name.
}
}

```

## **Inheritance**

- It is an important feature of Object Oriented Programming Paradigm, which helps in code reusability.
- Inheritance allows us to create a new class based on a class that has already been defined.
- A class whose properties are being acquired by some other class or a class that is inherited is called the super class or base class.
- The class that acquires the properties of some other class or the class that does the inheriting is called the subclass or derived class.
- Example: If 'Tree' is the class dealing with the general characteristics of a tree data structure, and if another class 'BinaryTree' has to be defined, and as this is also a kind of tree, this can acquire the general characteristics from the 'Tree' class and only the additional characteristics meant for a binary tree (such as number of child nodes should be  $\leq 2$ ) can be added to the 'BinaryTree' class. Thus it avoids defining 'BinaryTree' class from the scratch.  
Here, 'Tree' is the super class.  
      'BinaryTree' is the subclass.
- Thus, a subclass is one which inherits all of the members defined by the super class and adds its own, unique members.
- Now, what are the properties that can be acquired (ie., the members that can be accessed) by the subclass from the super class?
  - Assume there is a super class 'X' in package1. There is a subclass 'Y' derived from 'X' class within the same package, package1. There is another subclass 'Z' which is also derived from 'X', but is defined in a different package, package2.



| Access Mode | Subclass 'Y' | Subclass 'Z' |
|-------------|--------------|--------------|
| private     | X            | X            |
| default     | ✓            | X            |
| protected   | ✓            | ✓            |
| public      | ✓            | ✓            |

- ✓ - mark indicates that the member is inherited.  
 X - mark indicates that the member is not inherited.

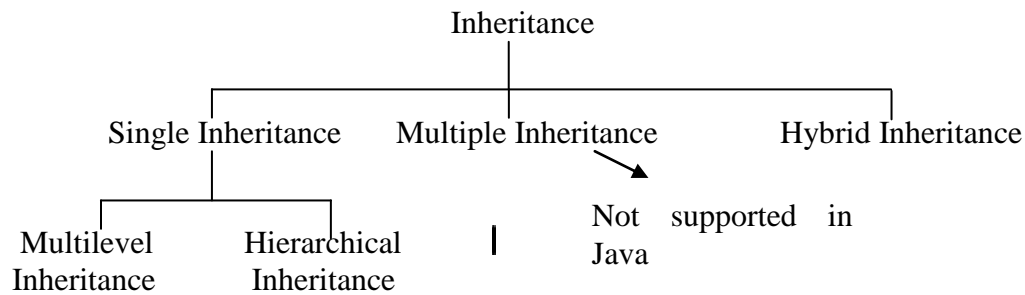
#### Points to remember:

- 1) The subclass object will always have a complete super class object within it – with all its data members and methods. If a base class member is not accessible in a derived class, then it is not an inherited member of the derived class. However, even the members that are not inherited, still form a part of a derived class object.  
 In the above example, object of class 'Z' although does not inherit the data members, 'a' & 'd', it does include these members also within it.
- 2) The base class methods that are inherited in a derived class can access all the base class members, including those that are not inherited.  
 ie., if there is a protected method, display( ) in class 'X', this method is inherited in class 'Z' also. Now this display( ) method within the class 'Z' can access the data members 'a' & 'd' also.
- 3) To derive a class from outside the package containing the base class, the base class must be declared as public. If a class is not declared as public it cannot be reached directly from outside the package.  
 ie., the class 'X' should be public in order for it to be inherited by 'Z' which is outside the package.
- 4) All the rules pertaining to inheritance are applicable to static variables. ie., a private static variable of base class is not inherited in derived class where as a protected static variable is inherited in the derived class.
- 5) Constructors in the base class are never inherited, regardless of their attributes. But they can be invoked from the sub class. ie, a child class cannot be instantiated by using the parent's parameterized constructor.



## Types of Inheritance

- There are 3 types of inheritance, generally.



### a) Single Inheritance

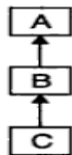
A subclass inherits only from a single class.



Here, class B inherits the features from class A.

#### i) Multilevel Inheritance

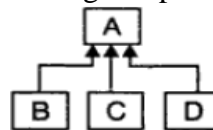
A subclass inherits from another subclass.



Here, class 'C' is inherited from class 'B' which in turn is inherited from class 'A'.

#### ii) Hierarchical Inheritance

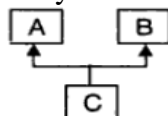
Two or more subclasses inherit from a single super class.



Classes 'B', 'C' and 'D' inherit the properties of class 'A'.

### b) Multiple Inheritance

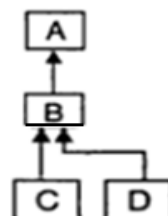
A subclass inherits more than 1 super class. But this type of inheritance is not supported in Java. Still, the functionality of multiple inheritance can be achieved in Java using interfaces.



Class 'C' is inherited from two parent structures 'A' and 'B'.

### c) Hybrid Inheritance

It is a combination of 2 or more inheritances, stated so far (single, multilevel, hierarchical)



General syntax for inheriting a class from a super class:

```
class subclassName extends superclassName
{
    ...
}
```

Example:

There is a class 'Rectangle' with instance variables 'l' and 'b'. A method perimeter( ) is defined to calculate its perimeter. Another class 'Fence' must be derived from 'Rectangle' which contains one instance variable – fencecostperunitlength and one method totalfencecost( ), to calculate the cost required to fence the whole rectangle.

```
class Rectangle
```

```
{
protected intl,b;
Rectangle()
{
    l=0; b=0;
}
Rectangle(int l,int b)
{
    this.l=l;
    this.b=b;
}
int perimeter()
{
    return(2*(l+b));
}
}
```

```
class Fence extends Rectangle
```

```
{
    int costperfeet;
    void totalfencecost()
    {
        int cost;
        cost=perimeter( ) * costperfeet;
        System.out.println(cost);
    }
}
```

/\* Members of Fence class:

- l, b, costperfeet  
- perimeter( )  
- totalfencecost( )

```
Fence(intl,intb,int c) → constructor
```

```
{
    this.l=l;
    this.b=b;
    costperfeet=c;
}
```

→ If the first statement in a derived class constructor is not a call to a base class constructor, the compiler will insert a call to the default base class constructor. So, had the default Rectangle( ) constructor not been defined, this program would show an error.

```
class fencemain
```

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

```

{
    Rectangle r=new Rectangle(2,3);
    Fence f=new Fence(3,4,10);
f.totalfencecost( );
    System.out.println(f.perimeter());
}
}

```

Output:

```

140
14

```

Since class Fence inherits class Rectangle, all members of Rectangle (except private members & constructor) are inherited in class Fence. Therefore, an object of Fence class can access l, b and perimeter( ).

In the constructor of Fence class, we are not making use of the assignment statements of the superclass constructor. The 2 same statements 'this.l=l;' and 'this.b=b;' are repeated in Fence class constructor also.

Always in a subclass constructor, if there is no explicit call to the superclass constructor, as the very first statement, the compiler will automatically insert a call to the default constructor of the super class. Thus, it is a must to provide a default constructor also, for a superclass in such situations. Observe the default constructor Rectangle( ) in Rectangle class which will be automatically called in the constructor of Fence( ) class. This is because, we have not called any constructor of Rectangle class, explicitly.

In the main class, an object of Fence is created. Now this object can access l, b, costperfeet, perimeter( ) and totalfencecost( ). When the object is created, Fence( ) constructor is invoked which assigns user specified values for l, b and costperfeet.

### How to call the super class constructor from the subclass constructor?

'super' keyword

- Whenever a subclass needs to refer to its immediate super class, it can do so by use of the keyword 'super'.
- 'super' has two forms
  - i) Calls the superclass constructor
  - ii) Used to access a member of the super class that has been hidden by a member of a subclass.

Note: 'super' cannot be used inside static methods and blocks.

- i) Calling the superclass constructor

super(arg-list);

specifies any argument needed by the constructor in the super class.

It should be the first statement executed inside a subclass constructor.

The Fence( ) constructor in the above program can be written as

```

Fence(int l,intb,int c)
{
    super(l, b);→ invokes the constructor of Rectangle(int l, int b) class
    costperfeet = c;
}

```

Fence( ) constructor now initializes only its unique variable, costperfeet. Thus Rectangle class can make its values 'private' if desired.

Since constructor can be overloaded, super( ) can be called using any form, defined by its superclass. The constructor executed will be the one that matches the arguments.

Example: In the above program, if super( ) is called, the default Rectangle( ) constructor will be invoked. If super(l, b) is called, then parameterized constructor Rectangle(int l, int b) will be invoked.

- ii) Second use for 'super':

If the subclass has a member with the same name as the member of a super class, the base class member may still be inherited, but will be hidden by the subclass. To avoid this and to refer to the inherited base class member, 'super' can be used. 'super' can be used to refer to either a method or an instance variable of the super class.

Syntax: super.member

Example: In the previous program, if the Fence class has another member 'l' which is a string representing the location of the rectangular land, then accessing 'l' within 'Fence' class, will access only the location and not the length of the super class.

```
class Fence
{
    String l; int costperfeet;
Fence(int ll,intb,int c)
{
    super(ll, b);
    costperfeet = c;
    l = "Chennai"; // 'l' represents the subclass member which
                  // is a string.
    voidtotalfencecost()
    {
        System.out.println( l );
        System.out.println(super.l); //prints 3 as 3 is the value assigned
                                     // for 'l' for main class
        System.out.println(perimeter( ) * costperfeet);
    }
}

classfencemain
{
    public static void main(String[] args)
    {
        Fence f=new Fence(3,4,10);
        f.totalfencecost( );
    }
}
```

Example 2:

Super class – Point with 2 instance variables x, y which indicate the x-y coordinates of the point. Contains default constructor which initializes a point in the origin and a parameterized constructor which initializes a user-specified point.

Sub class – Line which extends Point class. The constructor of Line class, should create a new Point object to denote the end point of the line. The starting point of the line is automatically created when it extends the Point class. Line class also includes calcDistance( ) method to find the length of the line, thus created.

```
import java.util.*;
class Point
{
    public int x,y;
    Point()
    {
        x=0;y=0;
    }
    Point(int x,int y)
    {
        this.x=x;
        this.y=y;
    }
    public String toString()
    {
        return "("+x+","+y+")";
    }
}
class Line extends point
{
    Point p;
    protected double dist;
    Line(int x,int y)           → A
    {
        super();           //invokes the default Point( ) constructor
        p=new point(x,y);
    }
    Line(int x1,int y1,int x2,int y2)   → B
    {
        super(x1,y1); //invokes the parameterized Point class constructor
        p=new Point(x2,y2);
    }
    public String toString()
    {
        return super.toString()+p.toString();
        //(or) return super.toString()+p;
    }
    double calcDistance()
    {
        double temp;
        temp=Math.sqrt(Math.pow((p.x-x),2)+Math.pow((p.y-y),2));
        System.out.println("Dist"+temp);    //uses  $\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$ 
        return temp;                        //formula to calculate the
```

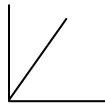
```

}                                     //distance between 2 points
}
class linemain
{
public static void main(String[] args)
{
    Line l1=new Line(3,4);           // this creates a line which starts from the origin
    Line l2=new Line(1,2,3,4);       //this creates a line which starts from a user specified
    System.out.println(l1);           point
    System.out.println(l2);
    l1.calcDistance();
    l2.calcDistance();
}
}

```

Explanation:

Line l1 = new Line(3, 4);



Two points are needed to construct a line. But in this constructor, x-y coordinates for a single point is only provided. So, we assume that the starting point of the line is from the origin (0,0) and thus invoke the Point( ) constructor, from within the Line( ) constructor. For the ending point, a new Point object is created.

Line l2 = new Line(1,2,3, 4);

The coordinates for both the starting point and ending point are provided. So, the parameterized constructor of point class is invoked.

Output:

(0,0)(3,4)

(1,2)(3,4)

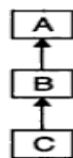
Dist 5.0

Dist 2.8284



### Multilevel Inheritance:

Here, class 'C' extends from class 'B' which in turn extends class 'A'.



The previous program can be used to demonstrate multilevel inheritance. A new class 'Triangle' can be made to inherit the Line class. Again, two types of constructors are defined in this class – one to create a triangle which starts from the origin and another to create a triangle which starts from user specified point.

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

```
class Point
```

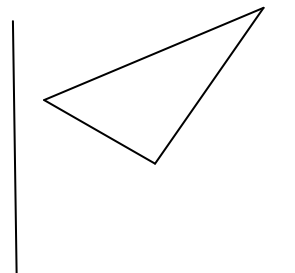
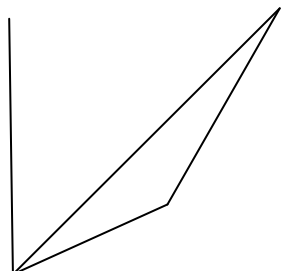
```
{
```

```
public int x,y;
```

```
Point()
```

```
{
```

```
x=0;y=0;
```



```

    }
    Point(intx,int y)
    {
        this.x=x;
        this.y=y;
    }
    public String toString()
    {
        return("("+x+", "+y+")");
    }
}
class Line extends Point
{
    Point p;
    protected double dist;
    Line(intx,int y)
    {
        super(); //invokes the default Point( ) constructor
        p=new point(x,y);
    }
    Line(int x1,int y1,int x2,int y2)
    {
        super(x1,y1); //invokes the parameterized Point class constructor
        p=new Point(x2,y2);
    }
    public String toString()
    {
        returnsuper.toString()+p;
    }
    doublecalcDistance()
    {
        double temp;
        temp=Math.sqrt(Math.pow((p.x-x),2)+Math.pow((p.y-y),2));
        System.out.println("Dist"+temp);
        return temp;
    }
}
class Triangle extends line
{
    Line l2,l3; // when Triangle extends Line class, only 1 line is created. So, we have to
    Triangle() create the other 2 lines needed to form a triangle.
    {
        super(3,4);
        l2=new Line(3,4,7,8);
        l3=new Line(7,8,0,0);
    }
    Triangle(intx,int y)

```

```

{
    super(x,y,3,4);          //for a triangle, the ending point of 1st line should be the
    l2=new Line(3,4,7,8);    starting point for the 2nd line and so forth. Hence, the
    l3=new Line(7,8,x,y);    constructor is invoked with suitable values
}
void calcArea()
{
    double a=calcDistance();
    double b=l2.calcDistance();
    double c=l3.calcDistance();
    double s=(a+b+c)/2;
    double area=Math.sqrt(s*(s-a)*(s-b)*(s-c));
    System.out.println("area="+area);
}
}
class pointmain
{
    public static void main(String[] args)
    {
        Line l1=new Line(3,4);
        Line l2=new Line(1,2,3,4);
        System.out.println(l1);
        System.out.println(l2);
        l1.calcDistance();
        l2.calcDistance();
        Triangle t1=new Triangle();
        Triangle t2=new Triangle(5,5);
        t1.calcArea();
        t2.calcArea();
    }
}

```

#### Order in which constructors are called:

In a class hierarchy, constructors are called in the order of derivation, from super class to subclass.

```

class Point
{
    Point()
    {
        System.out.println("Inside Point's Constructor");
    }
    ...
}
class Line extends Point
{
    Line()
    {
        System.out.println("Inside Line's Constructor");
    }
}

```



```
    ...
}
class Triangle extends Line
{
    Triangle( )
    {
        System.out.println("Inside Trianlge's Constructor");
    }
    ...
}
class pointmain
{
    public static void main(String[] args)
    {
        Triangle t = new Triangle( );
    }
}
```

Output:

```
Inside Point's Constructor
Inside Line's Constructor
Inside Triangle's Constructor
```

## **Polymorphism**

- 1) It is the ability to create a variable, a function or an object that has more than one form.
- 2) First of all, polymorphism works with derived class objects. A single variable of base class type can be created and made to reference objects of any derived class, and to automatically call the method that is specific to the type of the object, the variable references.
- 3) To get polymorphic operation when calling a method, the method must be a member of the base class as well as any derived classes involved. A base class variable cannot call a method of derived class, if it is not a member of base class.
- 4) Any definition of the method in a derived class must have the same signature (name & arguments) and same return type as in the base class, and must have an access specifier that is no more restrictive. This is called method overriding.

Method overriding: It is a technique which supports run time polymorphism. When a method in a subclass has the same signature and return type as that of the method in a super class, then the method in the subclass is said to override the method in the super class.

If the two methods are not identical and if only their names are similar, then the two methods are simply overloaded.

Example:

```
class Shape //base class
{
int area(int l)
{
    System.out.println("area will be
    calculated in Circle class");
}
}
Class Circle extends Shape
{
int area(int r)
{
    ...
}
}
```

Base class & sub class both have defined the `area( )` method. Moreover, their arguments and return types are same. Thus, method `area( )` is said to have been overridden by subclass `Circle`.

Had the method been defined as `int area(double r)`, then these 2 methods are said to be overloaded.

- 5) The method access specifier must be no more restrictive in the derived class than in the base class.

ie., if the `area( )` method is defined as default in the base class, in the derived class also, it must be defined with default access mode or with public access mode. But it cannot be defined with private access mode.

If in the base class, the method is defined as public, then in the subclass, it can have only public access modifier. It cannot even have default access modifier.

Example:

```
class Shape
{
    int area(int l)
    {
        ...
    }
}

Class Circle extends Shape
{
    privateint area(int r)
    {
        ...
    }
}
```

Compile Time Error

```
class Shape
{
    publicint area(int l)
    {
        ...
    }
}

Class Circle extends Shape
{
    int area(int r)
    {
        ...
    }
}
```

Compile time error because in the base class, the method has been defined with public access privilege. But in the subclass, default access mode has been specified.

- 6) Deciding which overridden method to call depends on the type of the object being stored and not on the type of the reference variable.

Because a reference variable of super class can refer to an object of any derived type, the kind of object that will be referred will not be known, until the program executes. Thus the choice of which method to execute has to be made dynamically (run time) when the program is running – it cannot be determined when the program is compiled. This is called Dynamic method dispatch or late binding.

Dynamic method dispatch:

Mechanism by which a call to an overridden method is resolved at run time, rather than at compile time. This is how Java implements run time polymorphism.

Example: In the Shape example, if the main class declares a single base class reference variable, it can be made to refer either a base class object or a derived class object.

```
class shapemain
{
    public static void main(String[] args)
    {
        Shape s;
        s = new Shape();
        s.area(5); ➔ //will invoke the area( ) method of Shape class, as
        s = new Circle();      Shape object has been stored in 's'
        s.area(5); ➔ will invoke the area( ) method of Circle class, as Circle
                                object has been stored in 's'
    }
}
```

- 7) Polymorphism applies only to methods and does not apply to data members. When you access a data member of a class object, the type of the reference variable always determines the class to which the data member belongs.

Example:

|                                                                                                                    |                                                                                                                                                  |                                                                                                                                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>class Shape {     int a;     Shape()     {         a=5;     }     int area( )     {         ...     } }</pre> | <pre>Class Circle extends Shape {     int a; int x;     Circle()     {         a=10; x=15;     }     int area( )     {         ...     } }</pre> | <pre>class shapemain {     public static void main(String[] args)     {         Shape s;         s = new Shape();         System.out.println(s.a); ➔ O/P: 5         s = new Circle();         System.out.println(s.a); ➔ O/P: 5         System.out.println(s.x); ➔ Error     } }</pre> |
|--------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

This is because the type of the reference variable only determines which data member to be accessed. 's' is a reference variable of 'Shape' type and so it will always access the data member 'a' from the Shape class and not from the derived class. Trying to access other data members that are specific to the subclass, Circle with the Shape reference variable will, in fact, flag a compile-time error itself. 'x' is a member of Circle class and not in base class. So, trying to access with the reference variable of Shape class will generate an error.

- 8) Deciding which method to call in the case of method overloading is done at compile time. For aspects that are performed at compile time, the type of reference variable matters. For aspects which are performed at run time, the type of the object being referred to, matters.

```

class Shape
{
    int a;
    Shape( ) { ... }
    void display( )
    {
        System.out.println("In Shape
        Class");
    }
    int area( ) { ... }
}

class shapemain
{
    public static void main(String[] args)
    {
        Circle c = new Circle( );
        c.display( );    ➔ O/P: In Shape Class
        c.display("Hello I am");    ➔ O/P: Hello I am In Circle Class

        Shape s;
        s = new Circle( );
        s.display( );    ➔ O/P: In Shape Class
        s.display( );    ➔ Error
    }
}

```

display( ) in Shape class & display(String msg) in Circle class are not overridden methods but they are overloaded methods. So, decision on which overloaded method to call, is made during compile time itself. In such a situation, what type of object is being stored in a reference variable does not matter. The type of the reference variable only determines the method to be called. Therefore, when the reference variable 'c' of Circle class is used, output was displayed without any problem as both the methods are available in Circle class.

But when the reference variable 's' which is of type Shape class, is used, it is not able to invoke the display(String msg) method of the subclass Circle, although it stores the reference of the Circle object.

Hence, it can be concluded that only the type of the reference variable is considered for deciding which overloaded method to call and does not depend on what object is being stored in it.

#### Example 2:

A Java program should be written to analyse if there is any benefit because of buying Message Booster and TenPaiseBooster packs, in Airtel service. Assume all calls made are of 1 minute duration. The base class AirPostPaid should read the number of calls and messages made by a customer and calculate the bill for him (charge per call = Re.1 and per message=0.90).

The subclass MessageBooster must calculate the bill assuming a booster pack of Rs.30 and no charge for any message.

The subclass TenPaiseBooster which extends MessageBooster class must calculate the bill assuming a booster pack of Rs.50 for which only 0.10 Re will be charged per call made to the same service (Airtel to Airtel). All other calls are charged as usual. Calculate the gain in both the cases. (Using MessageBooster only and using both MessageBooster and TenPaiseBooster).

```
import java.util.*;
class AirtelPostpaid
{
    int nr_calls;
    int nr_msgs;
    static final int chargepercall=1;
    static final double chargepermsg=0.90;
    AirtelPostpaid(int c,int m)
    {
        nr_calls=c;
        nr_msgs=m;
    }
    void calcBill()
    {
        System.out.print("please pay your bill amount of Rs.");
        System.out.println(nr_calls*chargepercall+nr_msgs*chargepermsg);
    }
}
class MessageBooster extends AirtelPostpaid
{
    static final int boosterpack=30;
    MessageBooster(int c,int m)
    {
        super(c,m);
    }
    void calcBill() → Overridden Method
    {
        System.out.print("please pay your bill amount of Rs.");
        System.out.println(nr_calls*chargepercall+boosterpack);
    }
    void calcGain()
    {
        double msgcharge=nr_msgs*chargepermsg;
        double diff=msgcharge-boosterpack;
        if(diff>0)
            System.out.println("you have gained Rs." +diff+" due to booster pack");
        else if(diff<0)
            System.out.println("you have lost Rs." +Math.abs(diff)+" due to booster pack");
        else
            System.out.println("No gain because of booster pack");
    }
}
```

```

    }
    class TenPaiseBooster extends MessageBooster
    {
        static final double sameservice = 0.10;
        static final int tenPaiseBooster = 50;
        int nr_sameservicecalls;
        TenPaiseBooster(int c, int m, int s)
        {
            super(c, m);
            nr_sameservicecalls = s;
        }
        void calcBill()
        {
            System.out.print("please pay your bill amount of Rs.");
            int nr_othercalls = nr_calls - nr_sameservicecalls;
            System.out.println((nr_sameservicecalls * 0.10) + nr_othercalls * chargepercall
                               + boosterpack + tenPaiseBooster);
        }
        void calcGain()
        {
            double msgcharge = nr_msgs * chargepermsg;
            double diff1 = msgcharge - boosterpack;
            double diff2 = (nr_sameservicecalls * chargepercall) - tenPaiseBooster;
            double diff = diff1 + diff2;
            if (diff > 0)
                System.out.println("you have gained Rs." + diff + " due to msg booster
                                   and ten paise booster");
            else if (diff < 0)
                System.out.println("you have lost Rs." + Math.abs(diff) + " due to due
                                   to msg booster and ten paise booster");
            else
                System.out.println("No gain or loss because of the 2 boosters");
        }
    }
    class Airtelmain
    {
        public static void main(String[] args)
        {
            System.out.println("Enter the nr.of calls and msgs");
            Scanner sc = new Scanner(System.in);
            int calls = sc.nextInt();
            int msgs = sc.nextInt();
            AirtelPostpaid a = new AirtelPostpaid(calls, msgs);
            a.calcBill();
            System.out.println("****Bill with msg booster****");
            a = new MessageBooster(calls, msgs);
            a.calcBill();
        }
    }

```

```

        System.out.println("****Bill with msg+ tenpaise booster****");
        System.out.println(" Enter the nr of Airtel to Airtel calls");
        intss=sc.nextInt();
        a=new TenPaiseBooster(calls,msgs,ss);
        a.calcBill();
        System.out.println("Gain or Loss");
        MessageBooster m=new MessageBooster(calls,msgs);
        m.calcGain();
        System.out.println("Bill with both booster");
        m=new TenPaiseBooster(calls,msgs,ss);
    m.calcGain();
    }
    }

```

Output:

```

Enter the nr. of calls and msgs
100
30
Please pay your bill amount of Rs. 127.0
****Bill with msg booster****
Please pay your bill amount of Rs. 130
****Bill with msg booster + tenpaise booster****
Enter the no of Airtel to Airtel calls
53
Please pay your bill amount of Rs. 132.3
Gain or Loss
You have lost Rs.3 due to booster pack
No gain or loss after adding both the booster packs

```

Explanation:

- (i) calcBill( ) method is found in all the 3 classes – super class and 2 subclasses. That means, they are overridden methods. So, a single reference variable, 'a' of type AirtelPostPaid can be created and can be used to store the objects of any of the 3 classes. Thus, if the object of MessageBooster is stored in it, 'a' can be used to invoke the calcBill( ) method of MessageBooster and so on.
- (ii) calcGain( ) method is found in class MessageBooster and its subclass TenPaiseBooster. So, a reference variable of type MessageBooster can either point to MessageBooster object or TenPaiseBooster object and accordingly the calcGain( ) method of one of those classes will be invoked.

## Abstract Classes

### Abstract Class:

It is a class in which one or more methods are declared, but not defined. The bodies of these methods are omitted, as implementing the methods does not make sense. Such methods are called abstract methods, as they have no definition.

### Why is an abstract class needed?

Sometimes, a super class which only defines a generalized form needs to be created and that will be shared by all of its subclasses, leaving it to each subclass to fill in the details, without providing complete implementation of every method. The super class is created merely to take advantage of polymorphism.

### Rules

- 1) In order for a subclass to override a method which has no implementation in the superclass, abstract type modifier can be specified.

Syntax:

```
abstract returntype nameofthemethod(Parameters list);
```

↘ no method body

- 2) An abstract method cannot be private since a private method cannot be inherited and therefore cannot be redefined in the subclass.
- 3) A class containing one or more abstract methods must be declared abstract.
- 4) An abstract class cannot be directly instantiated with the new operator (ie., an object cannot be created for an abstract class). If there is a subclass which extends this abstract class, then objects can be created for this.
- 5) There can be no abstract constructors or abstract static methods.(Because static methods cannot be overridden. Overriding depends on having an instance of a class. The point of polymorphism is that you can subclass a class and the objects implementing those subclasses will have different behaviors for the same methods defined in the superclass (and overridden in the subclasses). A static method is not associated with any instance of a class so the concept is not applicable.)
- 6) Any subclass of an abstract class must either implement all of the abstract methods in the superclass or be declared abstract itself.

### Example 1:

There is a super class student with fields student name, programme, year of study, hostel block names and in-time of the student. A method displayBasics( ) should display the basic details of the student. There should be 2 abstract methods getIntime( ) and getHostelBlock( ) as they have to be defined separately for boys and girls.

Now,

'BoysHostel' class should extend this class and display the HostelBlock names such as "Nethaji", "Raman", etc and the in-time as 9.30 PM.

'GirlsHostel' class should extend the superclass and display the hostel block names such as "Ida Scudder", "Mother Teresa",... and the in-time as 7.30 PM.

```
import java.util.*;
abstract class student
{
```



```
String sname;
static final String programme="B.Tech";
static final int year=2;
String[] hostelblock;
    Calendar intime;
student(String sn)
{
    sname=sn;
}
voiddisplayBasics()
{
    System.out.println(sname+" "+programme+" "+year+" ");
}
abstract void getHostelBlock();
abstract void getIntime();
}
classBoysHostel extends student
{
    BoysHostel(String sn)
    {
        super(sn);
        hostelblock=new String[]{"Raman","Isaac","Nethaji"};
        intime=Calendar.getInstance();
        intime.set(Calendar.HOUR,9);
        intime.set(Calendar.MINUTE,30);
        intime.set(Calendar.SECOND,00);
    }
    voidgetHostelBlock()
    {
        for(int i=0;i<hostelblock.length; i++)
            System.out.println(hostelblock[i]);
    }
    voidgetIntime()
    {
        System.out.println(intime.get(Calendar.HOUR)+":"+intime.get(Calendar.MINUTE));
    }
}
classGirlsHostel extends student
{
    GirlsHostel(String sn)
    {
        super(sn);
        hostelblock=new String[]{"Ida Scudder","Mother Teresa","KalpanaChawla"};
        intime=Calendar.getInstance();
        intime.set(Calendar.HOUR,7);
        intime.set(Calendar.MINUTE,30);
        intime.set(Calendar.SECOND,00);
    }
}
```

```

    }
    void getHostelBlock()
    {
        for(int i=0;i<hostelblock.length;i++)
            System.out.println(hostelblock[i]);
    }
    void getIntime()
    {
        System.out.println(intime.get(Calendar.HOUR)+":"+intime.get(Calendar.MINUTE));
    }
}
class abstractstudentmain
{
    public static void main(String[] args)
    {
        student s;
        s=new GirlsHostel("Anu");
        s.displayBasics();
        s.getHostelBlock();
        s.getIntime();
        s=new BoysHostel("Ajay");
        s.displayBasics();
        s.getHostelBlock();
        s.getIntime();
    }
}

```

If any of these 2 subclasses do not define either getHostelBlock() or getIntime() method, then these classes should also be declared abstract.

O/P:

Anu BTech 2  
 Ida Scudder  
 Mother Teresa  
 Kalpana Chawla  
 7:30  
 Ajay BTech 2  
 Raman  
 Isaac  
 Nethaji  
 9:30

#### Example 2:

There is an abstract class 'Card' which declares the method greeting( ) as abstract. Two classes Holiday and Birthday extend 'Card' class and define the greeting( ) method within them.

```

abstract class Card
{
    String name;
    int age;
    public abstract void greeting( );
}
class Holiday extends Card
{
    public Holiday(String n)
    {
        name=n;
    }
    public void greeting() // In the super class, this method has been declared public, so
    {                       here too it should be declared 'public'
        System.out.println("Dear "+name+" Season's Greeting");
    }
}

```

```
class Birthday extends Card
{
    public Birthday(String n,int a)
    {
        name=n;
        age=a;
    }
    public void greeting()
    {
        System.out.println("Dear "+name+" Happy "+age+" th Birthday");
    }
}
public class cardMain
{
    public static void main(String[] args)
    {
        Card c1 = new Holiday("Sai");
        c1.greeting();
        Card c2 = new Birthday("Anu",4);
        c2.greeting();
    }
}
```

Output:

Dear Sai Season's Greeting

Dear Anu Happy 4th Birthday

### 3 uses of word 'final':

- 1) 'final' keyword is used to create a named constant (discussed earlier).
- 2) Used to prevent method overriding.

To disallow a method from being overridden, 'final' can be used.

```
class A
{
    final void method()
    {
        System.out.println("final method");
    }
}
class B extends A
{
    void method() → as the method( ) is declared 'final' in the superclass, it
    {
        cannot be overridden in the subclass
        System.out.println("wrong");
    }
}
```

Advantages of 'final' methods:

- These methods are treated as inline functions because the compiler knows that they will not be overridden by a subclass. Inlining is an option only with 'final' methods. In

Java, programmers are not allowed to make a particular method to be compiled as inline. Only JVM decides on this.

- Normally, Java resolves calls to methods dynamically, at run time (Late Binding). Since, 'final' methods cannot be overridden, a call to one can be resolved at compile time (Early Binding).

3) Used to prevent inheritance.

'final' keyword is used to prevent a class from being inherited. If a class is declared as final, all of its methods are also declared final.

(eg)

```
final class A
```

```
{
```

```
...
```

```
}
```

```
class B extends A → Invalid
```

```
{
```

```
...
```

```
}
```

Note: A class cannot be both abstract and final, because an abstract class is incomplete by itself and relies upon its subclasses to provide complete implementation.

Trivial but interesting point:

Object of abstract class can be constructed by using anonymous inner class.

```
abstract class AbstractDemo
```

```
{
```

```
void fun()
```

```
{
```

```
System.out.println("Java");
```

```
}
```

```
abstract void show();
```

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

```
{
```

```
AbstractDemo dd = new AbstractDemo()
```

```
{
```

```
public void show()
```

```
{
```

```
System.out.println("Abstract class");
```

```
}
```

```
};
```

```
dd.fun();
```

```
dd.show();
```

```
}
```

```
}
```

This is the class which would extend the abstract class but there is an anonymous class without any name.

Output: Java

Abstract class

## Interface

- It is a collection of constants and abstract methods.
- It can either contain constants or abstract methods or both.
- It does not have instance variables and all the methods in it are defined without body.
- To make use of it, the interface should be implemented in a class, ie., declare a class by implementing the interface and write the code for each of the methods. However, each class is free to determine the details of its own implementation.

### Defining an Interface:

- Defined just like a class but by using the keyword 'interface'.
- Syntax:

```
accessmode interface Name
{
    returntype methodname1(parameter list);
    returntype methodname2(parameter list);
    ...
    type final variablename1=value;
    type final variablename2=value;
    ...
    type final variablenameN=value;
}
```

- 1) Methods are always abstract and public and so need not explicitly specify them.
  - 2) Constants are always public, static and final and so need not explicitly specify for these, either.
  - 3) Cannot have static methods or static blocks inside it.
- If the interface is declared as public, it can be used by any other code. In that case, the interface must be the only public interface declared in the file and the file must have the same name as the interface.

#### Example

```
Shape.java
public interface Shape
{
    ...
}
public class ShapeMain → Wrong, as there can be only a single public
{
    ...
}
```

class or interface within a file.

### Examples for defining an Interface:

i) interface Sample

```
{
int x=10;
```

ii) interface Conversion

```
{
double HP_TO-WATT = 745.7;
```

```

    void display(int n);
}
double WATT_TO_HP = 1/ HP_TO-WATT;
}

```

(i) Interface Sample contains a constant and one abstract method.

(ii) Interface Conversion contains only constants. The value of one constant can be defined in terms of a preceding constant. If we try to use a constant that is defined later in the interface, the code will not compile.

(eg) interface Distance

```

{
    KM_TO_M = 1000;
    M_TO_KM = 1/ KM_TO_M;
}

```

interface Distance

```

{
    M_TO_KM = 1/ KM_TO_M; ➔ Wrong
    KM_TO_M = 1000;
}

```

### Implementing the Interface:

- To implement an interface, include the 'implements' clause in the class definition, and then define the methods declared by the interface.
- Syntax:  

```

class className [extends superclass] [implements interface1 [,interface2 ...]]
{
    ...
}

```

These 2 orders should not be changed
- If the class extends another class and implements an interface, 'extends' clause should appear first and then the 'implements' clause. This is because a class can extend only one super class but can implement multiple interfaces separated by comma.
- The methods that implement an interface must be declared public as all the methods in an interface are public, by default.

### Advantages of interfaces

- Helps in multiple inheritance.
- There are a number of situations in Software Engineering when disparate groups of programmers agree to a contract that spells out how their software interacts. Each group should be able to write their code without any knowledge of how the other group's code is written. Interfaces are such contracts.

### Difference between Abstract Class and Interface

| Abstract Class                                               | Interface                                                                  |
|--------------------------------------------------------------|----------------------------------------------------------------------------|
| Never supports multiple inheritance.                         | Supports multiple inheritance.                                             |
| Method may or may not be abstract.                           | Each method is implicitly abstract.                                        |
| Programmer can define a non-abstract method and constructor. | Defines neither the constructor nor any method.                            |
| Declares any type of variable.                               | Variables are implicitly public, static & final and should be initialized. |

### When to choose interfaces over abstract classes?

- If the class which is going to inherit this parent structure needs to extend some other class also, it is better to define this as interface.

- ii) If the parent structure should contain only constants or if the implementation of its method does not make any sense, ie., if the subclasses which are going to be derived from this parent structure does not have any instance variable or method definition in common, it is better to define such a parent structure as an interface, than as an abstract class.

Note: In any case of inheritance (extending a class, abstract class or implementing an interface), the reference variable of super class or interface can access only the methods which have been overridden and not the additional methods of the subclass which are not defined in the super class.

Example:

interface shape

```
{
void area();
}
```

→ As there is no meaning in defining area( ) method for a shape, the parent structure is defined as an interface.

class square implements shape

```
{
int side;
square()
{
side=10;
}
```

public void area() → Implicitly, the area( ) method in shape interface is public & so define it as public here also.

```
{
System.out.println(side*side);
}
```

void perimeter() → The class 'square', has defined a method 'perimeter( )' which is specific to this class.

```
{
System.out.println(4*side);
}
```

class shapeMain

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

shape ss= new square( ); → cannot create an object for interface. But can create a reference variable & it can be made to refer to the classes which implement it

```
ss.area();
ss.perimeter( );
square s1 = new square( );
s1.perimeter( ); → valid
}
```

Output: 100  
40  
∴ ss.perimeter( ) is wrong.

D) Partial implementations

If a class does not fully implement the methods defined by an interface, then that class must be declared as abstract.

## II) Interfaces can be extended

An interface can extend another interface using 'extends' keyword. When a class implements an inherited interface, it must provide implementations for all the methods defined within the interface inheritance chain.

Example:

interface shape1 → super interface

```
{
void area();
}
```

→ Sub Interface

interface shape2 extends shape1

```
{
void volume();
}
```

→ Thus shape2 interface contains 2 undefined methods  
area( ) and volume( )

abstract class square implements shape2

```
{
int side;
square()
{
side=10;
}
```

As the square class does not define the 2 methods declared in the interface shape2, it should be defined as 'abstract'. But, it includes an additional method perimeter( )

```
public void area()
{
System.out.println(side*side);
}
```

```
void perimeter()
{
System.out.println(4*side);
}
```

```
class cube extends square
```

```
{
public void volume()
{
System.out.println(side*side*side);
}
```

→ Now the 'cube' class defines both the methods declared in the interface – area( ) defined in super class square and volume( ) defined in this itself. Hence, this class need not be abstract.

```
class shapeMain
```

```
{
public static void main(String[] args)
{
shape2 ss= new cube();
ss.volume();
ss.area();
}
```

```
→ Output: 1000
```

```
100
}
```



Nested interfaces

- I) An interface can be embedded within a class or another interface. ie., one interface will be a member of another interface or class.

```

class Geometry
{
    interface shape
    {
        void area();
        void perimeter();
    }
}
class square implements Geometry.shape
{
    int s;
    square(int s)
    {
        this.s=s;
    }
    public void area()
    {
        System.out.println(s*s);
    }
    public void perimeter()
    {
        System.out.println(4*s);
    }
}
class nestedinterface
{
    public static void main(String[] args)
    {
        Geometry.shapess = new square(5);
        ss.area();
        ss.perimeter();
    }
}

```

As 'shape' interface is defined within the class 'Geometry', the interface name should be mentioned along with the class name which encloses it. Thus, Geometry.shape, should be used, whenever the nested interface is used outside of its scope.

- II) Nesting classes in an interface.

An inner class to an interface will be static and public by default.

```

interface Port
{
    class Info → Objects of this would be of type Port.Info
    {
        ...
        ∴ Port.Info p = new Port.Info( )
    }
}

```

```
}
```

Example: To demonstrate achieving the effect of multiple inheritance through interface.

Write a program to encrypt the given text message.

- Interface 'Encryption' with encrypt( ) method.
- A class 'SpaceElimination' to remove the spaces in between the words of the given text message.
- Class 'Digram' should extend 'SpaceElimination' and implement 'Encryption' interface.

This class uses an encryption technique, where the given text is divided into several 2-letter substrings called digrams and each digram is reversed and concatenated.

Example: we | lc | om | e ➔ ewclmoe

- Class 'PigLatin' should extend 'SpaceElimination' and implement 'Encryption' interface.

This class uses an encryption technique, in which the first letter of the text is moved to the end of the string and the string "ay" is appended to it.

Example: Java ➔ avaJay

```
import java.util.*;
interface Encryption
{
    String encrypt();
}
class SpaceElimination
{
    String removeSpace(String s)
    {
        String result="";
        for(int i=0;i<s.length();i++)
        {
            if(s.charAt(i)==' ')
            {
                result=s.substring(0,i)+s.substring(i+1);
                s=result;
                i--; ➔ This is to repeatedly check the same location until no more spaces are
                    found in that location.
            }
        }
    }
}
//System.out.println(s);
return s;
}
}
class Digram extends SpaceElimination implements Encryption
{
    String orig;
    Digram(String s)
    {
        orig=s;
    }
    public String encrypt()
    {
```

```

        String modified="";
        int start=0,end=2;
        orig=removeSpace(orig);
        while(modified.length()!=orig.length())
        {
            String temp=orig.substring(start,end);
            //System.out.println("digram"+temp);
            String temp2=temp.substring(1)+temp.substring(0,1);
            modified+=temp2;
            start=end;
            if(end+2<=orig.length())
                end=end+2;
            else
                end=end+1;
        }
        return modified;
    }
}

class Piglatin extends SpaceElimination implements Encryption
{
    String orig;
    Piglatin(String s)
    {
        orig=s;
    }
    public String encrypt()
    {
        String modified="";
        orig=removeSpace(orig);
        String temp=orig.substring(0,1);
        modified=orig.substring(1)+temp+"ay";
        return modified;
    }
}

class encryptmain
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        String txt=sc.nextLine();
        Encryption e=new Digram(txt);
        System.out.println(e.encrypt());
        e=new Piglatin(txt);
        System.out.println(e.encrypt());
    }
}

```

I/P: hello welcome to java  
O/P: ehllwoleocemoajav  
ellowelcometo javahay

Applying Interfaces:

(Example) The stack can be implemented in many ways (fixed size or expandable or using linked list). No matter how the stack is implemented, the interface to the stack remains the same, ie., push( ) & pop( ) methods define the interface to the stack independently of the details of the implementation. Because the interface to the stack is separate from its implementation it is easy to define a stack interface, leaving it to each implementation to define the specifics.

```

interface stackDS
{
    void push(int item);
    int pop();
}
class fixedstack implements stackDS
{
    private int a[], top;
    fixedstack(int size)
    {
        a = new int[size];
        top = -1;
    }
    public void push(int item)
    {
        if (top == a.length - 1)
            System.out.println("full");
        else
            a[++top] = item;
    }
    public int pop()
    {
        if (top < 0)
        {
            System.out.println("underflow");
            return 9999;
        }
        else
            return a[top--];
    }
}
class extendstack implements stackDS
{
    private int a[], top;
    extendstack(int size)
    {
        a = new int[size];
        top = -1;
    }
    public void push(int item)
    {

```

```

if(top==a.length-1)
{
    int[] temp=new int[a.length*2];
    for(int i=0;i<a.length;i++)
        temp[i]=a[i];
    a=temp;
}
a[++top]=item;
}
public int pop()
{
    if(top<0)
    {
        System.out.println("underflow");
        return 9999;
    }
    else
        return a[top--];
}
}
}

```

```

class dynamicstack implements stackDS
{

```

```

    int data;
    dynamicstack next;
    dynamicstack head;
    dynamicstack()
    {
        head=null;
    }

```

```

    public void push(int item)
    {

```

```

        dynamicstack node=new dynamicstack();
        dynamicstack temp; → temp (reference variable)
        node.data=item;

```

```

        node.next=null;

```

```

        if(head==null)

```

```

            head=node;

```

```

        else

```

```

        {
            for(temp=head;temp.next!=null;temp=temp.next);
            temp.next=node;

```

```

        }

```

```

    }

```

```

    public int pop()
    {

```

```

        dynamicstack temp,prev=head;

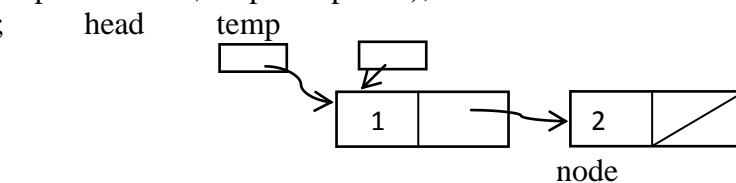
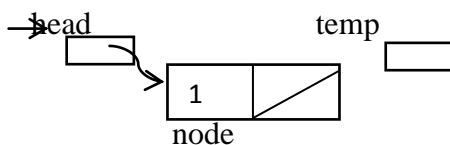
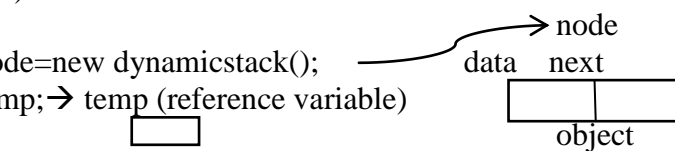
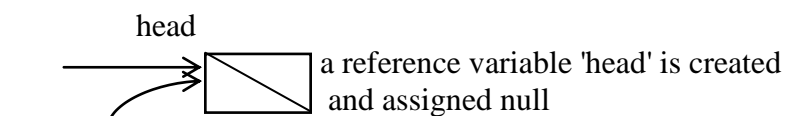
```

```

        if(head==null)

```

if head=null means no element in the stack



```

    {
        System.out.println("underflow");
        return 9999;
    }
    else
    {
        for(temp=head;temp.next!=null;temp=temp.next)
        { temp prev(Initially both temp &prev contain
            prev=temp;
        }
        int item=temp.data;
        prev.next=null;
        return item;
    }
}
}
}
classstackmain
{ prev
public static void main(String[] args)
{
    int item;
    stackDS s=new fixedstack(5);
    System.out.println(" Fixed Stack");
    for(int i=0;i<6;i++)
        s.push(i+1);
    for(int i=0;i<6;i++)
    {
        item=s.pop();
        if(item!=9999)
            System.out.println(item);
    }
    System.out.println(" Extend Stack");
    s=new extendstack(5);
    for(int i=0;i<=7;i++)
        s.push(i+1);
    System.out.println("Popped items are");
    for(int i=0;i<=7;i++)
        System.out.println(s.pop());
    System.out.println(" Dynamic Stack");
    s=new dynamicstack();
    for(int i=0;i<10;i++)
        s.push(i+1);
    for(int i=0;i<10;i++)
    {
        item=s.pop();
        if(item!=9999)
            System.out.println(item);
    }
}
}

```

Diagram illustrating the deletion of a node from a linked list:

'3' should be deleted. The link of 'prev' node is made null and data in temp node is returned.

This will be garbage collected later on

Calls to push( ) & pop( ) are resolved at runtime rather than at compile time. Accessing multiple implementations of an interface through an interface reference variable is the most powerful way Java achieves runtime polymorphism

```

    }
}
}

```

The previous program related to 'Encryption' interface also, helps to define an interface which hides the encryption technique that is being used to encrypt the given text message. The user can access the encrypt( ) method without knowing its internal implementation (ie., without bothering about whether a digram or piglatin technique is used).

## Exception Handling

### Difference between Error and Exception

| Error                                                                                                                                                                                          | Exception                                                                                           |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| Any departure from the expected behaviour of the system or program, which stops the working of the system and cannot be handled at the application level but only at the system level.         | Any error or problem which a programmer or an application can handle and continue to work normally. |
| Errors are always unchecked.<br>(Example)<br>'Out of Memory' error, 'Stack overflow', which can only be handled by the OS.                                                                     | Exceptions may be checked or unchecked exceptions.                                                  |
| Errors cannot be thrown, caught or created. Even if an 'OutOfMemory' error is caught the user will not be in a position to fix it, as it depends on OS, architecture and server configuration. | Exceptions can be created manually by the programmer, thrown or caught.                             |

### Difference between Checked and Unchecked Exceptions

| Checked Exceptions                                                                                                                                                                                                            | Unchecked Exceptions                                                                                                                                                                                                                    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Exceptions which should either be handled or registered that these may be thrown from the method under consideration, so that the caller of this method can handle these exceptions. Otherwise, the program will not compile. | The compiler ignores these exceptions and hence there will not be any compilation error, even if these exceptions are not handled or if these exceptions are not intimated to the caller of the method in which these exceptions arise. |
| Also called <u>compile time exception</u> .                                                                                                                                                                                   | Also called <u>runtime exception</u> .                                                                                                                                                                                                  |
| Example:<br>IOException, FileNotFoundException, etc.<br>These exceptions should be handled or forwarded to the caller. Else, compilation error will be generated.                                                             | Example:<br>ArrayIndexOutOfBoundsException, ArithmeticException, etc.<br>These exceptions do not stop the program from being compiled successfully even if they are not properly handled.                                               |

Exception handling in Java is managed through 5 keywords.

- try, catch, throw, throws, finally

Exception:

It is an object that is created when an abnormal situation arises in the program. It does not always indicate an error but also can signal some particularly unusual event in the program that deserves special attention.

Advantages of Exception Handling:

- i) Allows the programmer to fix the error and thus prevents the program from automatically terminating. Once the exceptions are handled, the program can resume execution from then onwards.
- ii) It separates the code that deals with errors from the code needed for the program logic. Hence, it increases readability. In languages like C, error handling code forms a part of the program logic, which reduces the program readability.

(eg) if(fp!=null)

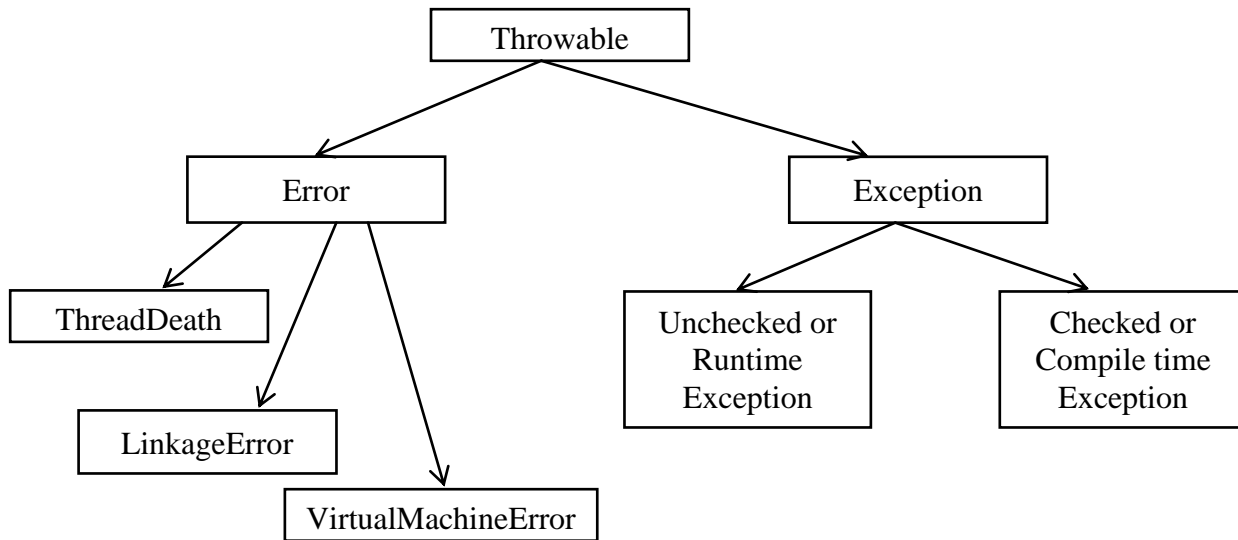
```
{  
    ...  
    if(!feof(fp))  
        ...  
    else  
        ...  
}  
else  
    ...
```

Note: Only unusual or catastrophic situations should be signalled by exceptions and not all errors, as it involves a lot of processing overhead.

- i) try – Program statements that need to be monitored for exceptions are contained within a try block. An exception occurring within the try block is thrown.
- ii) catch – Encloses the code that is needed to handle exceptions of a particular type that may be thrown in a try block.
- iii) throw – Helps to manually throw either a predefined exception or user defined exception (apart from the system generated exceptions).
- iv) throws – If a method is capable of causing an exception that it does not handle, 'throws' clause is used to specify this behaviour, so that the callers of this method can guard themselves against that exception.
- v) finally – Code in this block is always executed before the method ends, regardless of whether any exceptions are thrown in the try block or not.



Errors and Exceptions both inherit from 'Throwable' class in Java.



#### Sub classes of Runtime Exception

| Exception                                                                                                                       | Reason                                                                                                                                                                                               |
|---------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1) ArithmeticException                                                                                                          | Trying to perform division by zero.                                                                                                                                                                  |
| 2) ArrayStoreException                                                                                                          | Attempts to store wrong type of object into an array of objects.                                                                                                                                     |
| 3) IndexOutOfBoundsException<br>→ ArrayIndexOutOfBoundsException<br>→ StringIndexOutOfBoundsException                           | Trying to index an array, vector or a string, outside of its range.                                                                                                                                  |
| 4) IllegalArgumentException<br>→ NumberFormatException<br>& several other subclasses including<br>→ IllegalThreadStateException | Attempt to pass an illegal or inappropriate argument to a method.<br>→ Attempt to convert a string to one of the numeric types, but the string does not have the appropriate format.                 |
| 5) NoSuchElementException<br>→ InputMismatchException                                                                           | Useful in Enumeration class.<br>→ Thrown by a Scanner to indicate that the token retrieved does not match the pattern for the expected type or that the token is out of range for the expected type. |
| 6) NullPointerException                                                                                                         | Attempt to use null in case where an object is required.                                                                                                                                             |
| 7) ClassCastException                                                                                                           | Attempt to cast an object to a class type, which it does not belong to.                                                                                                                              |
| 8) SecurityException                                                                                                            | Thrown by the Security Manager to indicate a security violation.                                                                                                                                     |
| 9) BufferOverflowException                                                                                                      |                                                                                                                                                                                                      |
| 10) BufferUnderflowException                                                                                                    |                                                                                                                                                                                                      |
| 11) IllegalStateException                                                                                                       |                                                                                                                                                                                                      |
| 12) UnknownElementException                                                                                                     |                                                                                                                                                                                                      |
| ...                                                                                                                             |                                                                                                                                                                                                      |

Compile Time Exception:

- IOException
- ClassNotFoundException
- EOFException
- FileNotFoundException
- InterruptedException
- NoSuchFieldException
- NoSuchMethodException

Example to show what happens when an exception is caused.

```
class SampleException
{
    public static void main(String[ ] args)
    {
        int a =args.length;
        int b = 10;
        double c = b/a;      → Stmt (I)
        System.out.println(c);
    }
}
```

When this program is executed without providing any command line arguments, (like java SampleException) a's value will be zero. ∴ When the statement marked (I) is executed, b's value will be tried to be divided by zero. Division by zero is an abnormal condition as there is no Integer constant to define infinity and so this condition is raised as an exception by the Java Runtime Environment (JRE). It constructs a new ArithmeticException object and throws it. The thrown exception must be caught by an exception handler and dealt with immediately.

In the above program, as we have not supplied any exception handlers of our own, the thrown exception will be caught by the default Exception handler provided by JRE. Any exception that is not caught by the program will ultimately be processed by the default Exception handler.

Now the default handler will display a string describing the exception, print a stack trace from the point at which the exception occurred and finally terminate the program. So, if the above program is executed without providing any command line arguments, the exception generated will be printed as,

Exception in thread "main" java.lang.ArithmeticException:/by zero  
at SampleException.main(SampleException.java:7)

Indicates in which method the exception has been raised.

Indicates the program name & the line number in which this exception occurred.

When the program is modified to include a method, in which the exception is caused, the output will be as follows:

```
class SampleException
{
    static void method1( )
    {
        int a = args.length;
        int b = 10;
        double c = b/a;
```

```

        System.out.println(c);
    }
    public static void main(String[ ] args)
    {
        method1( );
    }
}
Exception in thread "main" java.lang.ArithmeticException:/by zero
    at SampleException.method1(SampleException.java:7)
    at SampleException.main(SampleException.java:12)

```

First 'method1' is displayed and then 'main' method name is displayed. This is because, when the methods are invoked while executing the program, their entries are pushed on to the stack. So, main method which was called first, is pushed on to the stack first and then the method1, which was called by main( ), is pushed on the stack.

|         |
|---------|
|         |
| method1 |
| Main    |

So, while displaying the stack trace, method1 is popped out first & displayed and then the main method.

Had any command line arguments been supplied, exception would not have been created at all and the program would be executed completely.

Although the default exception handler helps in debugging, the exceptions can be handled by us so as to prevent the program from being terminated if an exception occurs.

#### How to handle the exceptions by ourselves?

To guard against & handle a run-time error, simply enclose the code to be monitored inside a 'try' block. After the try block, include a 'catch' clause that specifies the exception type that you wish to catch.

#### General syntax of an Exception Handling Block:

```

try
{
    ...
}
catch(Exceptiontype1 e1)
{
    ...
}
catch(Exceptiontype2 e2) //optional
{
    ...
}
finally //optional
{
    //code block to be executed after the try block ends
}

```

The previous program can be modified to include the try-catch block.

Example 1:

```
class SampleException
{
    public static void main(String[ ] args)           O/P: Division by zero
    {
        int a = args.length;
        int b = 10;
        try
        {
            double c = b/a;
        }
        catch(ArithmeticException e)
        {
            System.out.println("Division by zero");
            c = 0;
        }
        System.out.println(c);
    }
}
```

"double c = b/a;" is the statement that may lead to ArithmeticException (Division by zero). So, enclose this statement inside try block so as to be monitored for any exception being thrown.

The catch block takes a single parameter, an object 'e' of type ArithmeticException. The actions to be performed, when an exception is thrown, can be mentioned within the catch block.

Program Flow:

All the statements in the main( ) method are executed one after the other, in order. Once an exception is thrown, the program control transfers out of the try block into the catch block. So, the other statements within the try block will not be executed, once the exception is thrown and the control is transferred to the catch block. [Only in a condition where there will not be any exception thrown, the remaining statements in the try block will be executed].

After executing all the statements within the catch block, the program control is given to the next line in the program following the entire try-catch mechanism.

So, in our example, when the catch block is executed, "Division by zero" is printed. Then 'c' is assigned zero.

Once the control comes out of the catch block, it continues to execute the other lines, which will print '0'. Thus, we can execute the program even beyond the point where exception is thrown.

Methods defined in 'Throwable' class to help display the description of the Exceptions in a similar way as the default exception handler would do.

(i) String getMessage( )

Returns the contents of the message describing the current exception.

(ii) a) void printStackTrace( )

Will display the exception message and the stack trace to the standard error output stream (monitor).

b) void printStackTrace(PrintStream ps)

Here, the output stream can be specified.

c) String toString( )

The Throwable class overrides the toString( ) method (of Object class) so that it returns a string containing a description of the exception. This description can be displayed in a println( ) method, by simply passing the Exception object as argument.

Modify the catch block of the previous program to include these methods.

```
class SampleException
{
    public static void main(String[ ] args)
    {
        int a = args.length;
        int b = 10;
        try
        {
            double c = b/a;
        }
        catch(ArithmeticException e)
        {
            System.out.println(e.getMessage( ));    → Statement I
            System.out.println(e);                  → Statement II
            e.printStackTrace( );                    → Statement III
            c = 0;
        }
        System.out.println(c);
    }
}
```

Output of Statement I: / by zero(ie., when e.getMessage( ) is executed)

Output of Statement II:

java.lang.ArithmeticException:/by zero  
(This is when toString( ) method is implicitly invoked by passing 'e' to System.out.println( ) method)

Output of Statement III:

java.lang.ArithmeticException:/by zero  
at SampleException.main(SampleException.java:9)  
This is when printStackTrace( ) method is invoked.

Rules for using try-catch block:

- 1) The try block must be surrounded by curly braces and cannot be used on a single statement.
- 2) Variables declared in a try block are only available within that block. The catch block itself is a separate scope from the try block.

Example: try

```
{
    int x = 10;
    ...
}
catch(Exception e)
{
```

System.out.println(x); → will give an error as 'x' is not visible to the

```

    }
    catch block for the reason that it is a local
    variable of try block

```

- 3) The parameter of catch block must be of type Throwable or one of its subclasses. If the class specified here has subclasses, the catch block will be able to process exceptions of that class + all its subclasses.

Example: `catch(Exception e)`

```

{
    ...
}

```

This block is now expected to handle all exceptions thrown by Exception class and also the exceptions thrown by its subclasses.

- 4) A 'catch' statement cannot catch an exception thrown by another try statement.

Example: try

```

{
    System.out.println(1 / 0);
}
catch(ArrayIndexOutOfBoundsException e)
{
    ...
}

```

 This catch block cannot catch the ArithmeticException thrown in the try block.

- 5) try-catch blocks are bonded together. If 'try' is embedded within a loop, catch also should be embedded inside that loop.
- 6) Must not include other code between a try block and its catch blocks or between the catch blocks and the finally block.

Example: try

```

{
    ...
}
x = y + z; → This is wrong (as we have included a statement between
catch(. . .)    try & catch blocks)
{
    ...
}

```

- 7) A method can have a single try block followed by all the catch blocks for the exceptions that need to be processed in the method.

```

try
{
    ...
}
catch(ArithmeticException e)
{
    ...
}
catch(ArrayIndexOutOfBoundsException e)

```

```

{
    ...
}

```

- 8) While defining multiple catch blocks, the most derived types should be placed first and the most basic type last. Otherwise the code will not compile.

i.e. Exception sub classes must come before any of their super classes, since a catch statement that uses a superclass will catch exceptions of that type + any of its subclasses.

Thus, a subclass would never be reached if it comes after its super class. In Java, unreachable code is an error.

Example

class supersubcatch

```

{
    public static void main(String[] args)
    {
        try
        {
            int a=0, b=42/a;
        }
        catch(Exception e)
        {
            System.out.println("I will catch all exceptions");
        }
        catch(ArithmeticException e)
        {
            System.out.println("Never reached");
        }
    }
}

```

Division by zero exception is thrown within the try block. While inspecting the catch blocks, the 1<sup>st</sup> catch block itself will be processed as it can handle exceptions of all the subclasses of the class Exception.

Under no circumstances, the catch block with ArithmeticException will be executed, because whenever an arithmetic exception is thrown, it will be caught by the 1<sup>st</sup> block itself.

Thus, an error will be displayed stating unreachable code because the exception has already been caught.

To solve this problem, reverse the order of the catch blocks. Catch block handling ArithmeticException should appear first.

Some more examples using simple try-catch blocks:

Example 2: To illustrate ArrayIndexOutOfBoundsException

```

public class ArrayIndex
{
    public static void main(String[ ] args)
    {
        try
        {
            int[] a={ 10, 20, 30};

```

```

        a[10]=5;
    }
    catch(ArrayIndexOutOfBoundsException e)
    {
        System.out.println("Give an index within the range");
        System.out.println(e);
    }
}

```

Output: Give an index within the range  
 java.lang.ArrayIndexOutOfBoundsException

### Example 3: NullPointerException

```

public class nullpointer
{
    public static void main(String[] args)
    {
        char c;String s;
        try
        {
            c = s.charAt(0);
        }
        catch(NullPointerException e) will throw an exception

        {
            System.out.println("You have not yet assigned any text to s");
            c = "A";
        }
        System.out.println(c);
    }
}

```

Output: You have not yet assigned any text to s  
 A

### Example 4: NumberFormatException

```

public class numberformat
{
    public static void main(String[] args)
    {
        try
        {
            String input = "VIT";
            int x = Integer.parseInt(input);
        }
        catch(NumberFormatException e)
        {
            System.out.println(e.getMessage());
        }
    }
}

```



```

        x = 10;
    }
    System.out.println(x+100);
}
}
Output: java.lang.NumberFormatException
110

```

### Multiple catch clauses:

A single piece of code may raise more than one exception. For this, 2 or more catch clauses can be specified. When an exception is thrown, each catch statement is inspected in order and the first one whose type matches with that of the exception is executed. After 1 catch statement is executed, others are bypassed and the execution continues after the entire try/catch block.

Example 1:

```

class multicatch
{
    public static void main(String[] args)
    {
        String s; int n;
        try
        {
            s = args[0];           → statement I
            n=Integer.parseInt(s); → statement II
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println(e.getMessage());
            s="10";
            n=Integer.parseInt(s);
        }
        catch(NumberFormatException e)
        {
            System.out.println(e);
            n = 10;
        }
        System.out.println(n*10);
    }
}

```

- i) Here, if the program is executed as 'java multicatch 10', then the output would be 100. The 2 separate catch blocks can also be written as a single catch block as follows:

**catch(ArrayIndexOutOfBoundsException | NumberFormatException e)**

- ii) If the command is java multicatch hello, then a NumberFormatException will be thrown in statement marked II. JRE then inspects all the catch blocks one by one to match this exception. The second catch block matches with this. So, it is executed and 'n' is set as 10 and finally 100 will be displayed.
- iii) If the command is java multicatch, an ArrayIndexOutOfBoundsException will be thrown by JRE, in statement I. Again while inspecting for a matching catch block, the first catch

block will be executed and the string 's' is set to "10". It is converted to an integer and stored in 'n'. The second catch block will be skipped and the last line will be executed resulting in the output of 100.

Java provides the flexibility of having as many try blocks as we want. This makes it possible to separate the various operations in a method by putting each of them in their own try block. Hence an exception thrown as a result of a problem with one operation does not prevent the subsequent operations from being executed.

The previous multi-catch program can be modified as follows:

```
class multicatch
{
    public static void main(String[] args)
    {
        String s; int n;
        try
        {
            s = args[0];
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println(e);
            s="10";
        }
        try
        {
            n=Integer.parseInt(s);
        }
        catch(NumberFormatException e)
        {
            System.out.println(e);
            n = 10;
        }
        System.out.println(n*10);
    }
}
```

This is the most preferred way of constructing the try-catch block.

Example 2:

```
class exceptionmain
{
    public static void main(String[] args)
    {
        int x, y;
        try
        {
            x = Integer.parseInt(args[0]);
        }
        catch(NumberFormatException e)
        {
        }
```

```

        {
            System.out.println(e);
            x = 10;
        }
    try
    {
        y = Integer.parseInt(args[1]);
    }
    catch(NumberFormatException e)
    {
        y = 10;
    }
    System.out.println(x + " " + y);
}
}

```

The above program can be written using a single try-catch block also, as follows:

```

try
{
    x = Integer.parseInt(args[0]);
    y = Integer.parseInt(args[1]);
}
catch(NumberFormatException e)
{
    x = 10;
    y = 10;
}
System.out.println(x + " " + y);

```

If the above program is executed as java Sample 10 Anu, no exception will be thrown while reading 'x' value. But args[1] does not represent a numerical value resulting in the throwing of exception.

Or if the above program is executed as java Sample Anu 20, args[1] is in appropriate format. But since args[0] does not represent a numerical value, exception will be fired while trying to read the value of 'x'.

In both the cases, one of the user input values is correct. Yet, both x and y are assigned default values within the catch rather than the values given by the user.

But the former program would set the default value only for the variable whose input format is wrong.

Hence the former program handles exceptions more efficiently, though it repeats the try-catch block twice for the same NumberFormatException.

Example: Write a program to find the number of valid and invalid integers entered through command line arguments.

```

    int invalid=0, valid=0, number;
    for(int i=0;i<args.length;i++)
    {
        try
        {
            number = Integer.parseInt(args[i]);

```

```

        valid++;
    }
    catch(NumberFormatException e)
    {
        invalid++; → only for an invalid input, the catch block will be executed. So
                    increment 'invalid' here
    }
}
System.out.println("Number of valid integers="+valid);
System.out.println("Number of invalid integers="+invalid);

```

Output: java sample 15 25.75 40 Anu  
 Number of valid integers=2  
 Number of invalid integers=2

### Nested try blocks:

Used in situations where a portion of a block may cause one exception and the entire block itself may cause another exception.

Example:

The code below may throw the following exceptions – `ArrayIndexOutOfBoundsException`, `NumberFormatException` & `ArithmeticException`.

As both outer and inner try blocks can cause `ArrayIndexOutOfBoundsException`, handle it in the outer catch block. As the inner block alone is prone to throw `ArithmeticException`, handle it in the inner catch block. Outer block may also fire `NumberFormatException` and so have another outer catch block for handling it.

class nestedtry

```

{
    public static void main (String[] args)
    {
        int x,y,quotient;
        try
        {
            x = 10;
            y = Integer.parseInt (args[0]);    → Statement A
            quotient;
            try
            {
                quotient = x / y;
                System.out.println(quotient);
                int[] a = {1, 2,3, 4, 5};
                for(int i=0;i<=5;i++)
                    System.out.println(a[i]); → Statement B
            }
            catch (ArithmeticException e) //for inner try block
            {
                quotient=0;
            }
        }
    }
}

```

```

catch (NumberFormatException e) //for outer try block
{
    y = 2;
}
catch (ArrayIndexOutOfBoundsException e) //outer try block
{
    System.out.println("Check the index");
}
}
}

```

Let's understand the scope of the two try-catch blocks:

Only those statements that are italicized are being monitored by the inner try block.

The statements written in bold (including the italicized ones) are being monitored by the outer try block.

In that case, any ArithmeticException raised outside of the inner try-catch block, is not monitored by the inner try block and hence will not be handled by the inner catch block.

The outer try block contains 2 matching catch blocks for NumberFormatException and for ArrayIndexOutOfBoundsException. Both the statements A & B are being monitored by the outer try block and so, ArrayIndexOutOfBoundsException thrown in both the statements A & B will be caught by the outer catch block meant for handling it.

The arithmetic exception thrown within the nested try block will be handled by the nested catch block.

Wrong way of structuring the try-catch blocks.

```

class nestedtry
{
    public static void main (String[] args)
    {
        int x,y,quotient;
        try
        {
            x = 10;
            y = Integer.parseInt (args[0]);    → Statement A
            quotient;
        }
        try
        {
            quotient = x / y;    → Statement C
            int[ ] a = { 1, 2,3, 4, 5 };
            for(int i=0;i<=5;i++)
                System.out.println(a[i]); → Statement B
        }
        catch (ArrayIndexOutOfBoundsException e) //for inner try block
        {
            System.out.println("Check the index");
        }
    }
    catch (NumberFormatException e) //for outer try block
    {

```

```

        System.out.println(e);
    }
    catch (ArithmeticException e) //outer try block
    {
        quotient = 0;
    }
}

```

Now the inner try block is for handling `ArrayIndexOutOfBoundsException`. Moreover, it can handle such an exception, only when raised from anywhere within the inner try block. So, `ArrayIndexOutOfBoundsException` raised by statement A cannot be handled by the inner catch block meant for handling such an exception as it is not within the scope of the inner try block. Thus, it will be handled by the default exception handler. [The inner try block will monitor only the statements inside it and throw an exception and the catch block will handle it].

However, the arithmetic exception thrown in statement C, can be handled by the outer catch block. This is because all the statements put inside the outer try block will be monitored for `NumberFormatException` & `ArithmeticException`. If any statement within the block raises an `ArithmeticException`, it will be caught by the outer catch block, if there is no such block provided for the inner try block.

When a try block encloses a method call & if there is another try statement within the called method, the 'try' within the method is still nested inside the outer try block, which calls the method.

class Sample

```

{
    public static void main(String[] args)
    {
        int x = 10;
        try
        {
            System.out.println(divide(x));
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println(e);
        }
    }
    static double divide(int x)
    {
        int[] y = { 1, 2, 3 };
        try
        {
            System.out.println( x / 0 ); → Statement A
        }
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }
    }
}

```

```

        return (y[3]); → Statement B
    }
}

```

divide( ) method is called from main( ). So, the control transfers to the divide( ) method. In this method, ArithmeticException thrown in Statement A is caught by the catch block provided in the method. However, the ArrayIndexOutOfBoundsException thrown in statement B is not caught within the method. But still, since the method call is enclosed within the try block of the main( ) method, it can be handled by that try-catch block.

#### throw:

- The exceptions studied so far were thrown by the Java Runtime Environment. It is also possible for the program to throw an exception explicitly, using the throw statement.
- Need for throwing the exception manually:
  - i) The calling program may need to know about the exception that had occurred in the called method. If the exception that has been caught has to be passed to the calling program, it can be re-thrown from within the catch block using a throw statement.
  - ii) Useful to throw user defined exceptions.

Syntax:

```
throw throwableInstance;
```

→ It is an object of Throwable class or its subclass

- Two ways to obtain a throwable object:
  - Using the parameter in the catch clause.
  - Creating a new one with the new operator.

Given below are the various types of constructors defined for Throwable class & its subclasses

useful for creating new exception objects.

- i) Throwable( ) → Creates an Exception that has no description.
- ii) Throwable(String msg) → Allows us to specify a description of the exception. This string is displayed when the object is used as an argument to the println( ).
- iii) Throwable(Throwable causeExc) → Constructs an Exception with the specified cause.
- iv) Throwable(String msg, Throwable causeExc) → Constructs a new Exception with the detailed msg and cause.

(iii) & (iv) forms are used for chained exceptions.

∴ To create a new NullPointerException object, the syntax is,

```
new NullPointerException( );
```

(or)

```
new NullPointerException("The string is empty");
```

To throw a NullPointerException, the syntax is,

```
throw new NullPointerException( )
```

(or)

```
NullPointerException NP = new NullPointerException( );
```

```
throw NP;
```

- The flow of execution stops immediately after the 'throw' statement. The nearest enclosing try block is inspected to see if it has a catch statement that matches the type of the exception. If there is a match, control is transferred to that statement. Else, the next enclosing try statement

is inspected and so on. If no matching catch is found, then the default exception handler halts the program & prints the stack trace.

- Example
- ```

class throwSample
{
    static void method1( )
    {
        try
        {
            throw new ArithmeticException("thrown explicilty"); → Constructor II
        }
        catch(ArithmeticException e) Although the exception has been caught it is
        {
            System.out.println("Caught inside method1");    thrown again, so that the caller of the method
            throw e;                                         comes to know about
                                                         the exception raised.
        }
    }
}

public static void main(String[] args)
{
    try
    {
        method1( );
    }
    catch(ArithmeticException e)
    {
        System.out.println("Caught inside main" + e);
    }
}

```

Output:

Caught inside method1

Caught inside main java.lang.ArithmeticException: thrown explicilty

This is the message passed to the constructor, while creating the exception object.

throws:

- If a method may cause an exception though it may not handle it, the method must specify this behaviour so that callers of the method can guard themselves against that exception.
- This is done by including a 'throws' clause in the method's declaration. A throws clause lists the type of Exceptions that a method might throw. In that case, the calling method must either provide the code for handling them or forward them to its caller.

- Syntax:

```

returntype nameofthemethod(parameters) throws exceptionlist
{
    ...
}

```

The various exceptions must be separated by comma.



- Example 1

```
class throwsexample
```

```
{
  static void method1()throws ArithmeticException
```

```
{
  System.out.println(10/0);
```

→ The exception thrown here is not handled within the method itself. Hence, it should atleast be reported to the main( ) method which calls it, using throws clause.

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

```
{
  try
```

```
{
  method1();
```

```
}
  catch(Exception e)
```

→ The main( ) method provides the appropriate try-catch block to handle this

```
{
  System.out.println("Caught");
```

```
}
```

```
}
```

If the main( ) method also is not ready to handle this exception, it can use throws clause to report to the JRE which hands it over to the default exception handler.

- 'throws' clause is helpful in the case of checked exceptions. For unchecked exceptions, even if 'throws' clause is not mentioned, it is automatically forwarded or passed to the top levels. Thus, in the previous example, even if the 'throws' clause had not been used, the raised exception would have been passed to the main method automatically and if it does not handle, it will be passed to the default exception handler.

- Example 2

```
class throwsexample
```

```
{
  static void method1( ) throws IOException
```

```
{
  int a=10, b=2, c;
```

```
  c = a / b;
```

```
  throw new IOException( );
```

→ This is a must as IOException is a checked exception. So, it will not be automatically forwarded

```
}
```

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

```
{
```

```
  try
```

```
{
    method1();
```

```
}
```

```
  catch(IOException e)
```

```
{
```

```
    System.out.println(e);
```

```
}
```

```
}
```

```
}
```

- Example 3

```
class throwsexample
```

```
{
    static void method1( ) throws RuntimeException
    {
        throw new RuntimeException( );
    }
    public static void main(String[ ] args)
    {
        try
        {
            method1( );
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
    }
}
```

This is not a must as RuntimeException is an unchecked exception and so it will be automatically passed to the caller, even if 'throws' clause is not mentioned.

- Similarly, if an object is created for Exception class, it is compulsory to mention the throws clause, if it is not handled in the method itself as Exception class includes checked exceptions also.
- If a user defined exception is created by extending Exception clause, then 'throws' is a must. Whereas if a user defined exception is created by extending RuntimeException clause, then 'throws' clause is not compulsory in order to forward the uncaught exception to the caller of the method.

#### 'finally' clause:

- Used when some particular code is to be run before a method returns, no matter whether exceptions are thrown within the try block or not.
- Example  
If a method opens a file upon entry and closes it upon exit, then the code that closes the file may be skipped by the exception handling mechanism.  
So, the code for closing a file can be enclosed within 'finally' block. It can be used to include the code for releasing other types of resources also.
- *Optional clause. But each try statement requires at least one catch or a finally block. If only the finally block is provided without a catch block, then the exception will be handed over to the Default exception handler. However, prior to doing so, the finally block will be executed.*
- The code in the finally block will be executed after a try-catch block has completed and before the code following the try-catch block.
- Will execute whether or not an exception is thrown.
- 'finally' block will be executed in all these situations.
  - i) try block executes normally without errors.

```
void method1( )
{
```

```

try
{
    System.out.println("Hello");
}
finally    → Though no exception is thrown, still finally is executed.
{
    System.out.println("in finally block");
}

```

ii) When a method gives up the program control by throwing an exception.

```

void method1( )
{
    try
    {
        throw new RuntimeException("sample");
    }
    finally
    {
        System.out.println("in finally block");
    }
}

```

→ when an exception is thrown, then also finally is executed.

iii) A try statement is exited via a return statement. Still finally clause is executed before the method returns.

```

void method1( )
{
    try
    {
        return; → before the control is returned, finally block will be executed.
    }
    finally
    {
        System.out.println("in finally block");
    }
}

```

If a value is returned within a finally block, this return overrides any return executed in the try block.

### Creating User-defined Exceptions

- To handle situations specific to our applications, our own exceptions can be created by extending the predefined Exception Classes.
- The methods defined in Throwable class (getMessage(), printStackTrace()....) will be available for user created exceptions also.

## Example 1

Read the age of the user. If it exceeds 120 or if it is negative, throw AgeException.

Step I: First create a new Exception Class with this name by extending any of the predefined Exception classes. Override the methods of Throwable class, if needed.

Step II: If any statement in the program tries to exceed the age limit, there throw an object of this newly created Exception.

Step III: Provide necessary exception handling mechanism to handle this or forward it using 'throws' clause. Remember, forwarding should be explicitly done only when the user defined exception extends checked exception. If it extends an unchecked exception, 'throws' clause is not compulsory.

import java.util.\*;

class AgeException extends Exception → Step I

```
{
    private int age;
    AgeException(int a)
    {
        age = a;
    }
    public String toString()
    {
        return "Exception:AgeException";
    }
}
```

→ Checked Exception  
If we do not define any constructor, a default constructor will be provided for this class too. Here, a constructor that takes a single argument has been defined. Hence, objects can only be created by passing an integer.

class agemain

```
{
    public static void main(String[] args) throws AgeException → Step III
}
```

→ This is compulsory as AgeException extends a checked Exception

```
{
    int age;
    Scanner sc=new Scanner(System.in);
    age=sc.nextInt();
    if(age>120 | age<0)
        throw new AgeException(age); → Step II
    System.out.println("age valid");
}
```

→ The constructor takes an integer, so pass an integer

(or) using try-catch block

```
public static void main(String[] args)
{
    int age;
    Scanner sc=new Scanner(System.in);
    age=sc.nextInt();
    try
    {
        if(age>120 | age<0)
            throw new AgeException(age);
        System.out.println("age valid");
    }
    catch(AgeException e)
```

```
        {  
            System.out.println(e);  
        }  
    }  
}
```

Example 2:

Create a base class 'father' and a derived class 'son' which extends father. If both their ages are not within 0 to 120, throw ImpossibleAgeException. If son's age is greater than father's age, throw AgeNotMatchedException.

So, 2 new user-defined exception classes should be created.

```
import java.util.*;  
class ImpossibleAgeException extends Exception  
{  
    int age;  
    ImpossibleAgeException(int a)  
    {  
        age=a;  
    }  
    public String toString()  
    {  
        return "age value"+age +"is impossible";  
    }  
}  
class AgeNotMatchedException extends Exception  
{  
    int f_age;  
    int s_age;  
    AgeNotMatchedException(int a,int b)  
    {  
        f_age=a;  
        s_age=b;  
    }  
    public String toString()  
    {  
        return "son's age ["+s_age+"] cannot be more than father's age ["+f_age+"]";  
    }  
}  
class father  
{  
    int age;  
    father(int a) throws ImpossibleAgeException  
    {  
        if(a<0 |a>120)  
            throw new ImpossibleAgeException(a);  
        age=a;  
    }  
}
```

```

}
class son extends father
{
int myage;
son(int a,int b) throws ImpossibleAgeException, AgeNotMatchedException
{
    super(a);
    super(b); → Son's age also can be checked in the super class 'father' itself.
    if (a<b)
        throw new AgeNotMatchedException(a,b);
    myage=b;
}
}
class agemain
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        try
        {
            System.out.println("Enter the age of the father");
int a=sc.nextInt();
father f=new father(a);
try
{
            System.out.println("Enter the age of the son");
int b=sc.nextInt();
son s=new son(a,b);
System.out.println(" Both the ages are agreeable");
}
catch(AgeNotMatchedException e)
{
            System.out.println(e);
}
}
catch(ImpossibleAgeException e)→ Nested try-catch block would suit this application
{
            System.out.println(e);
}
}
}

```

### Example 3:

For 'n' number of circles, read the radius. If radius is zero or negative throw InvalidRadiusException. Else, find the area of the circle.

```

import java.util.*;
class InvalidRadiusException extends Exception

```

```

{
    private double r;
    public InvalidRadiusException(double rad)
    {
        r = rad;
    }
    public String toString()
    {
        System.out.println("Radius"+r+"is not valid");
    }
}
class circle
{
    double x,y,r;
    public circle(double centrex, double centrey, double rad) throws InvalidRadiusException
    {
        if(rad<=0)
            throw new InvalidRadiusException(rad);
        else
        {
            x = centrex; y = centrey; r = rad;
        }
    }
    void area()
    {
        System.out.println(3.14*r*r);
    }
}
class circlemain
{
    public static void main(String[] args)
    {
        System.out.println("how many circles?");
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        circle[ ] c=new circle[n];
        for(int i=0;i<n;i++)
        {
            try
            {
                System.out.println("enter the x y coordinates");
                double x=sc.nextDouble();
                double y=sc.nextDouble();
                System.out.println("enter the radius");
                double r=sc.nextDouble();
                c[i] = new circle(x,y,r);
                c[i].area();
            }
            catch (Exception e)
            {
                System.out.println("Invalid Radius");
            }
        }
    }
}

```

```

    }
    catch(InvalidRadiusException e)
    {
        System.out.println(e);
    }
}
}

```

Note:

The try block is enclosed within the 'for' loop. Even if one set of input data throws an exception, the loop will be continued for the next set of input values. If the try block is placed outside the for loop (ie, enclosing the for loop), if one set of data throws an exception, the loop will be terminated once for all, as the execution will proceed from the line following the catch block.

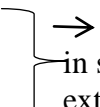
## Example 4:

Read whether the student is a fresher or senior. A fresher is allowed to register for a maximum of 27 credits. Else, throw CreditsCountException. If he is a senior, ask whether he has arrears. If so, read the total credits of arrear subjects. Then read the credits of the new courses, he wants to register. If the total credit count of new courses itself exceeds 27, throw CreditsCountException. Else if (arrear credits + total credits) exceeds 27, throw PendingArrearException. Else, display the message "Registration Successful".

```

import java.util.*;
class CreditsCountException extends Exception
{
    public String toString()
    {
        return "Total credits exceed 27";
    }
}
class PendingArrearException extends Exception
{
    int arrearcredits;
    PendingArrearException(int c)
    {
        arrearcredits=c;
    }
    public String toString()
    {
        return ("You can register new courses only for "+ (27-arrearcredits));
    }
}
class student
{
    static int creditlim=27;
    int nrcourses;
    int total,credit;
}

```

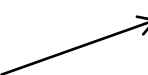

 → all these variables will be inherited  
 in senior class also which will  
 extend this.



```

student()
{
    total=0;
}
void checkCredit() throws CreditsCountException
{
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter the nr of courses u want to register");
    int nrcourses=sc.nextInt();
    for(int i=0;i<nrcourses;i++)
    {
        System.out.println("enter the nr of credits for course"+ (i+1));
        credit=sc.nextInt();
        total+=credit;
    }
    if(total>creditlim)
        throw new CreditsCountException();
}
}
class senior extends student
{
    int arrearcredits;
    senior()
    {
        arrearcredits=0;
    }
    void checkArrearsAlso() throws CreditsCountException, PendingArrearException
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("do you have any arrears? y/n");
        char havearrears=(sc.next()).charAt(0);
        if(havearrears=='y')
        {
            System.out.println("enter the total credits of the arrear subjects");
            arrearcredits=sc.nextInt();
        }
        System.out.println("enter the nr of courses u want to register");
        int nrcourses=sc.nextInt();
        for(int i=0;i<nrcourses;i++)
        {
            System.out.println("enter the nr of credits for course"+ (i+1));
            credit=sc.nextInt();
            total+=credit;
        }
        if(total>creditlim)
            throw new CreditsCountException();
        if(total+arrearcredits>creditlim)

```


 This method may throw any of the 2 exceptions. So, include both in the throws clause list

```

        throw new PendingArrearException(arrearcredits);
    }
}
class creditsmain
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Which year 1, 2, 3, 4?");
        int year=sc.nextInt();
        try
        {
            if(year==1)
            {
                student s=new student();
                s.checkCredit();
                System.out.println(" Can continue with the registration");
            }
            else
            {
                try
                {
                    senior ss=new senior();
                    ss.checkArrearsAlso();
                    System.out.println(" Can continue with the registration");
                }
                catch(PendingArrearException e)
                {
                    System.out.println(e);
                    e.printStackTrace();
                }
            }
        }
        catch(CreditsCountException e)
        {
            System.out.println(e);
            e.printStackTrace();
        }
    }
}

```

Output:	Case I	Case II
Which year 1, 2, 3, or 4?	Which year 1, 2, 3, or 4?	Which year 1, 2, 3, or 4?
2	2	2
Do you have any arrears?y/n	Do you have any arrears?y/n	Do you have any arrears?y/n
y	y	y

Enter the total credits of arrear subjects 12 Enter the nr. of courses u want to register 7 Enter the nr. of credits for course 1,2 ... 4 4 4 4 (This throws CreditsCountException) 4 4 4 Total credits exceed 27 at senior.checkArrearsAlso(creditmain.java:77)	Enter the total credits of arrear subjects 12 Enter the nr. of courses u want to register 7 Enter the nr. of credits for course 1,2 ... 3 3 3 3 3 3 3 You can register new courses only for 15 credits. at senior.checkArrearsAlso(. . .)
--	---

## Chained Exceptions

Once an application finds an exception, it may respond to it by throwing another exception. While doing so, if the information in the original exception has to be conveyed in the new exception, it can be done by chaining exceptions.

Thus, chained exceptions help to associate another exception with an exception where the 2<sup>nd</sup> exception describes the cause of the 1<sup>st</sup> exception.

Example:

If there is a NumberFormatException and the cause of which is an IOException, the caller of this method may be notified about the cause.

This is achieved by making use of chained exceptions.

Constructors & methods useful for creating chained exceptions:

- |  |                |
|--|----------------|
| i) Throwable(Throwable causeExc)                   | } Constructors |
| ii) Throwable(String msg, Throwable causeExc)      |                |
| iii) <u>Throwable</u> getCause( )                  |                |
| iv) <u>Throwable</u> initCause(Throwable causeExc) |                |
- ↙ Return types

The third and fourth form of Throwable class constructors and the 2 methods – getCause( ) and initCause( ) play a major role in chaining exceptions.

getCause( )

- Returns the exception that underlies the current exception. If no underlying exception exists, null is returned.

initCause(Throwable causeExc )

- Associates causeExc with the invoking exception and returns a reference to the exception. This can be called only once for each exception object.

If the cause Exception was set by a constructor, then it cannot be set again using initCause( ).

Example – if constructor is used to associate the cause as follows  
new RuntimeException(new ArrayIndexOutOfBoundsException( ))  
 then

e.initCause(new ArrayIndexOutOfBoundsException( )) cannot be done.

Example:

I) Setting the cause using Constructor.

```
class CauseException
{
public static void main(String[] args)
{
try
{
    ArrayIndexOutOfBoundsException e1=new ArrayIndexOutOfBoundsException();
    throw new RuntimeException("chainedException", e1);
}
catch(RuntimeException e)
{
    System.out.println(e);
    throw e;
}
}
}
java.lang.RuntimeException:chainedException }
Exception in thread "main" java.lang.RuntimeExceptionchainedException
at CauseException.main(CauseException.java:12)
Caused By: java.lang.ArrayIndexOutOfBoundsException
    at CauseException.main(CauseException.java:10)
```

After handling the exception in the catch block, it has been rethrown again. Having failed to give another try-catch block within the catch block, it is forwarded to JRE which hands it over to default exception handler. The last 4 lines of o/p are from default exception handler.

Here, the cause is set while creating the RuntimeException object (ie., in the constructor itself).

**Note: Cause cannot be set for subclasses of RuntimeException class, using constructor. Only 2 forms of constructors are there for these subclasses.**

Eg., ~~new ArithmeticException(new NumberFormatException())~~ is wrong

So, the only way of setting the cause for these classes is by invoking the initCause() method.

II) Second way of setting causes – using initCause( )

```
class CauseException
{
public static void main(String[] args)
{
    ArrayIndexOutOfBoundsException ae;
    ae=new ArrayIndexOutOfBoundsException("Chained Exception");
    ae.initCause(new RuntimeException());
    throw ae;
}
```

only 2<sup>nd</sup> form of constructor is used.  
 Later the cause is set.

```

}
}

```

Here, the constructor is not used to set the cause instead the `initCause( )` method is used to do this.

Only the 3<sup>rd</sup> and 4<sup>th</sup> form of constructors are useful in chaining exceptions.

Proof to show that 3<sup>rd</sup> and 4<sup>th</sup> forms of constructors are only used for chaining:

Using constructor type 2	Using constructor type 4
<pre> class test {     public static void main(String[] args)     {         try         {             throw new Exception("One");         }         catch(Exception e)         {             throw new Exception("Two");         }     } } </pre> <p>O/P: java.lang.Exception:Two at test.main(test.java:11)</p> <p><u>Notice that the information from the inner exception is lost when we display the stack trace of the outer exception.</u></p>	<pre> class test {     public static void main(String[] args)                         throws Exception     {         try         {             throw new Exception("One");         }         catch(Exception e)         {             throw new Exception("Two",e);         }     } } </pre> <p>O/P: java.lang.Exception:Two at test.main(test.java:11) <u>Caused by: java.lang.Exception:One</u> <u>at test.main(test.java:11)</u></p>

There is another method, which is useful in chained exception handling.

Throwable fillInStackTrace( )

- Will update the stack trace to the point at which this method is called.  
ie., if you put a call to this `fillInStackTrace( )` method in the catch block, the line number recorded in the stack record for the method in which the exception occurred will be replaced by the line number where `fillInStackTrace( )` is called.
- Use:  
If you want to rethrow an exception and record the point at which it is rethrown, this method can be used.  
Example: `e.fillInStackTrace( );`  
`throw e;`

Example:

Assume the main function calls a function `g( )` which in turn calls the `f( )` function. An exception is raised within `f( )`, which is caught by `g( )`, the caller of `f( )` method. After catching, when the `stackTrace` is printed, it will first display `f( )`, then `g( )` method and finally `main( )` method.

Then the `fillInStackTrace()` is invoked which stores the line number (in the stack) from where this is being invoked. Finally when it is rethrown to the main method, `stackTrace` contents will only be `g()` & `main()`.

```
public class Fillin
```

```
{
    public static void f() throws Exception
    {
        System.out.println("Exception originating in f( )");
        throw new Exception("thrown from f( )");
    }
    public static void g() throws Exception
    {
        try
        {
            f();
        }
        catch(Exception e)
        {
            System.out.println("Exception:" + e);
            System.out.println("Inside g( ) stackTrace contents are");
            e.printStackTrace();
            throw (Exception) e.fillInStackTrace();
        }
    }
}
```

`fillInStackTrace()` method returns a `Throwable` object. So, type casted to `Exception` object in order to match the 'throws' clause in the header of the method. Otherwise, the 'throws' clause should be `throws Throwable`

```
public static void main(String[] args) throws Exception
{
    try
    {
        g();
    }
    catch(Exception e)
    {
        System.out.println("Caught in main, the stackTrace contents are");
        e.printStackTrace();
    }
}
}
```

Output:

Exception originating in f( )  
Exception:java.lang.Exception:thrown from f( )

Inside g( ), stackTrace contents are  
java.lang.Exception:thrown from f( )

at Fillin.f(Fillin.java:6)  
at Fillin.g(Fillin.java:19)  
at Fillin.main(Fillin.java:26)

caught in main, the stackTrace contents are  
java.lang.Exception:thrown from f( )

at Fillin.g(Fillin.java:19)  
at Fillin.main(Fillin.java:26)

This is the difference in output  
after invoking fillInStackTrace( )  
method

Example 5:

Write a program to validate the given password. If the password is less than 8 characters or if the confirming password and the original password do not match, PasswordException should be thrown. If the password length is less than 8, ShortPasswordException should be set as the cause for PasswordException. If the confirming and original passwords do not match, WrongPasswordException should be set as the cause.

```
import java.util.Scanner;
```

```
class PasswordException extends Exception
```

```
{
```

```
String s=null;
```

```
PasswordException(String msg,Throwable t)
```

```
{
```

```
super(msg,t); → Invokes the super class constructor
```

```
s=msg; Use either the constructor or initCause() method to set the cause.
```

```
//initCause(t); If you do both, error will be reported.
```

```
}
```

```
public String toString()
```

```
{
```

```
String temp;
```

```
temp="java.lang. "+s+": Password Invalid";
```

```
return temp;
```

```
}
```

```
}
```

```
class ShortPasswordException extends Exception
```

```
{
```

```
public String toString()
```

```
{
```

```
return "Password too short";
```

```
}
```

```
}
```

```
class WrongPasswordException extends Exception
```

```
{
```

```
public String toString()
```

```
{
```

```

        return "Original and retyped passwords mismatch";
    }
}
class passwordmain
{
    public static void main(String[] args)
    {
        String password;
        repeat:
        do
        {
            try
            {
                System.out.println("Enter the password");
                Scanner sc=new Scanner(System.in);
                password=sc.next();
                setPassword(password);
                System.out.println("Password accepted");
                break repeat;
            }
            catch(PasswordException e)
            {
                System.out.println("*****");
                System.out.println(e);
                System.out.println("*****");
                System.out.println(e.getCause());
                e.printStackTrace();
            }
        } while(true);
    }
    static void setPassword(String p) throws PasswordException
    {
        checkLength(p);
        try
        {
            System.out.println("Retype the password to confirm");
            Scanner sc=new Scanner(System.in);
            String p2=sc.next();
            if(!(p.equals(p2)))
                throw new PasswordException("PasswordException", new
                                                WrongPasswordException());
        }
        catch(PasswordException e)
        {
            throw (PasswordException)e.fillInStackTrace();
        }
    }
}

```



```
static void checkLength(String p) throws PasswordException
{
    if(p.length()<8)
        throw new PasswordException("PasswordException",new
                                     ShortPasswordException());
}
}
```

Output:

Enter the password

Anu

java.lang.PasswordExcetion:Password Invalid

Password too short

java.lang.PasswordException:Password Invalid

at passwordmain.checkLength(passwordmain.java:84)

at passwordmain.setPassword(passwordmain.java:64)

at passwordmain.main(passwordmain.java:47)

caused by:Password too short

Enter the password

Engineering

Retype the password to confirm

Engineer

java.lang.PasswordExcetion:Password Invalid

Original and retyped passwords mismatch

java.lang.PasswordException:Password Invalid

at passwordmain.setPassword(passwordmain.java:5)

at passwordmain.main(passwordmain.java:47)

caused by:Original and retyped passwords mismatch

at passwordmain.setPassword(passwordmain.java:71)

Enter the password to confirm

Engineering

Password accepted

When the exception is because of ShortPasswordException which is thrown from the checkLength( ) method, this should also be shown in the stackTrace contents.

But when the exception is due to WrongPasswordException which is thrown from the setPassword( ) method, then there is no need to show the checkLength( ) method in the stackTrace contents. So, invoke the fillInStackTrace( ) method in the catch block of setPassword( ) method so that only setPassword( ) and main( ) methods are shown in the stackTrace contents.

Example 6:

Write a program to read values from the user to store in an integer array. If the input data is other than an integer or if the data entered already exists in the array, raise ArrayDataException. In the former case, set the cause of this exception as DataMismatchException. In the latter case, set the cause of this exception as DuplicateDataException.

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

```
class ArrayDataException extends Exception
```

```

{
    String msg;
    Throwable t;
    ArrayDataException(String s,Throwable t)
    {
        //super(s,t);
        msg=s;
        initCause(t);
        this.t=t;
    }
    public String toString()
    {
        return "ArrayDataException: caused by "+t;
    }
}
class DataMismatchException extends Exception
{
    public String toString()
    {
        return "DatatypeMismatchException";
    }
}
class DuplicateDataException extends Exception
{
    public String toString()
    {
        return "DuplicateDataException";
    }
}
class arrayexception
{
    public static void main(String[] args)
    {
        int[] a=new int[50];
        int i,j=0,x=0,index=0,count=0;
        System.out.println("Enter the nr of elements to be inserted");
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        System.out.println("Enter the elements of the Integer array");
        for(i=0;i<n;i++)
        {
            if(sc.hasNextInt())
            {
                x=sc.nextInt();
            }
            try
            {

```

```

        for(j=0;j<i;j++)
        {
            if(x==a[j])
                throw new ArrayDataException("Duplicate data",
                                                new DuplicateDataException());
        }
    }
    catch(ArrayDataException e)
    {
        System.out.println(e);
        count++;
        index=i-count+1;
        System.out.println("Give the next input data");
    }
    if(i ==j)
    {
        a[index]=x;
        index++;
    }
} //this is for outer if
else
{
    sc.next(); //read the mismatch type data also. But don't store it in the array
    try
    {
        throw new ArrayDataException("wrong data type",
                                        new DataMismatchException());
    }
    catch(ArrayDataException e)
    {
        System.out.println(e);
        count++;
        index=i-count+1;
        System.out.println("Give the next input data");
    }
}
}
}
System.out.println("the array contents are:");
for(j=0;j<n-count;j++)
    System.out.println(a[j]);
}
}

```

Count the number of duplicate data & wrong type data. Move that many indices backward so that the next data is entered in that index.

Output:

Enter the nr of elements to be inserted

5

Enter the elements of the array

1

2

e

ArrayDataException:caused by DatatypeMismatchException

Give the next input data

3

2

ArrayDataException:caused by DuplicateDataException

The array contents are:

1

2

3

### New JDK7 Exception Features:

- Automates the process of releasing a resource such as a file when it is no longer needed (try-with-resources → will be learnt in File I/O streams).
- Multicatch – allows 2 or more exceptions to be caught by the same catch clause. Each exception type in the catch clause should be separated with the OR operator.

Example:

```
try
{
    int[ ] x = { 1, 2 };
    int a = args.length;
    int b = 10;
    int c = b / a; → ArithmeticException
    x[2] = 100; → ArrayIndexOutOfBoundsException
}
catch(ArithmeticException | ArrayIndexOutOfBoundsException e)
{
    System.out.println(e);
}
```

## **Multithreading**

### Process:

- Any program under execution.
- All the programs that we write are saved in secondary memory. When we give the command to execute any particular program, it is loaded from the secondary memory into main memory and is now ready for execution.

In the current computing environment almost all the Operating Systems, support loading multiple programs into the main memory. All the programs being loaded into the main memory, will be in ready state (ie., ready for execution). The CPU scheduler (again a software) will decide which job should be given the CPU, for execution, among the several programs that have been loaded. Its decision is based on various criteria such as priority of the task, completion time, and so on. The program which is allotted the CPU time will move from ready state to running state and start execution. During its execution, it may be blocked and later on once again moved to the ready state, due to various reasons. The program which is currently being executed is called the process.

### Multiprocessing:

- The capability to execute multiple processes at the same time. This is termed as concurrency.
- In a multiprocessor environment, each process may execute one process and thus multiple processes may be executed concurrently.
- This way of running several processes at the same time (concurrently) in a multiprocessor environment is referred to as physical concurrency.  
(eg) Listening to music using VLC player & editing the MS word document, at the same time.

### Multiprogramming:

- How is concurrency achieved in a Uniprocessor Environment?
  - When a process is being executed by the single processor available, it may be blocked due to I/O operations (because I/O operations are much slower than computations) and till the process resumes its execution after finishing the I/O operation, the CPU has to be idle.  
This idle time of the CPU can be made use of, for executing another process which is in the ready state. So, even before the program finishes its execution, CPU switches to another process, to avoid the idle time. This process of CPU being switched from one process to another to utilize the idle time of the CPU efficiently is called context switching.  
Thus logical concurrency at least (though not physical concurrency) can be achieved even in a Uniprocessor system.

### Logical Concurrency:

- The frequent context switching between the processes to keep the CPU busy always, may appear as if multiple processes are executed concurrently, although they are not executed at the same time, in reality.

### Multithreading:

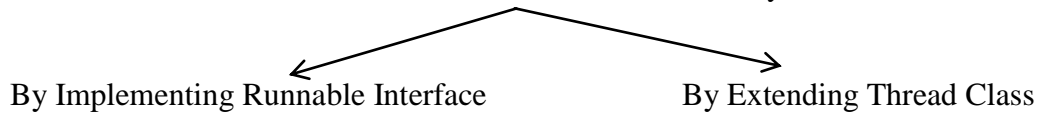
- What is a thread?  
A thread is a light weight process.  
[Light weight processes are the ones which share the same address space – same heap memory and only the stack memory will be different for light weight processes. Thus inter-process communication and context switching involves lesser overhead compared to heavy weight processes].  
Thus thread-based multitasking will be more efficient than process-based multitasking.  
In process-based multitasking, multiple processes are executed at the same time.  
In thread-based multitasking, if a single process contains 2 or more portions that can be run concurrently, then each portion is called a thread and all the threads can execute concurrently (logical or physical concurrency).  
This way of executing multiple threads of the same program concurrently is also called multithreaded programming.  
Java supports multithreaded Programming.

### How is multithreaded programming supported in Java?

- Two or more threads can be spawned (created) from within the same program and all these threads can be made to execute simultaneously.

- Creation of thread:

A thread can be created in two ways.



Runnable interface contains a single method run( ) which can be defined in the class which implements this. If the class is not going to include any additional functionality, it is better to implement the Runnable interface than to extend the Thread class.

### I) Implementing Runnable Interface

- Define a class implementing the Runnable Interface and define the run( ) method within this class.
- i) Syntax: `public void run( )`  
The class should define the run( ) method as a public member.  
run( ) method is the entry point for a new thread that is created. The thread will end when run( ) method returns. This method is not called by the JVM, when it is started, using the java command. It must be invoked by the start( ) method. Therefore, all the threads which have been created, can execute that piece of code included within the run( ) method simultaneously.
- ii) Syntax: `void start( )`  
It tells the JVM that the thread is ready for execution. Then the thread waits until it is called by the scheduler. When the thread is called, the run( ) method is invoked.
- iii) Syntax: `static currentThread( )`  
It returns a reference to the currently executing thread object which includes the name of the currently running thread, along with priority and the group it belongs to.  
If the thread is not assigned any priority, explicitly, then the default priority value 5 is assigned to it.  
There are 3 constants representing the minimum priority, default priority and the maximum priority, which can be assigned to a thread. They are:

Defined in Thread class	{	<code>public static final int NORM_PRIORITY = 5 (default priority)</code> <code>public static final int MIN_PRIORITY = 1 (minimum priority)</code> <code>public static final int MAX_PRIORITY = 10 (maximum priority)</code>
-------------------------------	---	--

Similarly, if names are not provided to the threads being created, default names are assigned for those threads (Thread-0, Thread-1, and so on).

The thread group by default is 'main' unless it is explicitly created.

- Example:  
Write a program to create 2 threads by implementing Runnable interface. The 2 threads must display the message "Hello" along with their thread name, priority and group.

class multithread implements Runnable

```
{
public void run( )
```

```
{
    Stmt A → System.out.println(Thread.currentThread( )+" says hello");
```

static method of the Thread class. So,  
use the class name

```

    }
}
class currentthread
{
    public static void main(String[] args)
    {
        Thread t1=new Thread(new multithread( ));
        Thread t2=new Thread(new multithread( ));
        // or can be done like
        // multithread m = new multithread( );
        // Thread t2 = new Thread(m);           → ∴ create an object of the class implementing
        Stmt B→System.out.println(Thread.currentThread()); Runnable interface & pass it
        t1.start();           //start( ) invoked as the parameter to the Thread class
        t2.start();           individually for each constructor to create a new Thread
        }
    }
}

```

When the Java program begins execution, the one thread that begins running immediately is called the main thread.

It is the thread from which other child threads are spawned.

- 1) In the above program, the main thread creates 2 child threads – t1 & t2.
- 2) Next it prints the description of itself, the main thread, which is returned by invoking the `currentThread()` method.
- 3) Then, it invokes the `start()` method for t1 & t2 threads which in turn will invoke the `run()` method for each of them. After invoking the `run()` method, the `start()` method immediately returns & continues to execute the other statements in the `main()` method, The 2 threads execute the code inside the `run` method concurrently.

Output:

Thread[main,5,main] → Result of executing statement A. As it is executed by main thread, its reference is returned.

Name of the thread

Thread[Thread-0, 5, main] says hello

Thread[Thread-1, 5, main]

- Name of the Thread Group which is 'main' by default.
- Default priority 5.
- Default name given to first thread
- Default name given to second thread

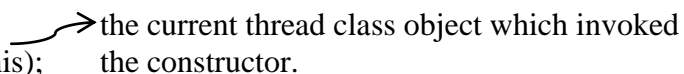
The order of execution of the 2 threads may be changed, so that Thread-1 gets executed first & Thread-2 gets executed next.

Note:

If the `run( )` method is invoked directly from the `main( )` method (as `t1.run( )`), it will be invoked on the current main thread & not on the newly created threads. So, `run( )` should only be invoked through `start( )` method.

The above program can be modified to pass, the object of the class implementing Runnable, to that class constructor where the Thread object is created & start( ) method is invoked. This avoids writing the 'start( )' statement several times, each time for 1 thread.

class threadclass implements Runnable

```
{
    Thread t; //Reference for Thread class
    threadclass( )
    {
        t = new Thread(this);    
        t.start( ); → invoked from within the constructor
    }
    public void run( )
    {
        System.out.println(Thread.currentThread( )+" says hello");
    }
}
class currentthread
{
    public static void main(String[] args)
    {
        threadclass t1=new threadclass(); → all other tasks are done within the
        threadclass t2=new threadclass();      threadclass constructor
    }
}
```

## II) By Extending Thread Class:

- Create a class by extending the Thread class. Define the run( ) method within it, to include the code which should be run in parallel.
- While creating thread object, do not create object for the Thread class (as we did while implementing Runnable Interface) but create object for the new class that has been created by extending Thread class.

- Example

class childthread extends Thread

```
{
    childthread( )
    {
        start( );
    }
    public void run( )
    {
        System.out.println(Thread.currentThread( )+" says hello");
    }
}
class childthreadmain
{
    public static void main(String[ ] args)
    {
```



```

        childthread t1=new childthread(); //create object for the user defined
        childthread t2=new childthread( );    thread class
    }
}

```

The start( ) method can also be invoked from the main( ). But an individual statement is needed for all the threads created.

Example 2:

Write a multithreaded program by creating 2 threads in which one thread will display Fibonacci numbers up to 'n' and another thread will display the factorial of 'n'.

<pre> import java.util.*; <u>class fibo implements Runnable</u> {     int n;     Thread t;     fibo(int n)     {         this.n = n;         t = new Thread(this);         t.start( );     }     public void run( )     {         int a=-1, b=1, c;         for(int i=0;i&lt;n;i++)         {             c=a+b;             System.out.println(c);             a=b;             b=c;         }     } } </pre>	<pre> <u>class fact implements Runnable</u> {     int n;     fact(int n)     {         this.n=n;     }     public void run()     {         int f=1;         for(int i=1;i&lt;=n;i++)             f=f*i;         System.out.println(f);     } } class fibofactmain {     public static void main(String[ ] args)     {         System.out.println("Enter the value of n:");         Scanner sc = new Scanner(System.in);         int n = sc.nextInt( );         fact fa=new fact(n);         fibo fi=new fibo(n); → for this, the Thread         //object is constructed within the fibo class         Thread t1=new Thread(fa);         t1.start( );     } } </pre>
--	---

Output:

Enter the value of n:

5

0

1

1

2

3

→ Fibonacci numbers

"The order of execution of the 2 threads may be changed for subsequent runs."

120 → Factorial result

Methods of Thread Class:

We can temporarily prevent a thread from execution by using any of the 3 methods of Thread class.

i) yield( )

ii) sleep( )

iii) join( )

i) yield( )

Syntax: static void yield( )

Causes the currently executing thread object to temporarily pause and allow other threads to execute.

It is simply a request for another thread to be scheduled. But the JVM, depending on the priorities of the other threads may reschedule the same yielding thread again.

ii) a) static void sleep(long milliseconds)

b) static void sleep(long milliseconds, long nanoseconds)

Causes the currently executing thread to sleep (temporarily cease execution), for the specified number of milliseconds. The second form is useful in environments that allow timing periods as short as nanoseconds.

iii) a) join( )

If any executing thread (say t1) calls join( ) on t2 ie., 't2.join( )' is invoked, then immediately t1 will enter into waiting state until t2 completes its execution.

b) join(long)

To specify the number of milliseconds up to which the calling thread must wait for the completion of the called thread.

The join( ) and sleep( ) methods throw an InterruptedException if the thread is interrupted during its sleep state or waiting state.

So, they should be used within a try-catch block or at least the Exception should be forwarded using 'Throws' clause.

```
try
{
    Thread.sleep(1000);
}
catch(InterruptedException e)
{
    System.out.println(e);
}
```

```
try
{
    t1.join( );
}
catch(InterruptedException e)
{
    System.out.println(e);
}
```

Example 3:

Create 3 threads – t1, t2 & t3. t1 should display the message "I am fast" for every 1 second, t2 should display the message "I am with moderate speed" for every 2 seconds and t3 should display the message "I am slow" for every 3 seconds.

<pre>class speedthread extends Thread {     String s;     long dur;     speedthread(String ss, long d)     {</pre>	<pre>class sleepmain {     public static void main(String[] args)     {         speedthread s1 = new speedthread("I am fast",1000);</pre>
--	---

<pre> s =ss; dur = d; start(); } public void run() {     while(true)     {         System.out.println(s); →prints 's' //for every specified duration of 'dur'         try         {             Thread.sleep(dur);         }         catch(InterruptedException e)         {             System.out.println(e);         }     } } </pre>	<pre> speedthread s2 = new speedthread("I am with moderate speed",2000);     speedthread s3 = new speedthread("I am slow",3000);  } } Output: I am fast I am with moderate speed I am slow I am fast I am with moderate speed I am slow . . . </pre>
--	--

**Example 4:**

Write a program to find the maximum element in a 1000-element array. Create 2 threads so that thread1 finds the maximum in the 1<sup>st</sup> half of the array & returns to the main( ) and thread2 finds the maximum in the 2<sup>nd</sup> half of the array & returns it to the main( ). The main( ) has to wait till both the threads complete their execution and return their individual maximum values from which the ultimate maximum can be found.

class maxarray extends Thread

```

{
    static int[ ] a=new int[1000];
    static      //array initialization if done inside the constructor, will be repeated by all the
    {          threads. So, static block.
        for(int i=0;i<1000;i++)
        {
            a[i] = i; //can get the input values form the user as well
        }
    }

    int low, high, max;
    maxarray(String name, int l, int h)
    {
        low=l;
        high=h;
        max=0;
        System.out.println(name+"has started");
        start( );
    }
}

```

```
    public void run()
    {
        max = a[low];
        for(int i=low+1;i<high;i++)
        {
            if (max<a[i])
                max = a[i];
        }
    }
    int getMax()
    {
        return max;
    }
}
class searchmain
{
    public static void main(String[] args)
    {
        maxarray m1=new maxarray("thread1",0,500);
        maxarray m2=new maxarray("thread2",500,1000);
        try
        {
            m1.join();
            m2.join();    // join( ) method also throws InterruptedException
        }
        catch(InterruptedException e)
        {
        }
        int max1=m1.getMax( ); → the maximum element from the 1st half of the array
        int max2=m2.getMax();→ the maximum element from the 2nd half of the array
        if(max1>max2)
            System.out.println(max1);
        else
            System.out.println(max2);
    }
}
```

Other important methods:

long getId()- Returns the id of the current thread

String getName() –Returns the thread's name

int getPriority()

setPriority(Thread.NORM\_PRIORITY);

Thread.State getState()- Returns the state of the thread (TERMINATED, BLOCKED, NEW, RUNNABLE)

.....

### Synchronization:

It is a process by which it is ensured that only one thread uses a shared resource at a time, although 2 or more threads need access to it. Key to synchronization is the concept of the monitor ( an object that is used as a mutually exclusive lock which can be owned by only 1 thread at a given time)

Most multithreaded systems expose monitors as objects that your program must explicitly acquire & manipulate. But in Java, there is no class monitor. But instead each object has its own implicit monitor that is automatically entered when one of the object's synchronized methods is called. Once a thread is inside a synchronized method, no other thread can call any other synchronized method on the same object. ie., synchronization support is built into the language.

The code can be synchronized in 2 ways:

- i) synchronized method
- ii) synchronized block

Synchronized method:

```
class callme
{
synchronized void call(String msg)
{
System.out.println("[ "+msg);
try
{
Thread.sleep(1000);
}
catch(InterruptedException e)
{
System.out.println("Interrupted");
}
System.out.println("]");
}
void display(String msg)
{
System.out.println("*****"+msg+"*****");
}
}
class Mythread extends Thread
{
String msg;
callme c;
Mythread(callme cc,String s)
{
c=cc;
msg=s;
start();
}
public void run()
{
c.call(msg);
c.display(msg);
}
```

```


}
class SyncMain
{
public static void main(String[] args) throws InterruptedException
{
callme cc=new callme();
    Mythread t1=new Mythread(cc,"Hello");
    Mythread t2=new Mythread(cc,"synchronized");
    Mythread t3=new Mythread(cc,"world");
        t1.join();
        t2.join();
        t3.join();
}
}

```

Synchronized void call(String msg) prevents other threads from entering call() when already some thread is using it. Once a thread enters any synchronized method on an instance, no other thread can enter any other synchronized method on the same instance. However, non synchronized methods on that instance will continue to be callable.

#### Synchronized block:

Synchronized methods will not work in all cases. Eg., If a class does not use synchronized methods (i.e., not designed for multithreaded access) but still the access to the objects of that class has to be synchronized or if the class was created by a third party, its source code cannot be accessed, and thus 'synchronized' cannot be added to the appropriate methods within the class; In these situations, simply call the methods defined by this class inside a synchronized block.

General Syntax:  Reference to the object being synchronized

```

    Synchronized(object)
    {
        // statements to be synchronized
    }

```

Previous example modified using synchronized block:

```

class callme
{
    void call(String msg)
    {
System.out.println "["+msg);
try
{
    Thread.sleep(1000);
}
catch(InterruptedException e)
{
System.out.println("Interrupted");
}
System.out.println("]");
}
}

```

Faculty: M Varalakshmi, Thilagavathi M

Note: Instead of defining the synchronized block while calling, it can also be defined within the called method as follows.

```

class callme
{
    void call(String msg)
    {
synchronized(this)
    {
System.out.println "["+msg);
try
{
    Thread.sleep(1000);
}
catch(InterruptedException e)
{
    System.out.println("Interrupted");
}
System.out.println("]");
}
}

```

```
void display(String msg)
{
System.out.println("*****"+msg+"*****");
}
}
class Mythread extends Thread
{
String msg;
callme c;
Mythread(callme cc,String s)
{
c=cc;
msg=s;
start();
}
public void run()
{
synchronized(c)
{
c.call(msg);
}
c.display(msg);
}
}
class SyncMain
{
public static void main(String[] args)
{
callme cc=new callme();
Mythread t1=new Mythread(cc,"Hello");
Mythread t2=new Mythread(cc,"synchronized");
Mythread t3=new Mythread(cc,"world");
try
{
t1.join();
t2.join();
t3.join();
}
Catch(InterruptedException e)
{
System.out.println(e);
}
}
}
```

### Inter Thread Communication

Another advantage of thread: It avoids polling. (i.e., One thread polling and thus wasting more CPU cycles waiting for the other threads to finish etc)

This is done by wait(), notify() and notifyAll() methods of Object class and so all classes have them.

- i) final void wait() throws InterruptedException  
-tells the calling thread to give up the monitor and go to sleep until some other thread enters the monitor and calls notify()
- ii) final void notify()  
-wakes up a thread that called wait() on the same object
- iii) final void notifyAll()  
-wakes up all the threads that called wait() on the same object. One of the threads will be granted access.

Note: These methods can be called only from within a synchronized context.

Oracle recommends that calls to wait() should take place within a loop that checks the condition on which the thread is waiting.

Example:

#### Producer-Consumer Problem

The Consumer should consume only if the Producer produces and Producer should continue producing only if the consumer has consumed the already produced data.

Wrong Implementation	Correct Implementation
<pre> class Q { int n; synchronized int get() { System.out.println("got:"+n); return n; } synchronized void put(int n) { this.n=n; System.out.println("put:"+n); } } class Producer implements Runnable { Q q; Producer(Q q) { this.q=q; new Thread(this,"Producer").start(); } public void run() { int i=0; for(int n=0;n&lt;10;n++) q.put(i++); </pre>	<pre> class Q { int n; boolean valueset=false; synchronized int get() { <u>while(!valueset)</u> try { <u>wait();</u> } catch(InterruptedException e) { System.out.println(e); } System.out.println("got:"+n); <u>valueset=false;</u> <u>notify();</u> return n; } synchronized void put(int n) { <u>while(valueset)</u> try { <u>wait();</u> } </pre>



<pre> } } class Consumer implements Runnable { Q q; Consumer(Q q) { this.q=q; new Thread(this,"Consumer").start(); } public void run() { int i=0; for(int n=0;n&lt;10;n++) q.get(); } } class PCmain { public static void main(String[] args) { Q q=new Q(); new Producer(q); new Consumer(q); System.out.println("Press ctrl-c to stop"); } } </pre>		<pre> catch(InterruptedException e) { System.out.println(e); } this.n=n; <u>valueset=true;</u> System.out.println("put:"+n); <u>notify();</u> } } class Producer implements Runnable { Q q; Producer(Q q) { this.q=q; new Thread(this,"Producer").start(); } public void run() { int i=0; for(int n=0;n&lt;10;n++) q.put(i++); } } class Consumer implements Runnable { Q q; Consumer(Q q) { this.q=q; new Thread(this,"Consumer").start(); } public void run() { int i=0; for(int n=0;n&lt;10;n++) q.get(); } } class PCmain { public static void main(String[] args) { Q q=new Q(); new Producer(q); new Consumer(q); System.out.println("Press ctrl-c to stop"); } } </pre>
<p>O/P of the above program:</p> <pre> put:0 got:0 got:0 got:0 got:0 Press ctrl-c to stop got:0 got:0 got:0 got:0 got:0 got:0 put:1 put:2 put:3 put:4 put:5 put:6 put:7 </pre>	<p>O/P of the correct Implementation:</p> <pre> put:0 Press ctrl-c to stop got:0 put:1 got:1 put:2 got:2 put:3 got:3 put:4 got:4 put:5 got:5 put:6 got:6 put:7 got:7 put:8 got:8 </pre>	

put:8 put:9	put:9 got:9	} }
----------------	----------------	--------

**Explanation:**

Producer should first produce the data. Thus, the Boolean variable 'valueset' is initialized to false. put() method will wait only if valueset =true. But in the beginning, as valueset=false, it will not wait but will go for producing the data. However, before starting to produce, it first sets the valueset to true. Only after producing the data, it calls the notify() method in order to wake up the consumer thread.

In the consumer side, if the valueset=false, it has to wait. Thus in the beginning, it should wait till the Producer produces some data and notifies it. Once the Producer makes valueset=true and calls notify(), consumer comes out of the wait state and starts to consume the produced data. Then it sets the valueset to false and notifies the Producer, so that Producer can resume its production work. This repeats for 10 iterations.

**Deadlock**

It occurs when two threads have a circular dependency on a pair of synchronized objects. Eg., If the thread in X tries to call any synchronized method on object Y, it will block as expected. However, if the thread in Y, in turn, tries to call any synchronized method on X, the thread waits forever, because to access X, it would have to release its own lock on Y so that the first thread would complete.

If your multithreaded program locks up occasionally, deadlock is one of the first conditions that you should check for.

Deadlocked program	Program with no deadlock
<pre> class A {     void method1(String s)     {         System.out.println(s+" is executing method1 now");     }     void method2(String s)     {         System.out.println(s+ " is executing method2 now");     } } class thread1 extends Thread {     A obj1,obj2;     thread1(A obj1,A obj2)     {         this.obj1=obj1;         this.obj2=obj2;     }     public void run() </pre>	<pre> class A {     void method1(String s)     {         System.out.println(s+" is executing method1 now");     }     void method2(String s)     {         System.out.println(s+ " is executing method2 now");     } } class thread1 extends Thread {     A obj1,obj2;     thread1(A obj1, A obj2)     {         this.obj1=obj1;         this.obj2=obj2;     }     public void run() </pre>

<pre> { synchronized(obj1) { String s=getName();     System.out.println(s+ " is trying to access method1");     obj1.method1(s); try {     Thread.sleep(2000); } catch(InterruptedException e) { System.out.println(e); }     System.out.println(s+ " is trying to access method2"); <b>synchronized(obj2)</b> {     <b>obj2.method2(s);</b> } } </pre> <p>// while holding lock on obj1, it tries to  }acquire lock on obj2 also. Same way, other  }thread too will try to get lock on obj1, while  already holding lock on obj2. This leads to  deadlock.</p> <pre> class thread2 extends Thread {     A obj1,obj2;     thread2(A obj1,A obj2)     {         this.obj1=obj1;         this.obj2=obj2;     }     public void run()     {         synchronized(obj2)         {             String s=getName();             System.out.println(s+ " is trying to access method2");             obj2.method2(s);         }         try         {             Thread.sleep(2000);         }         catch(InterruptedException e) </pre>	<pre> { synchronized(obj1) {     String s=getName();     System.out.println(s+ " is trying to access method1");     obj1.method1(s); try {     Thread.sleep(2000); } } catch(InterruptedException e) {     System.out.println(e); }     System.out.println(s+ " is trying to access method2"); <b>}→ Note the lock on obj1 by thread1 is released before getting lock on obj2.</b>     synchronized(obj2)     {         String s=getName();         obj2.method2(s);     } } } } </pre> <p>class thread2 extends Thread</p> <pre> {     A obj1,obj2;     thread2(A obj1, A obj2)     {         this.obj1=obj1;         this.obj2=obj2;     }     public void run()     {         synchronized(obj2)         {             String s=getName();             System.out.println(s+ " is trying to access method2");             obj2.method2(s);         }         try         {             Thread.sleep(2000);         }         catch(InterruptedException e) </pre>
--	--

<pre> {     System.out.println(e); } System.out.println(s+" is trying to access method1");     synchronized(obj1) {     obj1.method1(s); } } } } class deadlockdemo { public static void main(String[] args) {     A obj1=new A();     A obj2=new A();     thread1 t1=new thread1(obj1,obj2);     thread2 t2=new thread2(obj1,obj2);     t1.start();     t2.start(); } } </pre> <p>Output:</p> <p>Thread-0 is trying to access method1  Thread-1 is trying to access method2  Thread-1 is executing method2 now  Thread-0 is executing method1 now  <u>Thread-0 is trying to access method2</u>  <u>Thread-1 is trying to access method1</u>  The two threads will be blocked for ever  without getting terminated</p>	<pre> {     System.out.println(e); } System.out.println(s+" is trying to access method1"); } → <b>Note the lock on obj2 by thread2 is released before getting lock on obj1.</b> synchronized(obj1) {     String s=getName();     obj1.method1(s); } } } } class nodeadlock { public static void main(String[] args) {     A obj1=new A();     A obj2=new A();     thread1 t1=new thread1(obj1,obj2);     thread2 t2=new thread2(obj2,obj1);     t1.start();     t2.start(); } } </pre> <p>Output:</p> <p>Thread-0 is trying to access method1  Thread-1 is trying to access method2  Thread-1 is executing method2 now  Thread-0 is executing method1 now  Thread-1 is trying to access method1  Thread-0 is trying to access method2  Thread-0 is executing method2 now  Thread-1 is executing method1 now  Then the 2 threads terminate their execution.</p>
--	--

The same program can be written by defining a single thread class instead of two but thread1 should pass ob1 and obj2 whereas thread2 should pass ob2 & obj1.

<pre> class A { void method1(String s) {     System.out.println(s+" is executing method1 now"); } void method2(String s) { </pre>	<pre> class deadlockdemo2 {     public static void main(String[] args)     {         A obj1=new A();         A obj2=new A();         thread1 t1=new thread1(<b>obj1,obj2</b>);         thread1 t2=new thread1(<b>obj2,obj1</b>);         t1.start(); </pre>
---	---

<pre> System.out.println(s+ " is executing method2 now"); } } class thread1 extends Thread {     A obj1,obj2;     thread1(A obj1, A obj2)     {     this.obj1=obj1;     this.obj2=obj2;     }     public void run()     {     String s=getName();     synchronized(obj1)     {         System.out.println(s+ " is trying to access method1");         obj1.method1(s);         try         {             Thread.sleep(2000);         }         catch(InterruptedException e)         {             System.out.println(e);         }         System.out.println(s+ " is trying to access method2");          synchronized(obj2)         {             obj2.method2(s);         }     } } </pre>	<pre> t2.start(); } </pre> <p>With this, thread1 will acquire lock on obj1 first and without releasing this lock, try to get lock on obj2. Thread2 will first acquire lock on obj2 first and then try to get lock on obj1.</p>
--	--

Example:

Create 2 threads-one for generating 30 fibonacci numbers and another for generating all the prime numbers upto 30. The first thread should print first 20 fibonacci numbers and pause. It should resume its execution only after the second thread displays all its prime numbers.

<pre> class fiboprime { boolean fiboflag=true; synchronized void generatefibo(int n) { </pre>	<pre> class fibothread implements Runnable { int n; Thread t; fiboprime fp; </pre>
---	--

<pre> int a=-1,b=1,c=0; for(int i=0;i&lt;n;i++) { c=a+b; System.out.println("fibo"+c); if(i ==19) { fiboflag=false; notify(); while(!fiboflag) try { wait(); } catch(InterruptedException e) { System.out.println(e); } fiboflag=true; } a=b; b=c; } } synchronized void generateprime(int n) { while(fiboflag) try { wait(); } catch(InterruptedException e) { System.out.println(e); } fiboflag=false; int count=0; for(int i=2;i&lt;=n;i++) { for(int j=2;j&lt;i;j++) { if(i%j==0) count++; } } if(count==0) System.out.println("prime"+i); count=0; </pre>	<pre> fibothread(int n,fiboprime fp) { this.n=n; this.fp=fp; t=new Thread(this); t.start(); } public void run() { fp.generatefibo(n); } } class primethread implements Runnable { int n; fiboprime fp; Thread t; primethread(int n,fiboprime fp) { this.n=n; this.fp=fp; t=new Thread(this); t.start(); } public void run() { fp.generateprime(n); } } class fiboprime_wait { public static void main(String[] args) { fiboprime fp=new fiboprime(); primethread p1=new primethread(30,fp); fibothread f1=new fibothread(30,fp); } } </pre>
--	---

<pre> }     fiboflag=true; notify(); } } </pre>	
---	--

## Packages

### Package:

- It is a named collection of classes. It is both a naming control and visibility control mechanism. It is a way to properly manage the namespace.
- All the programs which we have done so far, have been stored in the default package which does not have any name.
- If needed, a new package can be created and all the files (classes) which we wish to store within that package can be defined inside it. So, those classes that are defined within that package become inaccessible by the code outside the package, unless that package is imported (within the code which wants to access it).
- Example: If there is a package named 'NumberChecking' in which we wish to store classes like ArmstrongNumber, Palindrome, PerfectNumber, etc., then these 3 classes become unavailable to the classes outside the package 'NumberChecking'.

### How to define a package?

- 'package' statement must be included and this must be the first statement in a Java program. Otherwise the class names will be saved inside the default package.
- Syntax:  

```
package packageName;
```
- Note: If there are any import statements within the program, they should also appear only after the package statement.
- Example  

```
package numberchecking;
```

If we wish to add the above mentioned classes to this package, it can be done as follows:

<pre> Armstrong.java package numberchecking; class Armstrong {     //code for checking     whether a number is     Armstrong or not } </pre>	<pre> Palindrome.java package numberchecking; class Palindrome {     void checkPalindrome(n)     {         //code     } } </pre>	<pre> Evennr.java package numberchecking; class Evennr {     void checkEven(int n)     {         //code     } } </pre>
--	--	--

### Steps:

- 1) Create a directory with the name 'numberchecking'.

- 2) Inside this directory, save these 3 programs as Armstrong.java, Palindrome.java and Evennr.java
- 3) If any program (which is not defined within this package) wants to use these classes, then import this package into the program.

Example: Assume we want to use these 3 classes in a program called WorkwithNumbers.java. Then the program must have imported the 'numberchecking' package.

WorkwithNumbers.java

```
import numberchecking.*; //this imports all the 3 classes inside the package
```

```
class WorkwithNumbers
```

```
{
    public static void main(String[ ] args)
    {
        int n = 10;
        Evennr e = new Evennr( );
        e.checkEven(n); → As the 'Evennr' class has been imported, checkEven( )
                        method can be used
    }
}
```

Important Note: In order for a class to be imported from outside the package in which it is defined, the class must be declared public. Even the methods which have to be accessed across the package must be declared public. ∴ In the 'Evennr.java' file, the class should be public and also the checkEven(int n) method should be public as well.

Evennr.java

```
public class Evennr
```

```
{
    public void checkEven(int n)
    {
        if(n%2 == 0)
            System.out.println("Even");
        else
            System.out.println("Odd");
    }
}
```

- 4) How to compile the program?

Assume "numberchecking" package is in the following directory C:\java\Programs

Then the 3 files are saved as

C:\java\Programs\numberchecking\Armstrong.java

C:\java\Programs\numberchecking\Palindrome.java

C:\java\Programs\numberchecking\Evennr.java

To Compile:

Move to the 'numberchecking' folder and compile each one of them.

C:\java\Programs\numberchecking:>javac Armstrong.java

C:\java\Programs\numberchecking:>javac Palindrome.java

C:\java\Programs\numberchecking:>javac Evennr.java

Now the 3 class files would have been generated and saved in the same folder 'numberchecking'.



Next to compile WorkwithNumbers.java, let us assume it is stored in C:\java\samples folder. Therefore go to that folder and compile.

C:\java\samples:>javac -cp "C:\java\Programs" WorkwithNumbers.java

This is because we try to use class files of 'Evennr' class in the program. ∴ the path where its class files can be found, should be specified. -cp is an option to override the class path set in the environmental variable

To Run:

java -cp.; "C:\java\Programs" WorkwithNumbers

→ this dot includes the current directory also. This is because to execute the program, the class files of Evennr as well as "WorkwithNumbers" are needed. The class file of "WorkwithNumbers" has been created inside 'C:\java\Programs' folder. Hence this folder is also included using '.'

Note: This way of execution is needed only when the program which uses the classes of the package & the package itself are in 2 different directories.

Example 1:

Create two classes CGPA and CGPMain as a single file in a package 'package1'.

package package1;

class CGPA

{

String name;

double cgpa;

CGPA(String name,double cgpa)

{

    this.name=name;

    this.cgpa=cgpa;

}

void display()

{

    System.out.println(name+" has a CGPA of "+cgpa);

}

}

class CGPMain

{

public static void main(String[] args)

{

    CGPA s1=new CGPA("Anu",9.1);

    CGPA s2=new CGPA("Ajay",9.2);

    s1.display();

    s2.display();

}

}

Inside the current working directory, create a subdirectory named 'package1'. Move to this folder and save this file as CGPMain.java.

∴ If the current directory is C:\Java, then the program is saved as C:\Java\package1\CGPMain.java

Compile: Within the 'package1' folder give the 'javac' command as follows

```
C:\Java\package1:>javac CGPAMain.java
```

Execute: C:\Java\package1:> java -cp "C:\Java" package1.CGPAMain→This is because CGPAMain class has also defined inside package1. The path for accessing the package1 should also be given while executing. Hence, "C:\Java"

O/P: Anu 9.1

Ajay 9.2

(or)

Go up one folder backward and give the following command.

```
C:\Java:>java package1.CGPAMain
```

This is because the main class also has been defined inside 'package1' directory.

Example 2:

Assume there are two branches of a restaurant and the accommodation charge is different in both the branches. Include all the classes - branch1, branch2 and the main class within the same package 'restaurant'. Calculate the bill for a customer in both the branches.

```
package restaurant;
public class branch1
{
    String loc;
    int rent;
    public branch1(String l, int r)
    {
        loc = l;
        rent = r;
    }
    public int calculateRent(int n)
    {
        System.out.println(rent * n);
        return (rent * n);
    }
}
```

→ save this inside the folder 'restaurant' with the file name branch1.java  
Therefore, your file is as shown  
C:\java\restaurant\branch1.java

```
package restaurant;
public class branch2
{
    String loc;
    int rent;
    public branch2(String l, int r)
    {
        loc = l;
        rent = r;
    }
    public int calculateRent(int n)
    {
        System.out.println(rent * n);
    }
}
```

→ save this file as restaurantmain.java inside the same folder 'restaurant'

```

    }
}
class restaurantmain
{
    public static void main(String[ ] args)
    {
        branch1 b1 = new branch1("TNagar",400);
        branch2 b2 = new branch2("AshokNagar",400);
        b1.calculateRent(4); //for 4 days
        b2.calculateRent(5); //for 5 days
    }
}

```

**Compile:**

Compile both the source files

C:\java\restaurant:>javac branch1.java

C:\java\restaurant:>javac -cp "C:\Java" restaurantmain.java

**Note:**

The path to the “restaurant” package needs to be given only while compiling the program which uses that package classes. ∴ Class path is not needed to be given while compiling branch1.java, as it does not use classes of any other package. So, simply go to that folder and compile.

**Execute:**

Go back one folder and execute the command as follows:

C:\java:>java restaurant.restaurantmain

**Or**

C:\Java\restaurant:> java -cp “C:\Java” restaurant.restaurantmain

A hierarchy of packages can also be created by simply separating each package name from the one above it by use of a period.

**Syntax:**

package pkg1[.pkg2[.pkg3]];

square bracket means, the contents inside it are not mandatory. But pkg1 should be compulsorily given (so, not enclosed within square bracket).

For the previous program, within the 'restaurant' folder, create another folder 'AC' and include an extra class ACRoom which includes AC charge also. This needs to access calculateRent( ) method from branch1 & branch2 classes of 'restaurant' package.

package restaurant.AC;

import restaurant.\*;

class ACRoom extends branch1

```

{
    int ACcharge;
    ACRoom(String l, int r, int a)
    {
        super(l, r);
        ACcharge = a;
    }
}

```

→ create a folder 'AC' within the 'restaurant' folder & save this file as 'ACmain.java'.  
your file will look like  
C:\java\restaurant\AC\ACmain.java

```
void roomRent( )
{
    int n = 5;
    System.out.println(calculateRent(n) * ACcharge * n);
}
}
class ACmain
{
    public static void main(String[] args)
    {
        ACRoom ac = new ACRoom("AshokNagar", 300, 50);
        ac.roomRent( );
    }
}
```

Compile:

Go to 'AC' folder & give the command

```
C:\java\restaurant\AC:>javac -cp "C:\java" ACmain.java
```

Execute:

```
C:\java\restaurant\AC:>java -cp "C:\java" restaurant.AC.ACmain
```

Example 2:

There is a package 'sphere' in which a class 'GeneralSphere' is defined. This includes a method for calculating the volume of a sphere. Create another sub package called 'planet' for this, which contains the class 'GeneralPlanet'. In this class define a method to find the time taken for light to travel from sun to the planet (Speed of light = 186000 miles).

i)

package sphere;

public class GeneralSphere

```
{
    double r;
    double volume;
    public GeneralSphere(double rr)
    {
        r = rr;
        volume = 0;
    }
    public void calcVolume( )
    {
        volume = (4.0 / 3) * 3.14 * r * r * r;
        System.out.println(volume);
    }
}
```

save this file inside the folder 'sphere'

∴ filename will be

C:\java\sphere\GeneralSphere.java

Compile: Go to this folder & compile

```
C:\java\sphere:>javac GeneralSphere.java
```

ii)

package sphere.planet;

```
import sphere.*;
public class GeneralPlanet extends GeneralSphere
{
    long dist;
    static long lightspeed = 186000; //miles per second
    public GeneralPlanet(double r, long d)
    {
        super(r);
        dist = d;
    }
    public void timeTaken( )
    {
        System.out.println(dist / lightspeed);
    }
}
```

Save this file in a folder 'planet'

Filename: C:\java\sphere\planet\GeneralPlanet.java

Compile: Go to this folder and compile

C:\java\sphere\planet:>javac GeneralPlanet.java

iii) Assume there are 2 files which have to use the methods in these packages – ballmain.java & planetmain.java. they are outside of these packages (ie., in "C:\Programs" folder)

ballmain.java

```
import sphere.*;
```

```
class ballmain
```

```
{
    public static void main(String[ ] args)
    {
        GeneralSphere volleyball = new GeneralSphere(20);
        GeneralSphere basketball = new GeneralSphere(24);
        volleyball.calcVolume( );
        basketball.calcVolume( );
    }
}
```

↑ approx. radius in cms

to indicate that the class path for the classes imported from 'sphere' package is this.

Compile: C:\Programs:>javac -cp "C:\java" spheremain.java

→ Go to 'Programs' folder where spheremain.java is saved

iv)

```
import sphere.*;
```

```
import sphere.planet.*;
```

```
class planetmain
```

```
{
    public static void main(String[ ] args)
    {
        GeneralPlanet earth = new GeneralPlanet((double)6371000, 9300000000);
        earth.timeTaken( );
        earth.calcVolume( );
    }
}
```

↑ (in cms)

↓ (distance from sun in miles)

$$\left. \begin{array}{l} \{ \\ \} \end{array} \right\}$$

Compile: Go to 'Programs' folder where the planetmain.java resides.

```
C:\Programs:>javac -cp "C:\java" planetmain.ajva
```

Execute:

C:\Programs:> java -cp "C:\java";. ballmain } don't give any package name before the filename because

```
java -cp "C:\java";. planetmain
```

they are not saved inside the package.

In the `-cp` option, path for the packages 'sphere' and 'planet' is given  $\rightarrow$  "C:\java"

Since the class path will be changed due to this, the class file for 'ballmain' and 'planetmain' cannot be found in this path. To execute these class files, they have to be accessed. So, their path 'C:\Programs' also has been included by using '.' as it denotes the current path.

## Access privilege:

Members with default access mode within a package cannot be seen outside the package (ie., cannot be accessed). So, if any member within a package has to be used across the package, it must be public, (or protected if the class outside the package is a subclass of this particular class to be accessed).

When a class is public, it must be the only public class declared in the file & the file must have the same name as the class. If 2 classes have to be declared public, save them in 2 separate files.

## I/O Streams

## Stream

### -Sequential & contiguous one way flow of data

- It is an abstract representation of input or output device that is a source or destination for data.

Java does not see any difference between different sources and sinks.

The source / sink may be a disk file, network socket, standard I/O devices like keyboard, monitor or even memory buffer. But the procedure for reading or writing data from and into these sources and sinks is the same.

### I) Byte oriented streams- handling i/p & o/p of bytes

II) character oriented streams- handling i/p & o/p of characters

### Why separate stream for characters?

Java internally stores characters in 16-bit UCS-2 character set format. But the external data source/sink could store characters in other character set formats such as UTF-8, UTF-16, US-ASCII.... Translations have to be done from UCS-2 to other char sets before storing character data in I/O devices. Similarly, while reading character data from I/O devices, they should be converted from other char sets to UCS-2.

All stream classes have been defined in `java.io` package. So, import this package.

### I) Byte Oriented Streams:

a) InputStream <ul style="list-style-type: none"> <li>• ByteArrayInputStream</li> <li>• FileInputStream</li> <li>• FilterInputStream             <ul style="list-style-type: none"> <li>- DataInputStream</li> <li>- BuferedInputStream</li> </ul> </li> </ul>	b) OutputStream <ul style="list-style-type: none"> <li>• ByteArrayOutputStream</li> <li>• FileOutputStream</li> <li>• FilterOutputStream             <ul style="list-style-type: none"> <li>- DataOutputStream</li> <li>- BuferedOutputStream</li> </ul> </li> </ul>
--	--

	- PrintStream
--	---------------

The abstract classes `InputStream` and `OutputStream` define several methods that the other stream classes implement. Two of the most important are `read()` and `write()`.

- `int read()` reads a single byte and returns it as an integer. Returns -1 if end of file is reached or read operation is not successful.
- `int read(byte[] buffer)` reads an entire byte array
- `int read(byte[] buff, int offset, int n)` reads the bytes into the array buff, starting from the position specified by offset and reads a total of 'n' characters.
- `void write(int b)`
- `void write(byte[] buffer)`
- `void write(byte[] buff, int offset, int n)`

## II) Character Oriented Streams:

c) Reader <ul style="list-style-type: none"> <li>• <code>BufferedReader</code></li> <li>• <code>CharArrayReader</code></li> <li>• <code>InputStreamReader</code> <ul style="list-style-type: none"> <li>- <code>FileReader</code></li> </ul> </li> </ul>	d) Writer <ul style="list-style-type: none"> <li>• <code>BufferedWriter</code></li> <li>• <code>CharArrayWriter</code></li> <li>• <code>OutputStreamWriter</code> <ul style="list-style-type: none"> <li>- <code>FileWriter</code></li> <li>- <code>PrintWriter</code></li> </ul> </li> </ul>
--	---

The abstract classes `Reader` and `Writer` define several methods that the other stream classes implement. Two of the most important are `read()` and `write()`.

- `int read()` reads a single character and returns it as an integer. Returns -1 if end of file is reached or read operation is not successful.
- `int read(char[] buffer)` reads a char array
- `int read(char[] buff, int offset, int n)` reads the characters into the array buff, starting from the position specified by offset and reads a total of 'n' characters.
- `void write(int ch)`
- `void write(char[] buffer)`
- `void write(char[] buff, int offset, int n)`

### To read data from KeyBoard:

Using ByteStreamClasses import java.io.*; class keyboardbuffer { public static void main(String args[]) { try { byte[] b=new byte[30]; BufferedInputStream bos=new BufferedInputStream(System.in); bos.read(b); } } }	Using CharacterStreamClasses import java.io.*; class bufferedreader { public static void main(String args[]) { try { InputStreamReader in=new InputStreamReader(System.in); BufferedReader br=new BufferedReader(in); } } }
---	---

<pre>String s=new String(b); System.out.println(s); } catch(IOException e) { System.out.println(e); } } }</pre> <p>-creates a ByteStream that is attached to the std input device, the keyboard, so that data can be read from the keyboard. Reads the byte array and converts to string to display it in a readable format. Using <b>BufferedInputStream reduces the number of disk accesses to read the data. Thus BufferedInputStream class can also be used to wrap around other stream classes for efficiency</b></p>	<pre>System.out.println("enter 'q' for quit"); char c; do { c=(char)br.read(); System.out.print(c); } while(c!='q'); } catch(IOException e) { System.out.println(e); } } }</pre> <p>-created a characterstream associated with keyboard. read() method reads a single character and returns it in the form of an integer and so typecasted to char before display. This is repeated till the user presses 'q'.</p>
--	--

readLine() method of BufferedReader class helps to read a String as a whole  
 -Returns null if it has encountered the end of the stream data like file.

```
import java.io.*;
class stringbufferedReader
{
public static void main(String args[])
{
try
{
InputStreamReader in=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(in);
System.out.println("enter 'quit' to terminate");
String s;
do
{
s=br.readLine(); // reads an entire String
System.out.println(s);
} while(!s.equals("quit"));
}
catch(IOException e)
{
System.out.println(e);
}
}
```



}

Writing can also be easily accomplished with print() and println() methods defined by the PrintStream class

<p>Using PrintStream</p> <pre>import java.io.*; import java.util.*; class printstream_screen {     public static void main(String args[])     {         PrintStream output = new         PrintStream(System.out);         output.print(true);         output.print((int) 123);         output.println((float) 123.456);         output.close();     } }</pre> <p>o/p: true 123 123.456</p>	<p>Using PrintWriter</p> <p><u>Constructors:</u></p> <p>PrintWriter(OutputStream os)</p> <p>PrintWriter(OutputStream os, boolean flushonnewline) - if true, flushing the o/p stream is done automatically whenever a println() method is called</p> <p>If the first type of constructor is used, then close() or flush() method must be invoked. Otherwise data will not be flushed properly.</p> <pre>import java.io.*; class printwriter {     public static void main(String args[])     {         PrintWriter pw=new         PrintWriter(System.out); // can         be used to write in a file         pw.print("welcome to ");         pw.println("Java Programming");         String s="Sample PrintWriter Program";         pw.println(s);         <b>pw.close(); // (or) pw.flush();</b>     } }</pre> <p>o/p: welcome to Java Programming Sample PrintWriter Program</p>
--	--

### Reading and writing files:

To open a file, simply create object of one of these classes

FileInputStream (String filename) throws FileNotFoundException – to read from a file

FileOutputStream (String filename) throws FileNotFoundException - to write to a file

FileReader(String filename) – to open a file to read characters

FileWriter(String filename) - to open a file to write characters into it

To close the file

void close() throws IOException

Example: To read the current Java program

<pre>import java.io.*; class readsamefile {     public static void main(String args[])     throws IOException</pre>	<p><u>Output:</u></p> <pre>import java.io.*; class readsamefile {     public static void main(String[] args)</pre>
---	--

<pre> {     File file1=new File("C:\\Users\\Desktop \\java\\readsamefile.java");     FileInputStream fos=new     FileInputStream(file1); // file object can         //also be passed instead of passing         //the file name      int i;     do     {         i=fos.read();         if(i!=-1)         {             if((char)c!='\n')                 System.out.print((char)c);             else                 System.out.println((char)c);         }     } while(i!=-1);     fos.close(); } </pre>	<pre> throws IOException {     FileReader fr=new     FileReader("readsamefile.java");     int c;     while((c=fr.read())!=-1)     {         if((char)c!='\n')             System.out.print((char)c);         else             System.out.println((char)c);     } } </pre>
---	---

<pre> import java.io.*; class bufferedoutputstream {     public static void main(String args[])     {         try         {             FileOutputStream fos=new FileOutputStream("datafile.txt");             BufferedOutputStream bos=new BufferedOutputStream(fos);             String s=" I am new to Java";             byte buff[]=s.getBytes();             bos.write(buff);             bos.flush();         }         catch(FileNotFoundException e)         {             System.out.println(e);         }         catch(IOException e)         {             System.out.println(e);         }         //*****Reading back from the file*****     } } </pre>
--

```

{
BufferedInputStream bis=new BufferedInputStream(
new FileInputStream("datafile.txt"));
byte buff2[]=new byte[1024];
bis.read(buff2);
String s1=new String(buff2);
System.out.println(s1); // note: Scanner class can also be used to read from
} // file.
catch(IOException e1)
{
    System.out.println(e1);
}
}
}

```

### **DataInputStream & DataOutputStream:**

To read and write primitive data types like int, float, double, char etc They implement DataInput & DataOutput interfaces which contain methods to convert primitive values to or from a sequence of bytes. They can be stacked on top of any InputStream and OutputStream to parse the raw bytes so as to perform I/O operations in the desired data format.

DataOutputStream:

```

final void writeDouble(double value) throws IOException
final void writeBoolean(boolean value) throws IOException
final void writeInt(int value) throws IOException
final void writeChar(char value) throws IOException

```

.....

DataInputStream:

```

final double readDouble() throws IOException
final boolean readBoolean() throws IOException
final int readInt() throws IOException
final char readChar() throws IOException

```

.....

```

import java.io.*;
class dataoutputstream
{
    public static void main(String args[])
    {
        try
        {
            FileOutputStream fos=new FileOutputStream("datafile.txt");
            DataOutputStream dos=new DataOutputStream(fos);
            dos.writeInt(5);
            dos.writeBoolean(true);
            dos.writeDouble(89.5);
        }
        catch(FileNotFoundException e)

```

```

    {
        System.out.println(e);
    }
    catch(IOException e)
    {
        System.out.println(e);
    }
    //*****Reading back from the file*****
    try(DataInputStream dis=new DataInputStream(new FileInputStream("datafile.txt")))
    {
        int i=dis.readInt();
        Boolean b=dis.readBoolean();
        double d=dis.readDouble();
        System.out.println("integer="+i+" Boolean="+b+" double="+d);
    }
    catch(IOException e1)
    {
        System.out.println(e1);
    }
}
}

```

```

import java.io.*;
class writenameinfile
{
    public static void main(String[] args) throws IOException
    {
        BufferedWriter fw=new BufferedWriter(new FileWriter("writename.txt"));
        /* or FileWriterfw=new FileWriter("writename.txt");
        Without buffering, each invocation of
        read() or readLine() could cause bytes to be read from the file,
        converted into characters, and then returned, which can be very inefficient.
        */
        fw.write("Hello!This is from file".toCharArray());
        char[] c=new char[50];
        FileReader fr=new FileReader("writename.txt");
        BufferedReader br=new BufferedReader(fr);
        br.read(c);
        System.out.println(c);
        br.close();
    }
}
o/p: Hello!This is from file

```

To write name, course and RegNo of 3 students into a file, read them back and display the details of the student with reg no 10mit0001.

```
import java.io.*;
```

```

import java.util.*;
class attendancefile
{
public static void main(String[] args) throws IOException
{
    PrintWriter fw=new PrintWriter(new FileWriter("attendance.txt"));
    for(int i=0;i<3;i++)
    {
        System.out.println("enter the name, course, regno");
        Scanner sc=new Scanner(System.in);
        String name=sc.nextLine();
        String course=sc.nextLine();
        String regno=sc.nextLine();
        fw.println(name);
        fw.println(course);
        fw.println(regno);
    }
    fw.close();
    char[] c=new char[50];
    FileReader fr=new FileReader("attendance.txt");
    BufferedReader br=new BufferedReader(fr);
    String name;
    while( (name=br.readLine())!=null)
    {
        String course=br.readLine();
        String regno=br.readLine();
        if(regno.equals("10mit0001"))
        {
            System.out.println(name);
            System.out.println(course);
            System.out.println(regno);
        }
    }
    br.close();
}
}

```

**File class:**

A class which describes the properties of a file instead of creating streams for those files.

Constructors:

File(String directorypathname) will create the obj within the current directory

File(String directory, String filename) will create the obj within the directory specified by 1<sup>st</sup> argument and 2<sup>nd</sup> argument is the child of that directory which can be a file or directory.

Important file methods:

getName() – returns the name of the file

getParent() – returns the name of the parent directory

getPath()  
 getAbsolutePath()  
 canWrite() – returns a Boolean value denoting whether the file can be written  
 canRead()  
 isFile() –returns true if the object is a file  
 isDirectory()- returns true if the object is a directory  
 length() – gives the size of the file  
 lastModified() - gives the date when the file was last modified

<pre> import java.io.*; public class file_methods {     public static void main(String args[])     {         File files=new         File("C:\\Users\\Desktop\\java");         System.out.println(files+         (files.isDirectory()? "is": "is not")         +" a directory");         System.out.println("File exists?"         +files.exists());         System.out.println("Is it the absolute         path?" +files.isAbsolute());         System.out.println("Does the file has         read access?" +files.canRead());          File file2=new         File("java/sampleclass.java");         System.out.println("Is it a file?"         +file2.isFile());         System.out.println("Is it the absolute         path?" +file2.isAbsolute());         System.out.println("Does the file has         read access?" +file2.canRead());         System.out.println("Does the file has         write access?" +file2.canWrite());          File file3=new         File("C:\\Users\\Desktop\\java");         System.out.println("Do both the files         have the same path?"         +files.equals(file3));     } } </pre>	<p>O/P:</p> <pre> C:\Documents          and Settings\Administrator\Desktop\java  is      a directory File exists?false Is it the absolute path?true Does the file has read access?false Is it a file?true Is it the absolute path?false Does the file has read access?true Does the file has write access?false Do both the files have the same path?true </pre>
---	--

A directory in java is also treated as a file with one additional property – using the list() method, a list of filenames within that directory can be examined.

list() method can be called on the directory object to extract the list of other files and directories inside it.

Two forms:

- i) String[] list() – the list of files is returned in an array of String objects.

```
import java.io.*;
public class sub_directories
{
    public static void main(String args[]) throws IOException
    {
        int j=0;
        File files=new File("C:\\Users\\Desktop\\java");
        System.out.println(" *****list of members in the directory*****");
        String s1[]=files.list();
        for(String i: s1)
            System.out.println(i);
        System.out.println("***** list of sub-directories *****");
        for(int k=0;k<s1.length;k++)
        {
            File f=new File("C:\\Users\\Desktop\\java\\" +s1[k]);
            if(f.isDirectory())
                System.out.println(s1[k]);
        }
    }
}
```

Inside the for loop, a File object is created for each String and checked if it is a directory or not.

The second form of list is used to list only the name of the specific types of files eg., only .txt files or only .c files and so on

- ii) String[] list(FileNameFilter obj)

FileNameFilter:

It is an interface which defines only a single method accept(). It is used to list only the files that match a certain file name pattern or filter.

As per the syntax of list() method, an obj of FileNameFilter is required. But FileNameFilter is an interface for which objects cannot be created.

So, first define a class by implementing the FileNameFilter interface.

Define the method accept() for this class and create an obj for this class

Use this object as argument for the list() method.

boolean accept(File directory, String Filename) – returns true for those files in the directory object( given as 1<sup>st</sup> argument) which matches with the filename (specified as 2<sup>nd</sup> argument) else returns false.

Example: To display files with .class extension

<pre>import java.io.*; class filter implements FileNameFilter {     String ext;</pre>	<p>-defines filter class by implementing FileNameFilter interface. Since it implements this i/f, the only method accept() of that</p>
---	---

<pre>public boolean accept(File f, String name) { return name.endsWith(ext); } public filter(String ext) { this.ext="."+ext; } } class filenamefilter { public static void main(String args[]) throws IOException { File files=new File("C:\\Users\\Desktop\\java"); System.out.println(" *****list of members in the directory using listFiles()*"); <b>filter only=new filter("class");</b>     <b>String s[]=files.list(only);</b>     for(String i:s)     {         System.out.println(i);     } } }</pre>	<p>interface should be defined.</p> <p>Here, it is checked whether the filename ends with “.class” or not. It returns true if the extension of the file name is “.class”.</p>     <p>First create file obj for the directory u want to check, so that list() method can be used for it. Next create an object for filter class to pass it as argument for list() method. Pass this obj as the argument for list() method. Now for each file in the list, the accept() method will be invoked implicitly.</p>
--	---

## Java Applets

## Applets:

- Small applications that are accessed on an Internet server, transported over the Internet, automatically installed and run as part of a web document.
  - have limited access to the resources of the client machine to which it arrives, so that it can produce a GUI (Graphical User Interface) and run complex computations without introducing the risk of viruses or breaching data integrity.
- Eg., An online log-antilog calculator or compound-interest calculator which we download (probably from Google server) and run in our machine, to get the result for our inputs.

Two types of applets:

- 1) Applets based on Applet class which use the AWT(Abstract Window Toolkit) to provide the GUI (used when a very simple user interface is needed)
- 2) Applets based on swing class JApplet which offers a richer and easier to use interface than does the AWT. However, JApplet inherits Applet.

## Two ways of deploying applets:



- 1) Using JNLP (Java Network Launch Protocol). They are not needed for simple applications but for real world applets, JNLP is the best choice.
- 2) Specifying an applet directly in an HTML file without the use of JNLP

Important differences between applets and other programs:

- All applets are subclasses of the class Applet
- They are not standalone programs but they run within either a web browser or an appletviewer (a tool provided by JDK itself)
- Applet programs do not start execution at main(). Instead, they begin execution when the name of its class is passed to an appletviewer or to a network browser.
- Output to the applet window is not by System.out.println() but using AWT methods like drawstring().
- Applets interact with the user through the AWT and not through the console-based I/O classes

Applet Architecture:

- Applets follow **event-driven programming** (and not procedural programming). An applet will wait until an event occurs. Once an event occurs, the Java Runtime system notifies the applet about the event, by calling an event handler that has been provided by the applet. In response to the notification, the applet must take appropriate action and then quickly return. **It should not maintain control for a long period of time.** In situations, where the applet needs to perform a repetitive task on its own (eg., displaying a scrolling message), an additional thread of execution must be started.
- In console-based programs, when the program needs input, it will prompt the user and read the input data. But in applets, the user interacts with the applet when he wants. These interactions are sent to the applet as events to which the applet must respond. Eg., when the user clicks the mouse while the applet's window has input focus, a mouseclick event is generated.

Applet class:

<pre> graph BT     Applet --&gt; Container     Frame --&gt; Window     Window --&gt; Container     </pre>	<p><b>Important methods of Component class:</b></p> <ul style="list-style-type: none"> <li>-setSize(int width, int height)</li> <li>-setBackground(Color c)</li> <li>-setVisible(boolean b)</li> </ul> <p><b>Important methods of abstract class Container:</b></p> <ul style="list-style-type: none"> <li>-add(Component comp)</li> <li>-setLayout (LayoutManager mgr)</li> </ul>
---	--

Four important methods of Applet class:

- 1) void init():
    - For any applet program, this is the method that is called first when it begins its execution and called only once during the runtime of an applet. Variables can be initialized here.
  - 2) void start():
    - called automatically after init() (i.e., when the applet first begins execution)
    - also called when the applet should restart after it has been stopped ( i.e., called each time an applet's HTML document is displayed on screen. So, if a user leaves a web page and comes back, the applet resumes execution at start())
  - 3) void stop():
    - called by the browser to suspend execution of the applet. Once stopped, an applet is restarted only when the browser calls start() (i.e., called when a web browser leaves the HTML document containing the applet - for example, when it goes to another page)
  - 4) void destroy():
    - called by the browser just before an applet is terminated (i.e., when the environment determines that the applet needs to be removed completely from memory).
    - can be overridden if any clean up has to be done prior to its destruction.
- Note: ***Default implementations for all these methods are provided. So, applet programs need not override those methods which they do not use.***

#### Methods of awt Component class:

- 5) void paint():
  - takes one parameter of type Graphics which describes the Graphics environment.
  - called by the JVM implicitly in two circumstances – one when the first time applet window is created and displayed and the other when the window is resized (by dragging the border with mouse) by the user or when the window is minimized and then restored.
  - this method should be overridden to change the way a Component is drawn.
- 6) void repaint():
  - called whenever the applet window has to be redrawn (updated) with the new data. (i.e., in the middle of the coding, if the paint() method has to be called, to draw some graphics on the frame or applet window, it is permitted to call only the **repaint()** method and not the paint() method directly.)
  - Why should we call repaint() and not the paint() method directly?
  - Before the window is to be drawn with new data, the old data must be erased, else, both will get overlapped and finally the data will not be readable. This automatic erasing is done by the **repaint()** method. How?
  - The **repaint()** method calls automatically **update()** method and in turn update() method calls paint() method.
  - repaint() -> update() -> paint()**
  - Four forms of repaint() method:
  - i) void repaint()
    - called whenever the entire window should be repainted
  - ii) void repaint(int x, int y, int width, int height)
    - repaints the rectangular region whose left-top corner point is specified by x and y and length and breadth are specified by width and height
  - iii) void repaint(long maxdelay)
    - calls the update() method, within maxdelay milliseconds. If the time elapses before update() can be called, it is not called

iv) `repaint(long maxdelay, int x, int y, int width, int height)`

Repaints the specified rectangle of the applet window within maxdelay milliseconds.

7) `void update():`

**clears** the earlier drawings or contents on the applet window. On the cleared surface, the `paint()` method draws afresh with the latest graphics. If the programmer is allowed to call `paint()` method directly, there is every possibility that he may forget to call the `update()` method explicitly. To avoid this and to make Java as a **robust language**, the designers are not allowed to call `paint()` directly.

1) Program to display a message in an applet:

<pre>import java.applet.*; import java.awt.*; public class sampleapplet extends Applet {     public void paint(Graphics g)     {         g.drawString("Welcome to Applet Programming", 10,20);     } }</pre>	<p>Every applet program should extend the Applet class. Although the <code>awt Component</code> class is the ultimate super class for the Applet class, it is not defined in <code>awt</code> package. So, import <code>applet</code> package wherein the Applet class has been defined. <code>paint()</code> is a method of <code>awt Component</code> class and also <code>Graphics</code> is a class of <code>awt</code> package. In order to work with these, import <code>awt</code> package also.</p>
--	---

-applet programs need not have `main()` method, as the execution does not start there.

-as default implementations of `init()`, `start()`.... are implicitly available for any applet program, need not define them again, unless some modifications are required.

-However, `paint()` method must be defined. It takes a single argument- a reference to the `Graphics` class. This argument is needed, because the `drawstring()` method used to display messages on to applet window, belongs to `Graphics` class and hence the `Graphics` reference variable is used to invoke the `drawString()` method [`System.out.println()` cannot be used to display the message].

Syntax of `drawstring()`:

`drawstring(String msg, int x, int y)`- displays the String beginning at the specified x, y location and this method is called within `update()` or `paint()` method.

In a Java window, the upper left corner is 0, 0. When we move along the horizontal direction towards the right, the x value gets increased. When we move along the vertical direction downwards, the y value gets increased.

0,0	1	2	3	4	5	6	7	8	9
1									
2									
3									
4									
5									
6						*			
7									

Assuming the applet window to be a grid of pixels as shown, the point denoted by star symbol represents (6,5). i.e., the x coordinate and y coordinate of the point are 6 and 5 pixels away from the upper left corner (0, 0)

drawString() does not recognize '\n' character. So, if a line of text has to be started on another line, it must be done manually, specifying the precise x, y location where the line has to begin.

Compile the program as usual → javac sampleapplet.java → sampleapplet.class will be generated.

#### Execution:

First write a HTML program containing a tag that loads the applet.

<pre>&lt;applet code="sampleapplet" width=400 height=400&gt; &lt;/applet&gt;</pre>	Save this file with .html extension. Example 'sample.html'
--	---

HTML: Hyper Text Mark Up Language – Language for describing the web pages.( i.e., tells how your web page should look like)

-html documents contain html tags and plain text.

-html tags are keywords surrounded by angle brackets like<html>,<head>,<applet>.....

-html tags normally come in pairs like <b> and </b> where <b> is called the start tag or opening tag and </b> is called the end tag or closing tag.

<applet> tag helps for loading the applet. (<object> tag can also be used)

There are several attributes for the <applet> tag of which three are mandatory –code, width and height.

code attribute – to specify the name of the file containing the applet's compiled .class file.

width and height attributes – to specify the width and height of the display area used by the applet window. (size of the applet window)

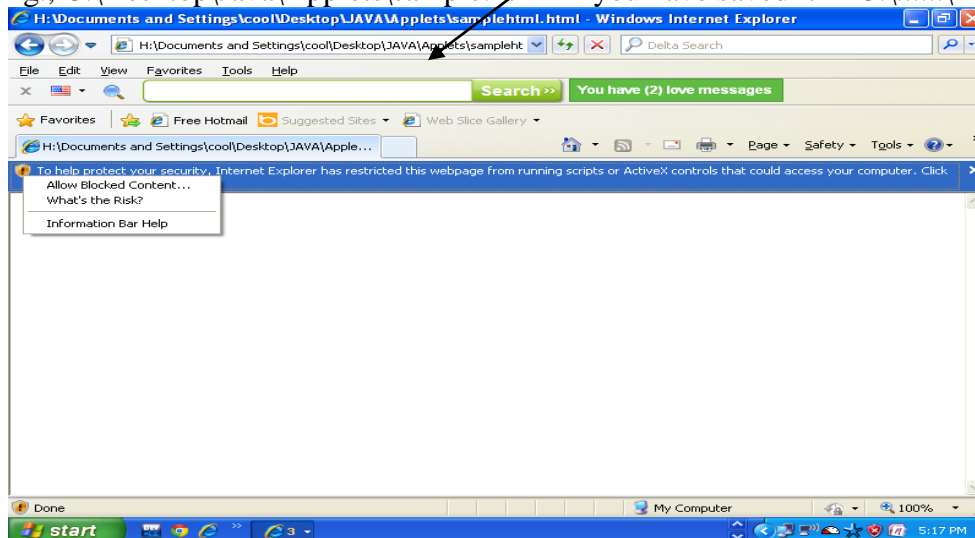
Now execution can be done in two ways

i) can be executed within a Java-compatible web browser

Load the html file into a browser such as GoogleChrome, IE or FireFox, which causes the 'sampleapplet' to be executed. The purpose of a web browser is to read the HTML documents and display them as web pages. The browser does not display the tags but uses the tags to display the contents of the page.

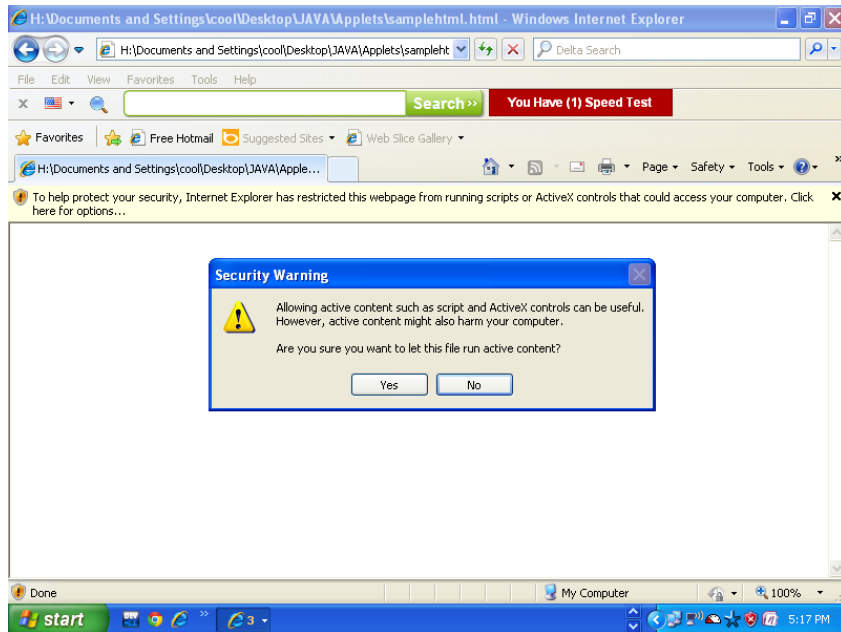
Give the entire path for the html file in the URL of the browser:

Eg., C:\Desktop\Java\Applets\sample.html if you have saved it in C:\.....\Applets Folder

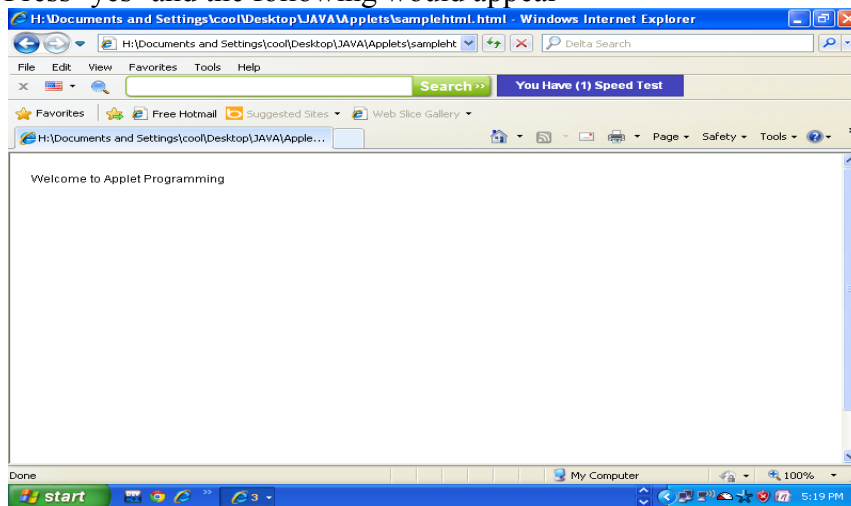


Click on Allow Blocked Content ( this is when we execute in Internet Explorer)

## Core Java

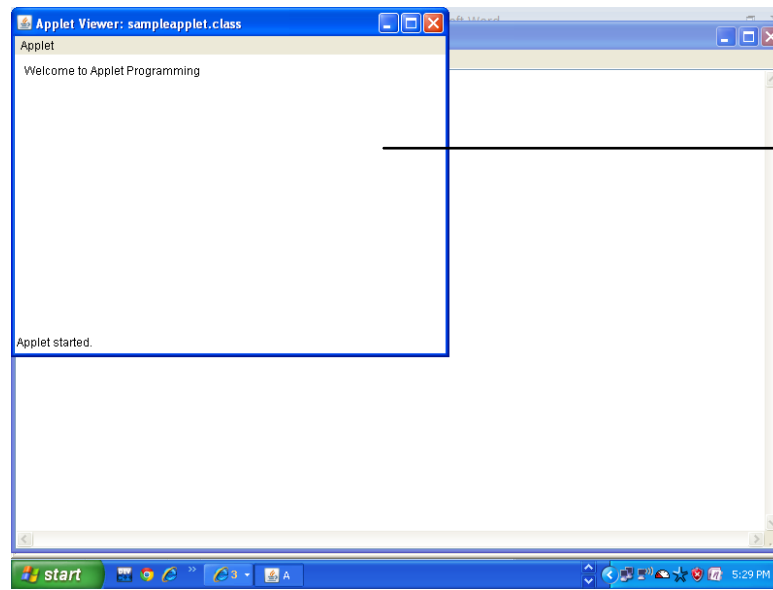


Press 'yes' and the following would appear



ii) using the appletviewer tool:

C:\Desktop\Java\Applets\ appletviewer sample.html



This is the applet window whose width & height have been specified as 400 and 400 pixels in the <applet> tag

iii) There is a third option which is the most convenient way.

Include the HTML code containing the applet tag as a comment, within the Java source file itself.

```
import java.applet.*;
import java.awt.*;
/*
<applet code="sampleapplet" width=400
height=400>
</applet>
*/
public class sampleapplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Welcome to Applet
Programming", 10,20);
    }
}
```

Save as sampleapplet.java

Compile as usual

Execute as appletviewer sampleapplet.java

The same applet window as shown above will be displayed.

Note: As the applet tag is given as a comment, the Java compiler will ignore it and so no error will be reported, for using tags.

However, the appletviewer will encounter the applet tag, can recognize it and so executes it.

Full form of the applet tag:

<applet [codebase=codebaseURL] **code=appletfile** [alt=alternate text] [name=applet instance name] **width=pixels height=pixels** [align=alignment] [vspace=pixels] [hspace=pixels]>

Attributes mentioned within square brackets are optional and hence not needed to be mentioned in the applet tag unless it is required.

Codebase- directory that will be searched for the applet's executable class file. If not specified, the HTML document's URL directory itself is used as the code base

Name- to specify the name for the applet instance. This helps other applets within the same page to find them by name and communicate with them

Align-to specify the alignment of the applet. Possible values-LEFT, RIGHT,TOP,BOTTOM, MIDDLE, BASELINE, TEXTTOP , ABSMIDDLE AND ABSBOTTOM  
vspace & hspace- to specify the space in each side of the applet, in pixels

#### Passing parameters to the applets:

Parameters can be passed to the applet, through the applet tag.

```
<applet code=.....>
  [< param name= name of parameter1  value=value for parameter1>]

  [ <param name=name of parameter2 value=value for parameter2>]
  .....
</applet>
```

Parameters can be retrieved using getParameter() method.

Syntax: String getParameter(String) - returns the value of the specified parameter in the form of a String object.

#### Using status window:

showStatus() is used to display in the status window of the browser or applet viewer on which it is running.

Syntax:

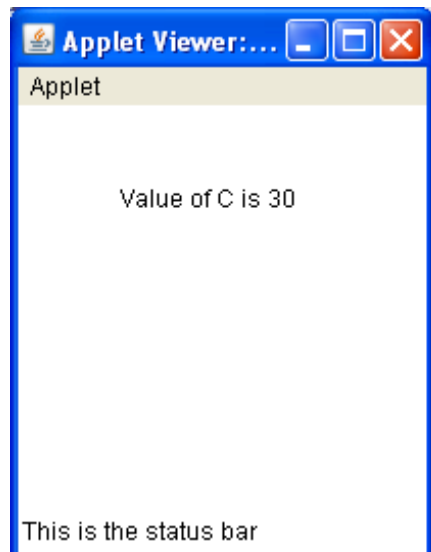
showStatus (String msg)

Status window: Can be used to give the user, feedback about what is occurring in the applet.

#### 2. Program to use parameters and display message in the status window:

```
import java.awt.*;
import java.applet.*;
/*
<applet  code="statusapplet"  width=200
height=200>
<param name=a value=10>
<param name=b value=20>
</applet>
*/
public class statusapplet extends Applet
{
    int c,a,b;
    public void init()
    {
        a=Integer.parseInt(getParameter("a"));
        b=Integer.parseInt(getParameter("b"));
    }
    public void paint(Graphics g)
    {
        c=a+b;
        g.drawString("Value of C is "+c,50,50);
        showStatus("This is the status bar");
    }
}
```

Output Window:



}	
---	--

In the above program, the values of 'a' and 'b' are passed as parameters to the applet.

<param name =a value=10><param name=b value=20>

Note that the values of the parameters are retrieved within the applet through `getParameter()`. `showStatus()` method displays the message "this is the status bar" in the status window.

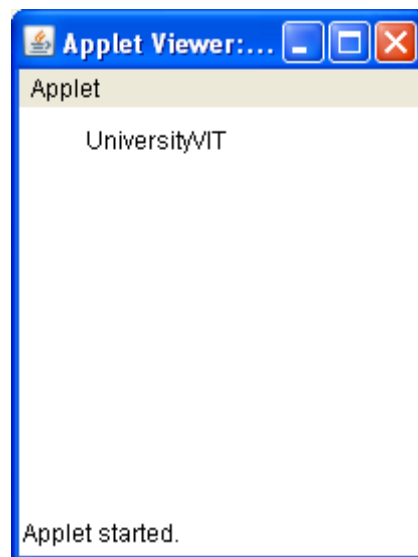
3. Program to display a scrolling message continuously:

For this, as the applet must perform a repetitive task, create a separate thread to do this.

Also invoke `repaint()` method, for every repetition, as each time the updated message has to be displayed.

Eg., initially the message should be "VIT UNIVERSITY", then →IT UNIVERSITY V, next→T UNIVERSITY VI and so on.

```
import java.awt.*;
import java.applet.*;
/*
<applet    code="bannerapplet"    width=200
height=200 align="ABSMIDDLE">
</applet>
*/
public class bannerapplet extends Applet
implements Runnable
{
    Thread t=null;
    String msg="VIT University";
    public void init()
    {
        t=new Thread(this);
        t.start();
    }
    public void run()
    {
        while(true)
        {
            try
            {
                repaint();
                Thread.sleep(250);
            }
            catch (InterruptedException e)
            {
            }
        }
    }
    public void paint(Graphics g)
    {
```



`repaint()` ultimately invokes `paint()`. In the



<pre>char c=msg.charAt(0); msg=msg.substring(1,msg.length()-1)+c; g.drawString(msg,200,200); } }</pre> <p>The applet class should implement Runnable interface in order to create a thread object. It cannot extend Thread class as it has already extended Applet class. The thread has to repeatedly scroll the msg from right to left. So, call start() for invoking the run() method and in the run() method, run an infinite loop and call repaint() to continue painting the message. At the end of each iteration, make the thread sleep for 250 milliseconds so that the msg does not move too fast thus making it unreadable. Enclose the sleep() method in try-catch block as it would throw InterruptedException.</p>	<p>paint() method, the very first character in msg is removed and appended at the end of the remaining portion of the msg. That msg is displayed. For the next call of paint() method, the current first character in msg is removed and appended at the end of the remaining portion of the msg. This continues infinitely.</p>
--	--

#### Color class:

- Defined in awt package
- Color class defines predefined constants to represent the following standard colors.

Color.black, Color.white, Color.red, Color.green, Color.blue, Color.yellow, Color.orange, Color.pink, Color.magenta, Color.cyan, Color.gray, Color.lightGray, Color.darkGray

#### Using Color class:

Now to set the background color of the applet window with any of these colors,

- **void setBackground(Color c)**

To set the foreground color ( to set the color in which text is shown)

- **void setForeground(Color c)**

To get the current color settings of the Background and Foreground,

**Color getBackground()** Both the methods return a Color object.

**Color getForeground()**

Apart from these standard colors, custom colors (user defined colors) can also be created by using the constructors of the Color class.

- Color(int red, int green, int blue) → helps to specify the color as a mix of red, green and blue. They can have any value within the range 0 to 255.
- Color(int rgbvalue) → single integer that contains the mix of red, green and blue packed into an integer
- Color(float red, float green, float blue) → 0.0 to 1.0 that specify the relative mix of r,g,b

Once a custom color is created, it can be used to set the foreground &/ background color by using the `setForeground()` and `setBackground()`.

Also, the color of the Graphics object can be changed by calling the Graphics method `setColor()`

**void setColor(Color newcolor)**

The current color of the Graphics object can be obtained by using

**Color getColor()**

The individual red, green and blue components of a color can also be obtained by using the following methods:

**int getRed(), int getGreen(), int getBlue(),**

**int getRGB()**->to obtain a packed RGB representation of a color

Font class:

Creating the desired font by using the constructor:

`Font(String fontname, int fontstyle, int fontsize)` -> specifies the size of the font in points

Font.ITALIC)      ↳ Specifies style of the font (Font.PLAIN, Font.BOLD and

Font.ITALIC)      ↳ Name of the desired font

Eg., `Font f=new Font("TimesNewRoman", Font.BOLD, 24);`

As the Color object is used for setting the color of the Graphics object, the Font object can be used for setting the Font of the Graphics object.

**void setFont(Font obj)**

The current font of the Graphics object can be obtained by calling `getFont()`

**Font getFont()**

The individual components of the Font class can be obtained by the following methods

**getSize(), getStyle(), getName()**

#### 4. Program to use Font and Color class:

```

import java.awt.*;
import java.applet.*;
/*
<applet code="fontapplet" width=400
height=400>
</applet>
*/
public class fontapplet extends Applet
{
    public void paint(Graphics g)
    {
        Font f=new Font("Arial",Font.BOLD,24);
        g.setFont(f);
        g.drawString("hi this is written with the
new font",10,100);
        Font f2=g.getFont();
        g.drawString(f2.getName()+" "
+f2.getSize()+" " + f2.getStyle(), 10,150);
        Color c1=new Color(50,200,200);
        g.setColor(c1);
        Color c=g.getColor();
        g.drawString(c.getRed()+" "+ c.getGreen()
+" "+ c.getBlue(), 10, 200);
    }
}

```

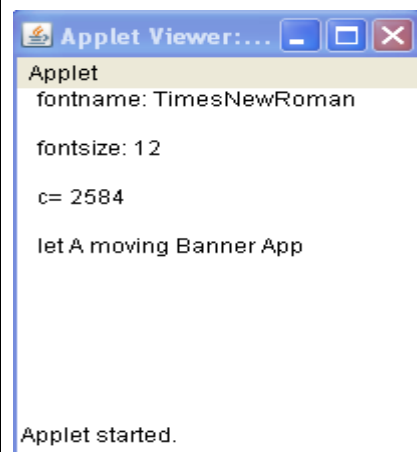


5. Program to display a moving banner, print the fibonacci sequence continuously, retrieve and display the font components passed through applet parameters:

```

import java.awt.*;
import java.applet.*;
/*
<applet code="parameterapplet" width=200
height=200>
<param name=fontname value=TimesNewRoman>
<param name=fontsize value=12>
<param name=a value=-1>
<param name=b value=1>
<param name=msg value="A moving Banner Applet"
">
</applet>
*/
public class parameterapplet extends Applet
implements Runnable
{
    Thread t=null;
    String fontname;

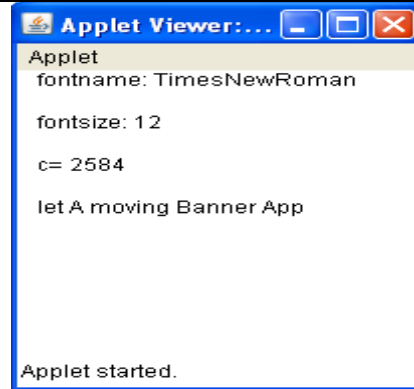
```



```

String msg;
int fontsize;
int a, b, c;
public void start()
{
String s;
t=new Thread(this);
fontname=getParameter("fontname");
s=getParameter("fontsize");
fontsize=Integer.parseInt(s);
msg=getParameter("msg");
s=getParameter("a");
a=Integer.parseInt(s);
s=getParameter("b");
b=Integer.parseInt(s);
t.start();
}
public void run()
{
while(true)
{
try
{
repaint();
Thread.sleep(250);
}
catch(InterruptedException e)
{
}
}
}
public void paint(Graphics g)
{
g.drawString("fontname: "+fontname,10,10);
g.drawString("fontsize: "+fontsize,10,40);
c=a+b;
g.drawString("c= "+c,10,70);
a=b;
b=c;
msg=msg.substring(1,msg.length())
+msg.charAt(0);
g.drawString(msg,10,100);
}
}

```



Since continuous output should be there, thread object should be created. Font name and font size are passed through applet parameters and retrieved using `getParameter()` and displayed. 'a' and 'b' values are also got through applet parameters and in the `run()` method of the thread, `repaint()` is called repeatedly with specified time interval between each call. In the `paint ()` method, font name and size are displayed. 'c' value is calculated and displayed. Then 'a' and 'b' values are modified to be used for the next iteration.

**getDocumentBase():** returns an URL object that represents the directory of the html file that started the applet

**getCodeBase():** returns an URL object representing the directory from which the applet's class was loaded.

### 5. Program to display the directory of the applet class file and the html file

<pre>import java.awt.*; import java.applet.*; import java.net.*; /* &lt;applet code="codebaseapplet" width=400 height=400&gt; &lt;/applet&gt; */ public class codebaseapplet extends Applet { String msg; public void paint(Graphics g) { URL codeurl=getCodeBase(); msg=codeurl.toString(); g.drawString(msg,10,30); URL docurl=getDocumentBase(); msg=docurl.toString(); g.drawString(msg,10,80); } }</pre>	<p>Output:</p> <div data-bbox="820 483 1485 892" style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <pre>file:/C:/Desktop/Java/Applets file:/C:/Desktop/Java/Applets/codebaseapplet.java</pre> </div> <p>Note: URL class has been defined in 'net' package. So import that.          getCodeBase() returns the folder in which the java .class file has been stored          getDocumentBase() returns the java file name as the java file only hosts the html code.          Convert the URL object to String and use it in drawString()</p>
---	---

### Graphics class in detail:

So far we have been using only the drawstring() method of Graphics class. But there are also other methods that can be used for drawing desired shapes.

These methods can be used to draw edge-only (without filling with color) or filled shapes. Objects are drawn and filled in the currently selected Graphics color, which is black by default.

When a graphics object is drawn exceeding the dimensions of the window, the output is automatically clipped.

Methods:

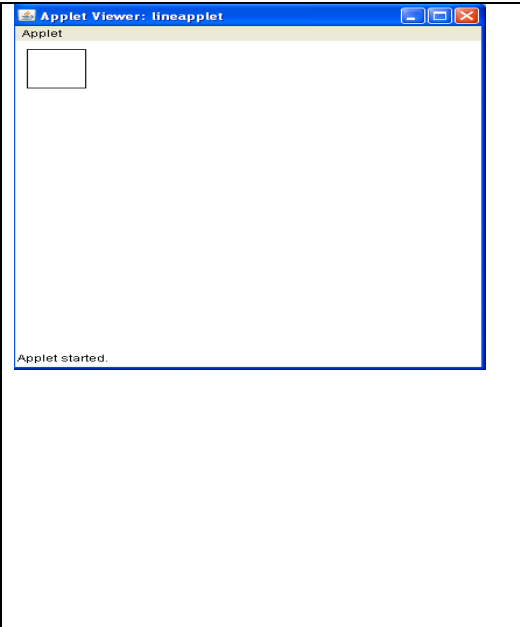
i) For drawing lines:

**void drawLine(int startx, int starty, int endx, int endy)**  
 startx, starty → x,y coordinates of the starting point of the line  
 endx, endy → x,y coordinates of the ending point of the line

```

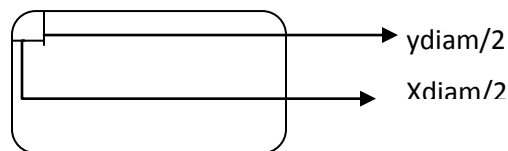
6) import java.awt.*;
import java.applet.*;
/*
<applet   code="lineapplet"   width=400
height=400>
</applet>
*/
public class lineapplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawLine(10,10,60,10);
        g.drawLine(60,10,60,60);
        g.drawLine(60,60,10,60);
        g.drawLine(10,60,10,10);
    }
}

```

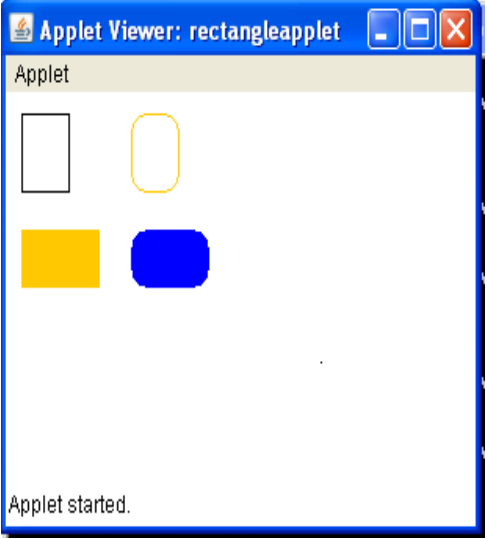


ii) For drawing rectangles:

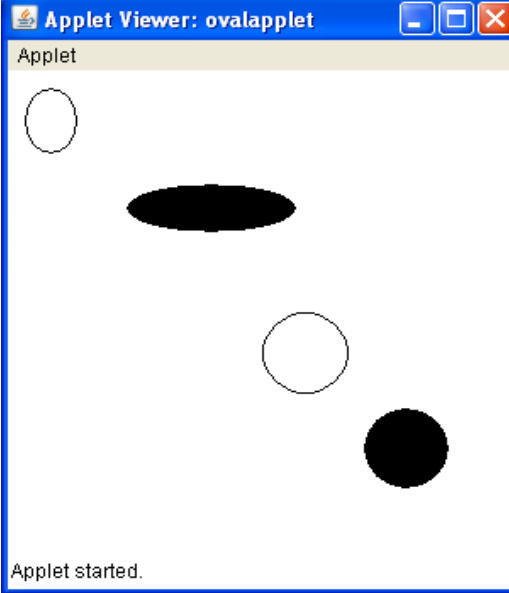
- a) void drawRect(int top, int left, int width, int height)  
-draws a edge-only rectangle whose starting point coordinates are top & left and the width & height of the rectangle are given by the last two parameters.
- b) void fillRect(int top, int left, int width, int height)  
-draws a filled rectangle
- c) void drawRoundRect(int top, int left, int width, int height, int xdiam, int ydiam)  
-draws a rectangle with rounded edges. The diameter of those rounded corners are given by xdiam and ydiam.



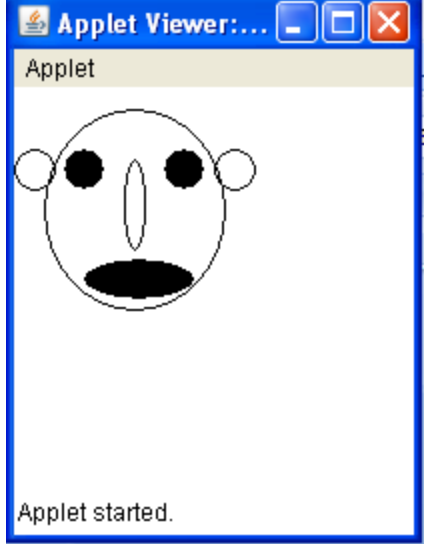
- d) void fillRoundRect(int top, int left, int width, int height, int xdiam, int ydiam)  
-draws a filled round edged rectangle  
Note: To draw a square, the width and height values should be the same

<pre> 7) import java.awt.*; import java.applet.*; /* &lt;applet code="rectangleapplet" width=300 height=200&gt; &lt;/applet&gt; */ public class rectangleapplet extends Applet {     public void paint(Graphics g)     {         g.drawRect(10,10,30,40);         //30 is the width and not the         ending point         g.setColor(Color.orange);         g.fillRect(10,70,50,30);         g.drawRoundRect(80,10,30,40,20,20);         g.setColor(Color.blue);         g.fillRoundRect(80,70,50,30,20,20);     } } </pre>	
--	--

iii) For drawing Oval shapes:

<p>a) void drawOval(int top, int left, int width, int height)</p> <p>b) void fillOval(int top, int left, int width, int height)</p> <p>Ellipse (oval shape) is drawn within a bounding rectangle whose upper left corner is specified by top, left &amp; the width, height specify the breadth and length of the rectangle. <u>To draw a circle, specify a square as the bounding figure.</u></p>	
<pre> 8) import java.awt.*; import java.applet.*; /* &lt;applet code="ovalapplet" width=300 height=300&gt; &lt;/applet&gt; */ public class ovalapplet extends Applet {     public void paint(Graphics g)     {         g.drawOval(10,10,30,40);         g.fillOval(70,70,100,30);         g.drawOval(150,150,50,50);         g.fillOval(210,210,50,50);     } } </pre>	

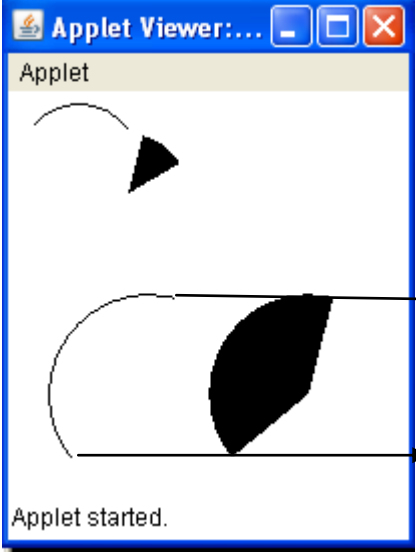
9) Write a program to draw a face using drawOval shape.

<pre>import java.awt.*; import java.applet.*; /* &lt;applet code="faceapplet" width=200 height=200&gt; &lt;/applet&gt; */ public class faceapplet extends Applet {     public void paint(Graphics g)     {         g.drawOval(15,10,90,100); // outline         g.fillOval(25,30,20,20); //eye         g.fillOval(75,30,20,20); //eye         g.drawOval(55,35,10,45); //nose         g.fillOval(35,85,55,20); //mouth         g.drawOval(0,30,20,20); //ear         g.drawOval(100,30,20,20); //ear     } }</pre>	
--	--

iv) To draw arcs:

<p>a) void drawArc(int top, int left, int width, int height, int startangle, int sweepangle)</p> <p>b) void fillArc(int top, int left, int width, int height, int startangle, int sweepangle)</p> <p>the arc will be bounded by the rectangle whose dimensions are given by the first four parameters. The arc is started from the startangle and drawn through the angular distance specified by sweep angle. Angles are measured in degrees. Zero degrees is on the horizontal (at 3 O'clock position in the wall clock) and 90 degree is at 12 O'clock position &amp; so forth</p> <p>If the sweep angle is positive, the arc is drawn in counter clockwise direction and if it is negative, it is drawn in the clock wise direction. Therefore, to draw an arc from 2 O'clock to 10 O'clock, the startangle should be 35 and the sweep angle should be 105</p>
--



<pre> 10) import java.awt.*; import java.applet.*; /* &lt;applet code="arcapplet" width=200 height=200&gt; &lt;/applet&gt; */ public class arcapplet extends Applet {     public void paint(Graphics g)     {         g.drawArc(5,5,60,60,35,105);         g.fillArc(30,20,60,60,75,-45);         // fills the arc in the direction it         // is moved         g.drawArc(20,100,100,100,75,145);         g.fillArc(100,100,100,100,75,145);     } } </pre>	 <p style="position: absolute; left: 830px; top: 220px;">75°</p> <p style="position: absolute; left: 820px; top: 280px;">Moved 145° from 75° (so, the arc ends at 220°</p>
--	--

V) To draw polygon:

a) void drawPolygon(int[] x,int[] y, int numpoints)

b) void fillPolygon(int[] x,int[] y, int numpoints)

The x array stores the x-coordinates of all the end points of the polygon

The y array stores the y-coordinates of all the end points of the polygon

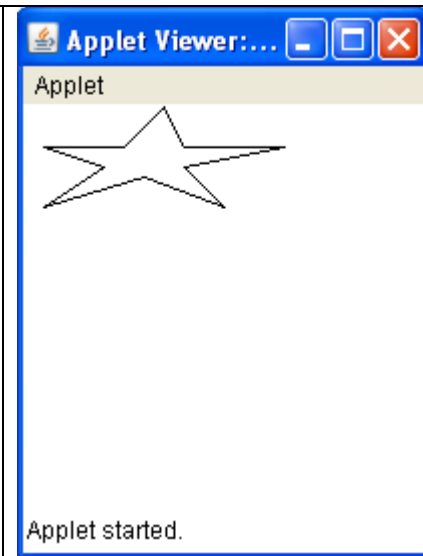
Numpoints denotes the total number of endpoints

Eg., To draw the start mentioned below, 10 end points are needed altogether. So, numpoints should be 10 and x and y array also contain 10 coordinate values each

```

11) import java.awt.*;
import java.applet.*;
/*
<applet code="polygonapplet" width=200 height=200>
</applet>
*/
public class polygonapplet extends Applet
{
    public void paint(Graphics g)
    {
        {
            int[] xcoor={ 10,50,70,80,130,80,100,60,10,40};
            int[] ycoor={ 20,20,0,20,20,30,50,35,50,30};
            g.drawPolygon(xcoor,ycoor,10);
        }
    }
}

```



## AWT Controls

### Controls:

Components that allow a user to interact with the application in various ways (Eg., a commonly used control is the push button)

The Layout manager automatically positions components within a container.

The LayoutManager is set by the setLayout() method. If no call to setLayout( ) is made, then the default LayoutManager is used

AWT supports the following types of controls:

- Labels, push buttons, check boxes, check box groups, lists, scroll bars, text fields etc.

### Adding Controls:

Create an instance of the desired control and then add it to a window by calling add() method, defined by the Container class.

Syntax: **Component add(Component obj)** // obj is the instance of the control to be added

### Removing Controls:

- a) **void remove(Component obj)** // removes a control from a window when the control is no longer needed
- b) **void removeAll()** // removes all the controls

Almost all the AWT controls may throw a HeadlessException when an attempt is made to instantiate a GUI component in a non-interactive environment. (such as one in which no display , mouse or keyboard is present)

Label is the only passive control (i.e., they do not generate events). All other controls generate events when they are accessed by the user.

Example: When a pushbutton is clicked, an event is sent that identifies the pushbutton.

Table of Controls, their constructors and the important methods:

Class name of the control	Constructors	Methods
1) Label	a) <b>Label()</b> - blank Label created b) <b>Label(String str)</b> - creates a label containing the String specified by str which is left justified c) <b>Label(String str, int align)</b> - The label contains the String str and aligns the String according to 'align' align can take any of the 3 constants- Label.LEFT, Label.RIGHT, Label.CENTER	i) <b>void setText(String str)</b> – the text in a label can be set or changed ii) <b>String getText()</b> - to obtain the current text iii) <b>void setAlignment(int align)</b> – to set the alignment of the String within the Label iv) <b>int getAlignment()</b>
2) Button	a) <b>Button()</b> – empty button created b) <b>Button(String str)</b> – button created with a name	i) <b>void setLabel(String str)</b> ii) <b>String getLabel()</b>
3) Checkbox -to turn an option on or off. Consists of a small	a) <b>Checkbox()</b> b) <b>Checkbox(String str)</b> c) <b>Checkbox(String str, boolean b)</b> - if true, the box is initially checked d) <b>Checkbox(String str, boolean b,</b>	i) <b>void setLabel(String str)</b> ii) <b>String getLabel()</b> iii) <b>boolean getState()</b> – to get the current state of the checkbox iv) <b>void setState(Boolean on)</b> – sets the

box that can either contain a check mark or not. Its state is changed by clicking on it. A Label is generally associated with it to describe the option	<b>CheckboxGroup cbg)</b> – cbg is the name of the CheckboxGroup to which it belongs. If the Checkbox is not part of a group, then CheckboxGroup must be null <b>e)Checkbox(String str, CheckboxGroup cbg, boolean b)</b>	current state of the checkbox
4)Checkbox Group- to create a set of mutually exclusive checkboxes. i.e., at any point of time, only one Checkbox can be checked	<b>CheckboxGroup()</b>	i) <b>Checkbox getSelectedCheckbox()</b> ii) <b>void setSelectedCheckbox(Checkbox cb)</b> - 'cb' represents the checkbox that has to be selected. Previously selected checkbox will be turned off
5) List-created to show any number of choices in the visible window	a) <b>List()</b> b) <b>List(int numRows)</b> – specifies the number of entries in the List that will always be visible c) <b>List(int numRows,boolean multipleselect)</b> – if true, user may select 2 or more items at a time. If false, only one item can be selected	i) <b>void add(String itemname)</b> – adds the item specified by itemname at the end of the List ii) <b>void add(String itemname, int index)</b> – adds the item at the specified index iii) <b>String getSelectedItem()</b> – returns null, if no selection made or if more than 1 item selected iv) <b>String[] getSelectedItems()</b> - for lists that allow multiple selections v) <b>int getSelectedIndex()</b> - returns the index of the selected item. Returns -1, if no item or more items are selected vi) <b>int[] getSelectedIndexes()</b> -returns an array containing the indexes of the currently selected items vii) <b>int getItemCount()</b> – returns the nr of items in the List viii) <b>String getItem(int index)</b> - returns the name associated with the item at that index
6) Scrollbar	a) <b>Scrollbar()</b> b) <b>Scrollbar(int style)</b>	i) <b>void setValues(int initialvalue, int visibleamt, int min, int max)</b> – if the

	<p>c) <b>Scrollbar(int style, int initialvalue, int visibleamount, int min, int max)</b> where style-&gt; Scrollbar.VERTICAL, Scrollbar.HORIZONTAL  initialvalue-&gt; initial value of the Scrollbar  visibleamount-while scrolling down the text, it represents the amount of text visible  min,max-&gt; minimum &amp; maximum values for the Scrollbar</p>	<p>Scrollbar is created using (a) &amp;(b) constructors, then its parameters should be set using this method, before using it  ii) <b>int getValue()</b> –to get the current value of the Scrollbar  iii) <b>void setValue(int newval)</b> – to set the current value for the Scrollbar  iv) <b>int getMinimum()</b>  v) <b>int getMaximum()</b> – to get the min and max values of the Scrollbar  vi) <b>setUnitIncrement(int incr)</b> – each time the arrow keys in the Scrollbar are pressed, 1 is incremented or decremented from the Scrollbar, by default. This default setting can be changed using this method  vii) <b>setBlockIncrement(int incr)</b> – by default, the page up and down increments are 10. This default setting can be changed by this method</p>
7)TextField	<p>a) <b>TextField()</b>  b) <b>TextField(int numchars)</b>-creates a TextField which is numchars wide  c) <b>TextField(String str)</b> – initializes the TextField with the given String  d) <b>TextField(String str, int numchars)</b>-initializes it with the str and also sets its width</p>	<p>i) <b>String getText()</b>- to retrieve the String currently contained in the TextField  ii) <b>void setText(String s)</b> – to set the text  iii) <b>void setEchoChar(char ch)</b> – specifies a single char that the Textfield will display for any character that is entered( used for typing passwords)  iv) <b>char getEchoChar()</b> –to retrieve the echo character  v) <b>void setEditable(boolean edit)</b> – if edit is true, the text in the TextField can be changed otherwise not</p>
8) TextArea	<p>a) <b>TextArea()</b>  b) <b>TextArea(int nlines,int nchars)</b>  c) <b>TextArea(String str)</b> –&gt;str-initial text  d) <b>TextArea(String str, int nlines, int nchars)</b>  e) <b>TextArea(String str, int nlines, int nchars, int sbars)</b>  nlines- specifies the height of the TextArea in lines  nchars-specifies the width of the TextArea in characters  sbars- specifies the scrollbars which the TextArea must have- can be SCROLLBARS_BOTH,</p>	<p>i) <b>String getText()</b>  ii) <b>void setText(String str)</b>  iii) <b>void append(String str)</b>- adds at the end of the current text  iv) <b>void insert(String str, int index)</b> –inserts the String at the specified index  v) <b>void replaceRange(String str, int start, int end)</b> –replaces characters from start to end-1, with the replacement text passed in str.</p>

	SCROLLBARS_NONE, SCROLLBARS_HORIZONTAL_ON LY,SCROLLBARS_VERTICAL_ON LY	
9)MenuBar A Menubar contains 1 or more Menu objects. Each Menu object contains a list of MenuItem objects.	a) <b>MenuBar()</b> -for creating MenuBar b) <b>Menu()</b> c) <b>Menu(String menuname)</b> (b) & (c) -for creating Menu objects d) <b>MenuItem()</b> e) <b>MenuItem(String itemname)</b> (d) & (e) -for creating MenuItem objects	i) <b>setMenuBar(MenuBar mb)</b> ii) <b>void setLabel(String name)</b> iii) <b>String getLabel()</b> iv) <b>void setEnabled(boolean b)</b> v) <b>boolean isEnabled()</b>

12) Program to display 2 label boxes, 2 TextFields to read the values of 'a' and 'b' and a button with 'name'.

Demonstrates the use of two important methods:

**a) setLayout(LayoutManager obj)**

-used to assign the LayoutManager. A GridLayout, FlowLayout or BorderLayout can be used to arrange the controls within the applet window

Eg., **setLayout(new CardLayout());**

If no LayoutManager Is assigned explicitly, the default LayoutManager will be used. The default layout manager is called FlowLayout. Here, objects are placed left-to-right, top-to-bottom in the order in which they are added. Thus, if multiple objects are added, the order in which they are added determines where they will appear. When the window is resized, the positions of the objects are changed to fit the window size.

**setLayout(null)** indicates that no LayoutManager will be used and the positioning of the various controls will be taken care of, within the program itself. This statement prevents the usage of default LayoutManager.

**b) setBounds(int startx, int starty, int width, int height)**

-used to specify the x,y coordinates of the starting point of the component to be displayed and the size of the component.

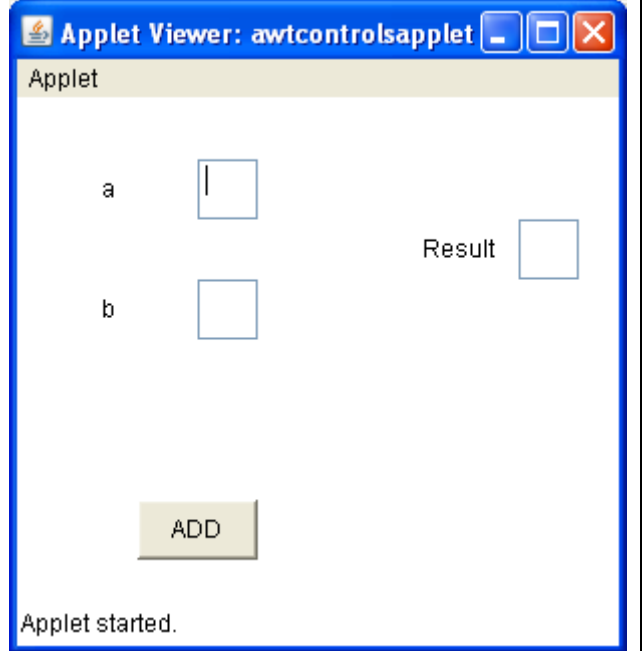
Eg., b. **setBounds(30,40,50,50);** -> will display the Button object 'b' from the starting point (30,40) and its size is 50 pixels in width and 50 pixels in height.

**Note:** The explicit positioning of the components done by using **setBounds()** will be applicable only if the default LayoutManager is also disabled using **setLayout(null)**.

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet      code="awtcontrolsapplet"
width=300 height=250>
</applet>
*/
public class awtcontrolsapplet extends
Applet
{
    TextField t1,t2,t3;
    Label l1,l2,l3; Button b1;
    public void init()
    {
        l1=new Label("a");
        l2=new Label("b");
        l3=new Label("Result");
        t1=new TextField();
        t2=new TextField();
        t3=new TextField();
        Button b=new Button("ADD");
        setLayout(null);
        l1.setBounds(40,30,30,30);
        l2.setBounds(40,90,30,30);
        t1.setBounds(90,30,30,30);
        t2.setBounds(90,90,30,30);
        b.setBounds(60,200,60,30);
        l3.setBounds(200,60,50,30);
        t3.setBounds(250,60,30,30);
        add(l1); add(l2); add(l3);
        add(t1); add(t2); add(t3);
        add(b);
    }
}

```



In this program 3 Labels, 3 TextFields and one Button are created and added to the applet window using add() method. **setLayout(null)** indicates that no LayoutManager will be used and the positioning of the various controls will be taken care of, within the program itself. This statement prevents the usage of default LayoutManager. setBounds() is used to position the controls in the desired locations.

Note: Though setBounds() is used to arrange the controls in the specified location, it will be followed only when the default LayoutManager is disabled using setLayout(null). Otherwise, the components will be arranged as per the control of the default LayoutManager

The above program displays the various controls on to the applet window. However, there will not be any response, upon clicking the Button 'ADD'. paint() method can be defined to read the 'a' and 'b' values from the TextFields and display their sum in the third TextField. But this paint() method will be invoked automatically after the applet starts its execution. So, it will not wait for us to enter the input values in the TextFields and then go for retrieving those values and adding them.

In this situation, we expect the action (addition of the two values) to take place, only after clicking the ADD Button. This can be achieved by way of Event Handling mechanism.

## Event Handling

An event in an Applet is an object that represents some action such as clicking the mouse, pressing a key on the keyboard, closing a window, pressing a button, etc.

Objects (e.g. Buttons, TextFields, mouse, etc) are said to generate or fire events. Different types of objects can fire different kinds of events. For example,

- **Button** and **TextField** objects fire **ActionEvent** objects
- a mouse can generate a **MouseEvent**
- a window can generate a **WindowEvent**

Every object that can fire an event is called an event source.

A Listener is an object that is notified when an event occurs

There are two Event handling models

i) Event-inheritance (or) hierarchical model:

If an event occurs, the event would be propagated to all the components, up the containment hierarchy until it is handled by a component. This required components to receive even the events that they do not process and it wastes time.

ii) Delegation event model:

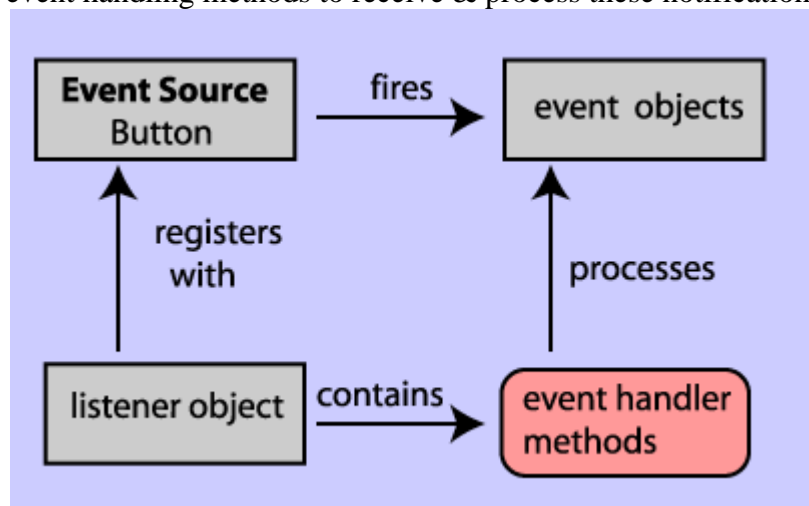
A listener simply waits until it receives an event. When a source generates an event and sends it to one or more listeners, the listener receives it, processes it and then returns.

In this, the listeners must register with a source in order to receive an event notification. Thus notifications are sent only to the listeners that want to receive them. A listener object has methods that specify what will happen when an event is sent to the listener. These methods are called event handlers. The programmers have to define these event handlers.

To perform event handling the following two steps have to be done.

i) The Listeners must register with 1 or more sources to receive notifications about specific types of events

ii) must implement event handling methods to receive & process these notifications



### How to register Listeners?

Each type of event has its own registration method.

Syntax: **public void addTypeListener(TypeListener ref)**

where type should be the name of the event and ref is the reference to the event listener

Eg., the method that registers a keyboard event listener → `addKeyListener()`

The method that registers a mouse motion listener → `addMouseMotionListener()`

When an event occurs, all registered listeners are notified & receive a copy of the event object.

Listeners are created by implementing 1 or more of the interfaces defined by the `java.awt.event.*` package. So, to use the event handling in the application, import the `java.awt.event.*` package.

Whenever an event is fired from a component, an event object is propagated to an appropriate method in the Listener object, and that method contains code to handle the event. Corresponding to every AWTEvent there is a corresponding event listener. For example, for ActionEvent we have ActionListener.

There is a method in ActionListener interface `actionPerformed()` to which the ActionEvent object is passed and this method can be defined to handle the event.

Below is the list of important Event classes and their corresponding Listener interfaces

Event Class	Important methods in Event Class	Listener interface	Methods in the interface
<b>1)ActionEvent-</b> generated when a Button is pressed, list item double-clicked, menu item selected	<b>String getActionCommand()</b> -to get the command name for the invoking ActionEvent object. The command name is equal to the label on the button	<b>ActionListener</b>	<b>void actionPerformed (ActionEvent ae)</b>
<b>2)ItemEvent-</b> generated when checkbox or list item is clicked	a) <b>Object getItem()-</b> obtains a reference to the item that generated this event  b) <b>int getStateChange()-</b> returns the state change (SELECTED or DESELECTED) for the event	<b>ItemListener</b>	<b>void itemStateChanged (ItemEvent ae) –</b> invoked when the state of an item changes
<b>3)AdjustmentEvent-</b> generated when Scrollbar manipulated	<b>int getValue()-</b> returns the amount of adjustment made in the Scrollbar	<b>AdjustmentListener</b>	<b>void adjustmentValueChanged (AdjustmentEvent ae)</b>
<b>4)TextEvent-</b> generated when value of TextField or TextArea is changed		<b>TextListener</b>	<b>void textValueChanged (TextEvent e) –</b> invoked when a change occurs in a TextArea or TextField
<b>5)KeyEvent-</b> generated when input is received from	a) <b>char getKeyChar() –</b> returns the character that was entered	<b>KeyListener</b>	<b>void keyPressed (KeyEvent ke)-</b> invoked when a key is



keyboard Constants- VK_F1, VK_F2, ...,VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT, VK_PAGE_UP, VK_PAGE_DOWN etc	b) <b>int getKeyCode()</b> – returns the Key code		pressed  <b>void keyReleased (KeyEvent ke)</b> – invoked when a key is released  <b>void keyTyped (KeyEvent ke)</b> -invoked when a character is generated by the key stroke (so, not invoked when function keys or shift,ctrl,alt are pressed
6) <b>MouseEvent</b> - generated when a mouse is clicked, pressed, released, dragged, moved and when it enters or exits a component	a) <b>int getX()</b>  b) <b>int getY()</b> -return the x-y coordinates of the mouse  c) <b>Point getPoint()</b> – to get the coordinates of the mouse as object  c) <b>int getClickCount()</b> - returns the nr of mouse clicks for this event  d) <b>int getXOnScreen()</b>  e) <b>int getYOnScreen()</b> – these return the x-y coordinates of the mouse relative to the screen rather than the component	<b>MouseListener</b>	<b>void mouseClicked( MouseEvent me)</b> -called when the mouse is pressed & released at the same point  <b>void mousePressed( MouseEvent me)</b>  <b>void mouseReleased( MouseEvent me)</b>  <b>void mouseEntered( MouseEvent me)</b> -when a mouse enters a component  <b>void mouseExited( MouseEvent me)</b> - when a mouse exits a component
		<b>MouseMotionListe ner</b>	<b>void mouseDragged( MouseEvent me)</b> – called multiple times as the mouse is dragged  <b>void mouseMoved( MouseEvent me)</b> – called multiple times as the mouse is moved
7) <b>WindowEvent</b> - when window activated, closed,	a) <b>Window getWindow()</b> - returns the window object that generated the event	<b>WindowListener</b>	i) <b>void windowActivated (WindowEvent we)</b>

deactivated, deiconified, iconified, opened or quit	<b>b)Window getOppositeState()</b> <b>c) int getOldState()</b> <b>d)int getNewState() –</b> ( c)& (d) return the previous &current state of the window		<b>ii) void windowDeactivated (WindowEvent we)</b>  <b>void windowIconified (WindowEvent we)</b>  <b>void windowDeiconified (WindowEvent we)</b>  <b>void windowOpened (WindowEvent we)</b>  <b>void windowClosed (WindowEvent we)</b>  <b>void windowClosing (WindowEvent we)</b>
---	---	--	--

Having seen the list of Event classes and their corresponding Listeners, the previous applet program for adding 2 numbers using textfields and buttons can be modified as follows.

i)first define the applet class by implementing the ActionListener interface( along with extending Applet class)

ii)next, register listener for the Button so as to receive notifications –b.addActionListener(this);

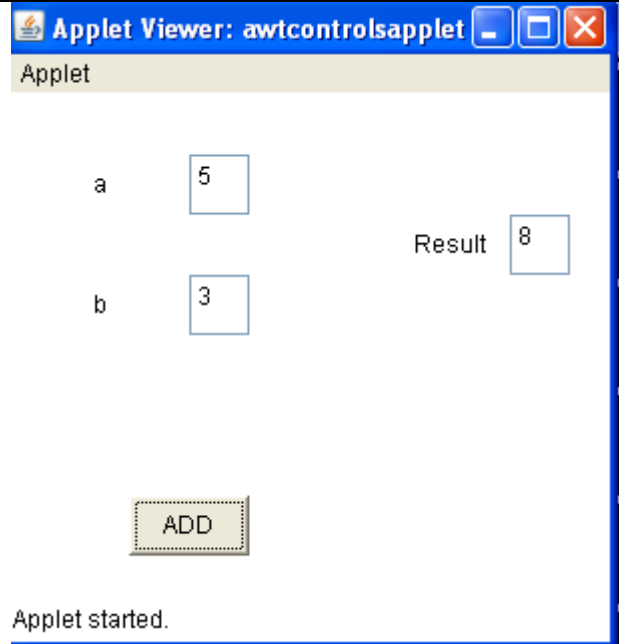
As the addActionListener() method requires a reference to the object of the class implementing the interface, 'this' can be used since it is the reference to the current object of the class

iii) now, define the actionPerformed() method of ActionListener interface . The applet class implements ActionListener and so this method should be defined in our program. Otherwise, the class will be expected to be an abstract class. Include the code for adding the 2 numbers within this method

```

13)import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="awtcontrolsapplet" width=300
height=250></applet>
*/
public class awtcontrolsapplet extends Applet
implements ActionListener
{
    TextField t1,t2,t3;
    Label l1,l2,l3; Button b1;
    public void init()
    {
        l1=new Label("a");
        l2=new Label("b");
        l3=new Label("Result");
        t1=new TextField();
        t2=new TextField();
        t3=new TextField();
        Button b=new Button("ADD");
        setLayout(null);
        l1.setBounds(40,30,30,30);
        l2.setBounds(40,90,30,30);
        t1.setBounds(90,30,30,30);
        t2.setBounds(90,90,30,30);
        b.setBounds(60,200,60,30);
        l3.setBounds(200,60,50,30);
        t3.setBounds(250,60,30,30);
        add(l1); add(l2); add(l3);
        add(t1); add(t2); add(t3);
        add(b);
b.addActionListener(this);
    }
    public void actionPerformed(
ActionEvent ae)
    {
        int a=Integer.parseInt(t1.getText());
        int b=Integer.parseInt(t2.getText());
        int c=a+b;
        t3.setText(c+"");
    }
}

```



Class `awtcontrolsapplet` extends `Applet` implements `ActionListener`.... So, now this class can define methods of the `ActionListener` to respond to `ActionEvent` that might be generated when the Button is clicked. The source ( which is the Button) registers with `ActionListener`, by the statement, `b.addActionListener(this);`

We need a reference to the `ActionListener` object as the argument for this method. So, the reference to the current object can also be passed, as this class implements `ActionListener`.

Now onwards, whenever the button is pressed, the `ActionEvent` will be generated and that will be notified to the Listener object. (Here, the listener is the applet program itself). What has to be done, upon receiving the `ActionEvent` notification, can be provided in the `actionPerformed()` method. The code inside this method will be executed whenever the Button is pressed.

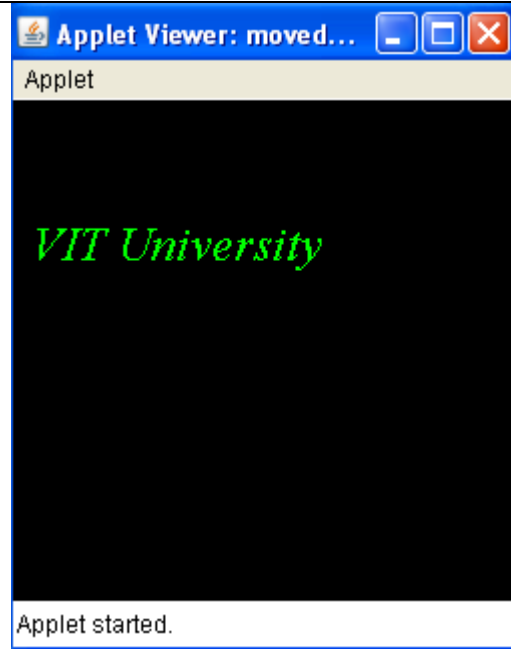
Here, the contents from the 2 textfields are retrieved using `getText()` method and as this method returns a `String`, convert those strings to integers before adding. The result should be displayed in textfield, `t3`. Hence, use `setText()` method to assign the 'c' value to it. But convert 'c' to a `String` just by appending a null

	String , before assigning it to t3. (can also be done as String s=Integer.toString(c ); t3.setText(s); )
--	---

14) Program to draw a String (“VIT UNIVERSITY”) in Applet window and move the String from top to bottom of the window continuously. Set the font as times new roman, italic style and 24 size. Set the background color as black, foreground color as green

```
import java.awt.*;
import java.applet.*;
/*
<applet code="movedownapplet" width=400
height=400>
<param name="message" value="VIT
University">
<param name="y" value=10>
</applet>
*/
public class movedownapplet extends Applet
implements Runnable
{
    String msg;
    int y; Thread t;
    Font f;
    public void init()
    {
        msg=getParameter("message");
        y=Integer.parseInt(getParameter("y"));
        setBackground(Color.black);
        setForeground(Color.green);
        f=new Font("Times New Roman",
        Font.ITALIC,24);
        t=new Thread(this);
        t.start();
    }
    public void run()
    {
        while(true)
        {
            repaint();
            try
            {
                Thread.sleep(1000);
            }
            catch(InterruptedException e)
            {

```



For this program, as continuous processing is required (no need to handle any particular event), thread object can be used. The applet program need not implement any listener. For every call to the repaint() and hence the paint() method, the y-coordinate value should be increased so as to display the message in the next line. So, increase the value of y by 10 in the paint() method where y is the instance variable of this class. Also, set the font as explained earlier.

```

}
}
}
public void paint(Graphics g)
{
g.setFont(f);
g.drawString(msg,10,y);
y+=10;
}
}

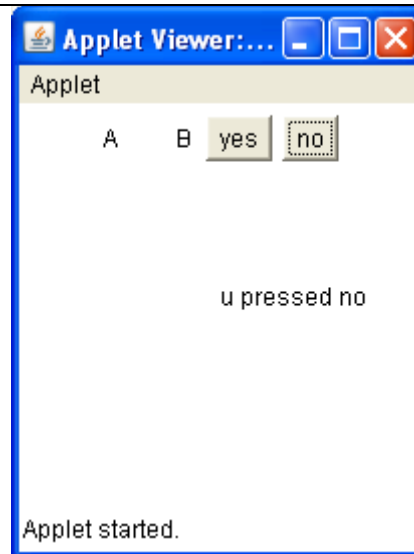
```

15) Program to display the appropriate msg when 'yes' and 'no' buttons are pressed

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="labelapplet" width=200
height=200>
</applet>
*/
public class labelapplet extends Applet
implements ActionListener
{
    Button yes;
    Button no;
    String msg="";
    public void init()
    {
        Label l1=new Label("A");
        Label l2=new Label("B",Label.RIGHT);
        add(l1);
        add(l2);
        Button yes=new Button("yes");
        Button no=new Button("no");
        add(yes);
        add(no);
        yes.addActionListener(this);
        no.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String s=ae.getActionCommand();
        if(s.equals("yes"))
            msg="u pressed yes";
        else if(s.equals("no"))
            msg="u pressed no";
    }
}

```



The class should implement ActionListener interface. Include addActionListener(this) for the 2 buttons to receive the notifications. Define the actionPerformed() method. Use getActionCommand() method to get the command name of the invoking object. This is needed to identify which button was pressed, when multiple buttons are used. This method returns the label on the button('yes' and 'no'). based on the command, the msg is set and displayed.

```

repaint();
}
public void paint(Graphics g)
{
    g.drawString(msg,100,100);
}
}

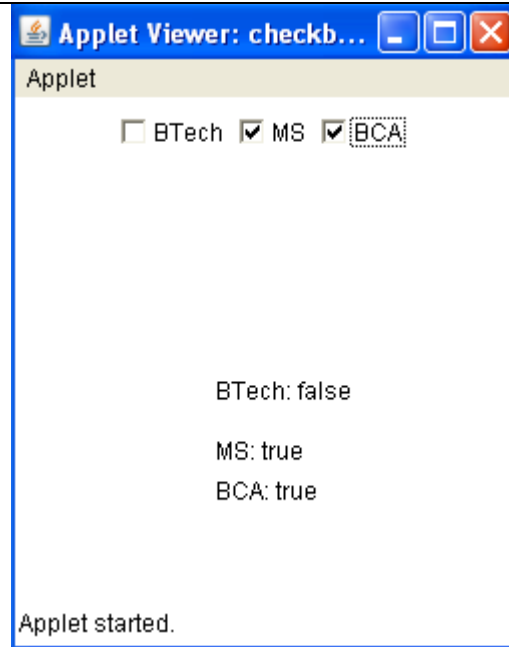
```

16) Program to use checkbox

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="checkboxapplet" width=250
height=250>
</applet>
*/
public class checkboxapplet extends Applet
implements ItemListener
{
    Checkbox BTech,MS,BCA;
    String msg="";
    public void init()
    {
        BTech=new Checkbox("BTech");
        MS=new Checkbox("MS",true,null);
        // constructor
        BCA=new Checkbox("BCA");
        add(BTech);
        add(MS);
        add(BCA);
        BTech.addItemListener(this);
        MS.addItemListener(this);
        BCA.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent ie)
    {
        repaint();
    }
    public void paint(Graphics g)
    {
        msg=BTech.getLabel()+" : "+
        BTech.getState();
        g.drawString(msg,100,150);
        msg="";
        msg=MS.getLabel()+" : "+MS.getState();
        g.drawString(msg,100,180);
    }
}

```



Class should implement ItemListener. Include addItemListener(this) for all the 3 checkboxes. Define the method itemStateChanged(). The name of the checkbox and whether it is checked or not should be displayed. So, use getLabel() and getState() methods of the Checkbox class

```

    msg="";
    msg=BCA.getLabel()+" : "
+BCA.getState();
    g.drawString(msg,100,200);
}
}

```

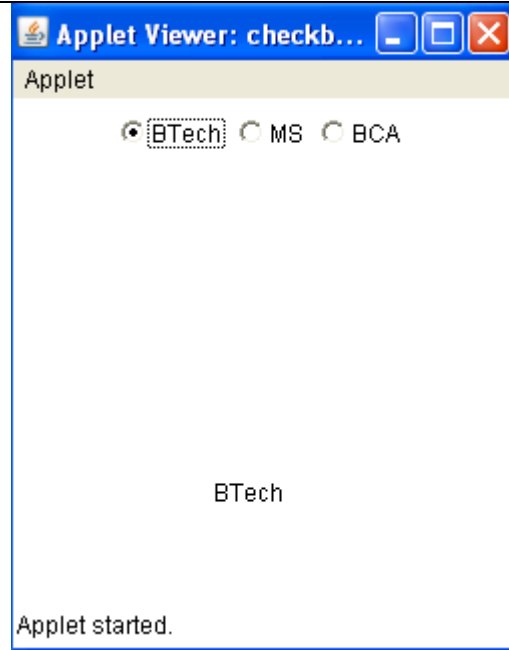
#### 17) Program to use CheckboxGroup:

Within a group, only one checkbox can be selected at a time

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet      code="checkboxgroupapplet"
width=250 height=250>
</applet>
*/
public class checkboxgroupapplet extends
Applet implements ItemListener
{
    CheckboxGroup cg=new
CheckboxGroup();
    Checkbox BTech,MS,BCA;
    String msg="";
    public void init()
    {
        BTech=new Checkbox("BTech",cg,false);
        // 4th type of constructor
MS=new Checkbox("MS",cg,false);
BCA=new Checkbox("BCA",cg,false);
        add(BTech);
        add(MS);
        add(BCA);
        BTech.addItemListener(this);
        MS.addItemListener(this);
        BCA.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent
ie)
    {
        repaint();
    }
    public void paint(Graphics g)
    {
        /* if(BTech.getState())
        {

```



First create an object for CheckboxGroup class.

Then create Checkbox objects using the 5<sup>th</sup> form of the constructor which includes the reference to the CheckboxGsrup to which the Checkboxes should belong.

Then, as in the commented block, every individual checkbox's state can be verified to know which one has been selected or `getSelectedCheckbox()` method of `CheckboxGroup` class can be used to identify which checkbox has been selected and thus its state can be displayed.

```

        msg=BTech.getLabel()+" : "
+BTech.getState();
        g.drawString(msg,100,120);
    }
    else if(MS.getState())
    {
        msg=MS.getLabel()+" : "
+MS.getState();
        g.drawString(msg,100,160);
    }
    else if(BCA.getState())
    {
        msg=BCA.getLabel()+" : "
+BCA.getState();
        g.drawString(msg,100,200);
    }
    */
    msg=cg.getSelectedCheckbox().getLabel();
    g.drawString(msg,100,200);
    //cg.setSelectedCheckbox(MS);
    }
}

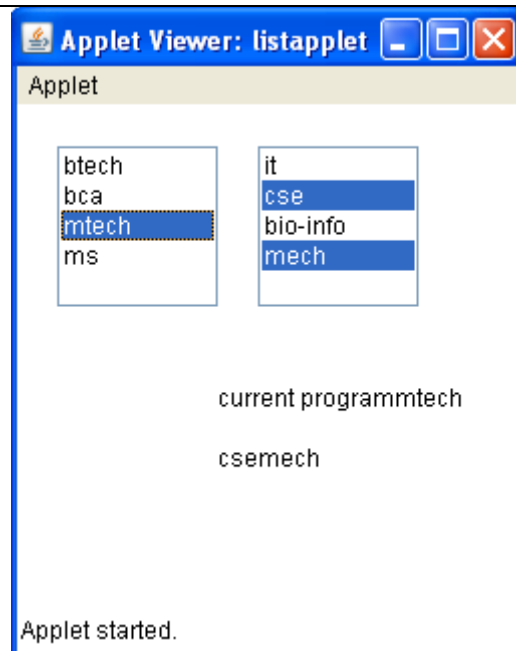
```

18) Program to use List:

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet      code="listapplet"      width=250
height=250>
</applet>
*/
public class listapplet extends Applet
implements ActionListener
{
    List program, branch;
    String msg="";
    public void init()
    {
        program=new List(3,false);
        branch=new List(3,true);
        program.add("btech");
        program.add("bca");
        program.add("mtech");
        program.add("ms");
        branch.add("it");
        branch.add("cse");

```



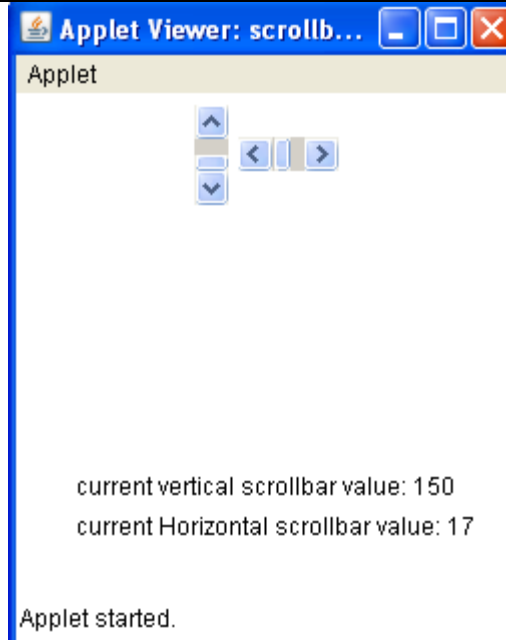
Add all the options(btech,ms,bca) to the list named 'program' by  
 Program.add("btech");  
 and then add this list to the applet by  
 add(program); Similarly do for branch also



<pre> branch.add("bio-info"); branch.add("mech"); branch.select(1); setLayout(null); program.setBounds(20,20,80,80); branch.setBounds(120,20,80,80); add(program); add(branch); branch.addActionListener(this); program.addActionListener(this); } public void actionPerformed(ActionEvent ae) {     repaint(); } public void paint(Graphics g) {     msg="current program";     String s1=program.getSelectedItem();     msg+=s1;     g.drawString(msg,100,150);     msg="";     int[] ind;     ind=branch.getSelectedIndexes();     for(int i=0;i&lt;ind.length;i++)         msg+=branch.getItem(ind[i]);     g.drawString(msg,100,180); } } </pre>	<p>But branch is created by enabling the multiselect option to true. Therefore, multiple options can be selected from the branch. So, <code>getSelectedIndexes()</code> can be used to get the indexes of all the selected options in the list and a 'for' loop is run to display the item in the selected index of the list.</p>
---	---

19) Program to display the value chosen from the Scrollbar

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="scrollbarapplet" width=250
height=250>
</applet>
*/
public class scrollbarapplet extends Applet
implements AdjustmentListener
{
    String msg="";
    Scrollbar vertsb,horisb;
    public void init()
    {
        vertsb=new
        Scrollbar(Scrollbar.VERTICAL,
        0,50,0,200);
        vertsb.setUnitIncrement(2);
        vertsb.setBlockIncrement(5);
        add(verts);
        verts.addAdjustmentListener(this);
        horisb=new
        Scrollbar(Scrollbar.HORIZONTAL,
        0,1,0,100);
        add(horisb);
        horisb.addAdjustmentListener(this);
    }
    public void adjustmentValueChanged
(AdjustmentEvent ae)
    {
        repaint();
    }
    public void paint(Graphics g)
    {
        Integer val=verts.getValue();
        msg="current vertical scrollbar value: "
+val.toString();
        g.drawString(msg,30,200);
        val=horisb.getValue();
        msg="current Horizontal scrollbar
value: "+val.toString();
        g.drawString(msg,30,220);
    }
}
```

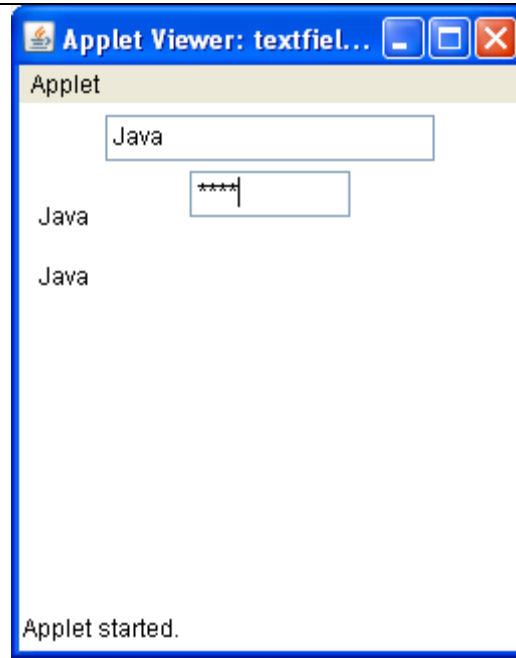


For the vertical scrollbar, the visible amount is set to 50. So, the max value for the scroll bar, is displayed on the screen as 150, even when dragged till the end. The default values for UnitIncrement and BlockIncrement values have also been changed in the program.

The values denoted by the scroll bars positions are got by using the method `getValue()` and displayed.

20) Program to display the contents of the textfield

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="textfieldapplet" width=250
height=250>
</applet>
*/
public class textfieldapplet extends Applet
implements TextListener
{
    TextField username,passwd;
    public void init()
    {
        username=new TextField(20);
        passwd=new TextField(8);
        passwd.setEchoChar('*');
        add(username);
        add(passwd);
        username.addTextListener(this);
        passwd.addTextListener(this);
    }
    public void textValueChanged(
TextEvent ae)
    {
        repaint();
    }
    public void paint(Graphics g)
    {
        g.drawString(username.getText(),10,60);
        g.drawString(passwd.getText(),10,90);
    }
}
```

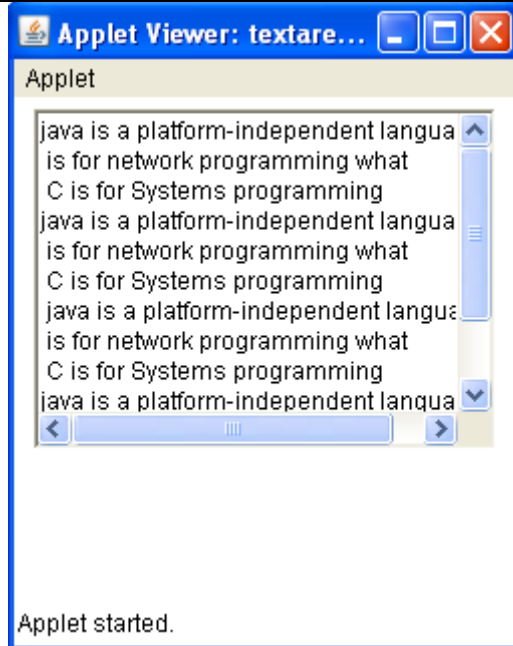


If the TextListener interface is implemented, the action will be done as and when the changes are being done in the textfields. Else, a button can also be created and on clicking the button, the contents of the textfields can be made to be displayed

21) Program to work with TextArea:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="textareapplet" width=250
height=250>
</applet>
*/
public class textareapplet extends Applet
{
    TextArea ta;
    public void init()
    {
        String val="java is a platform-independent
language\n"+
        " is for network programming what
\n"+
        " C is for Systems programming\n"+
        "java is a platform-independent
language\n"+
        " is for network programming what
\n"+
        " C is for Systems programming\n"+
        " java is a platform-independent
language\n"+
        " is for network programming what
\n"+
        " C is for Systems programming\n" +
        "java is a platform-independent
language\n"+
        " is for network programming what
\n"+
        " C is for Systems programming\n";

        ta=new TextArea(val,10,30);
        add(ta);
    }
}
```



## Frames

The two most common windows are those derived from Panel class which is used by applets and those derived from Frame class.

The window provided by Panel class does not contain a menubar, titlebar, borders and resizing corners.

Thus when an applet is executed in a browser, it does not show any border or title, as applets use window provided by the Panel class.

The Frame class inherits Window class. Generally Window objects cannot be created and hence objects for its subclass Frame, are created.

The window provided by the Frame contains menubar, title bar, borders and resizing controls.

**Also, Frame class can be used to create a window in a stand alone application.**

Constructor:

i) **Frame()**

ii) **Frame(String title)**

Method of Frame class, to make the frame visible:

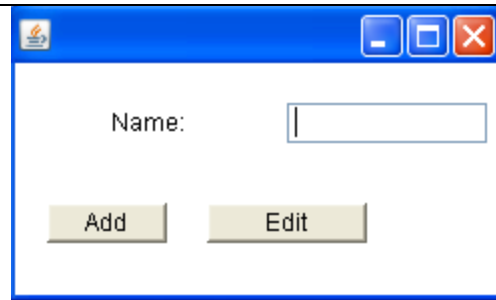
**setVisible(true)**

**setSize(width,height)** –to set the size for the frame window

Unless this method is invoked, a frame will not be visible on the screen.

22) Program to create a frame window for a standalone application

```
import java.awt.*;
class frameapplication
{
    public static void main(String[] args)
    {
        Frame f=new Frame();
        Label lab=new Label("Name: ");
        TextField text=new TextField(20);
        Button b1=new Button("Add");
        Button b2=new Button("Edit");
        f.setLayout(null);
        lab.setBounds(50,50,80,20);
        text.setBounds(140,50,100,20);
        b1.setBounds(20,100,60,20);
        b2.setBounds(100,100,80,20);
        f.add(lab);
        f.add(text);
        f.add(b1);
        f.add(b2);
        f.setVisible(true);
        f.setSize(250,150);
    }
}
```



This is not an applet program. This is a standalone application which uses main() method. A window which houses a label, textbox and 2 buttons can be created using Frame class.

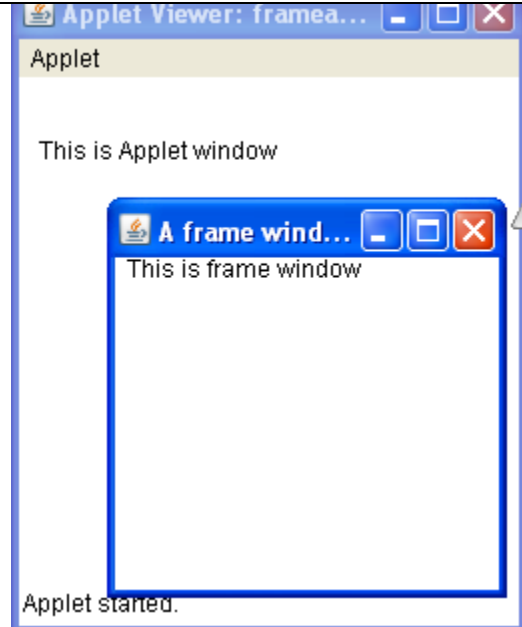
Call f.setVisible(true) in order to display the frame.

A Frame class can also be used to create a window in applets. User-defined frame class can also be created by extending the predefined Frame class. Eg., class myframe extends Frame

As Frame is a subclass of Component class, it also has paint() method. This method can be defined to display the desired contents.

23) Program to use user-defined frame in applet

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet    code="frameapplet"    height=250
width=250>
</applet>
*/
public class frameapplet extends Applet
{
    sampleframe f;
    public void init()
    {
        f=new sampleframe("A frame window");
        f.setSize(200,200);
        f.setVisible(true);
    }
    public void start()
    {
        f.setVisible(true);
    }
    public void stop()
    {
        f.setVisible(false);
    }
    public void paint(Graphics g)
    {
        g.drawString("This is Applet window",
        10,40);
    }
}
class sampleframe extends Frame
{
    sampleframe(String title)
    {
        super(title);
    }
    public void paint(Graphics g)
    {
        g.drawString("This is frame window",
        10,40);
    }
}
```



In the paint() method of sampleframe class , display the message “this is frame window”. When an object for this sampleframe class is created , say f, and if f.setVisible(true) is invoked, the new frame will be visible

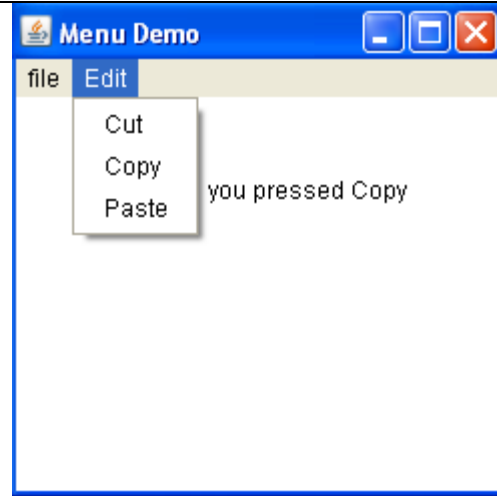
23) Program to use MenuBar  
MenuBars are not subclasses of Component. So, create a frame to host it.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet      code=menubarapplet      width=250
height=250>
</applet>
*/
class menuframe extends Frame implements
ActionListener
{
    String msg;
    menuframe(String title)
    {
        super(title);
        MenuBar mb=new MenuBar();
        setMenuBar(mb);
        Menu file=new Menu("file");
        MenuItem m1,m2,m3;
        file.add(m1=new MenuItem("New"));
        file.add(m2=new MenuItem("Open"));
        file.add(m3=new MenuItem("save"));
        mb.add(file);

        Menu edit=new Menu("Edit");
        MenuItem mi1,mi2,mi3;
        edit.add(mi1=new MenuItem("Cut"));
        edit.add(mi2=new MenuItem("Copy"));
        edit.add(mi3=new MenuItem("Paste"));
        mb.add(edit);

        m1.addActionListener(this);
        m2.addActionListener(this);
        m3.addActionListener(this);

        mi1.addActionListener(this);
        mi2.addActionListener(this);
        mi3.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae)
    {
        msg="you pressed ";
        String s=ae.getActionCommand();
```



First a frame named menuframe is created by extending the Frame class. All the menu items are added to this frame and ActionListeners are registered for these.

In the applet class, the only thing to be done is to create an object for this menuframe class and make it visible.

In the menuframe class, initially create MenuBar. Then create 2 Menu objects named file & edit.

Create 3 MenuItem objects m1, m2 & m3 and add them to file Menu.

Similarly create 2 MenuItem objects m4 & m5 and add them to edit menu.

Register ActionListeners for these 5 MenuItem objects.

Note: Events are generated only when any of the MenuItems is clicked and not when Menu object is chosen. So, add ActionListeners for MenuItems alone

```

        if(s.equals("New"))
msg+="New";
    else if(s.equals("Open"))
        msg+="Open";
    else if(s.equals("save"))
msg+="save";
    else if(s.equals("Cut"))
msg+="Cut";
    if(s.equals("Copy"))
msg+="Copy";
    if(s.equals("Paste"))
msg+="Paste";
    repaint();
}
public void paint(Graphics g)
{
g.drawString(msg, 100,100);
}
}
public class menubarapplet extends Applet
{
    Frame f;
    public void init()
    {
        f=new menuframe("Menu Demo");
        f.setSize(250,250);
        f.setVisible(true);
    }
    public void start()
    {
        f.setVisible(true);
    }
    public void stop()
    {
        f.setVisible(false);
    }
}

```

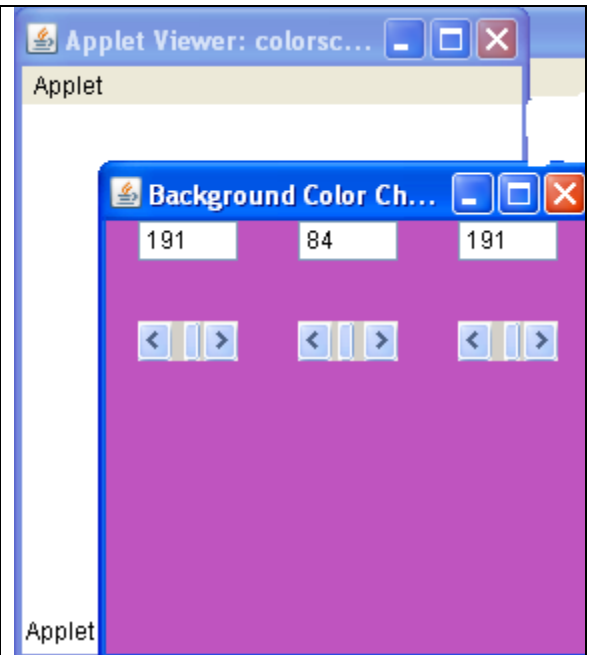
24) Program to create a frame with 3 text boxes called Red, Green and Blue. Each of these text boxes has a scroll bar with integers ranging from 0 to 255. We have to input the red, green and blue values. There must be a button called Change Color. When we click this button the background color must change to a new color created with these red, green and blue values



```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="colorscroll" width=250 height=250>
</applet>
*/
public class colorscroll extends Applet implements
AdjustmentListener
{
    Frame f;
    TextField t1,t2,t3;
    Scrollbar hsb1,hsb2,hsb3;
    public void init()
    {
        f=new Frame("Background Color Change");
        t1=new TextField(8);
        t2=new TextField(8);
        t3=new TextField(8);
        hsb1=new
Scrollbar(Scrollbar.HORIZONTAL,0,1,0,255);
        // actual max=max-visibleamount(which is 1
        here)
        hsb2=new Scrollbar(Scrollbar.HORIZONTAL,
        0,1,0,255);
        hsb3=new Scrollbar(Scrollbar.HORIZONTAL,
        0,1,0,255);
        t1.setBounds(20,30,50,20);
        t2.setBounds(100,30,50,20);
        t3.setBounds(180,30,50,20);
        hsb1.setBounds(20,80,50,20);
        hsb2.setBounds(100,80,50,20);
        hsb3.setBounds(180,80,50,20);
        f.add(t3);
        f.add(t1);
        f.add(t2);
        f.add(hsb1);
        f.add(hsb2);
        f.add(hsb3);
        f.setLayout(null);
        f.setSize(250,250);
        f.setVisible(true);
        hsb1.addAdjustmentListener(this);
        hsb2.addAdjustmentListener(this);
        hsb3.addAdjustmentListener(this);
    }
    public void adjustmentValueChanged

```



Frame is created as usual. The values in the 3 Horizontal scroll bars are used to create a user defined color and set that color as background color for the frame

```

(AdjustmentEvent ae)
{
t1.setText(Integer.toString(hsb1.getValue()));
t2.setText(Integer.toString(hsb2.getValue()));
    t3.setText(Integer.toString(hsb3.getValue()));
f.setBackground(new
Color(hsb1.getValue(),hsb2.getValue(),hsb3.getValue()
));
}
}

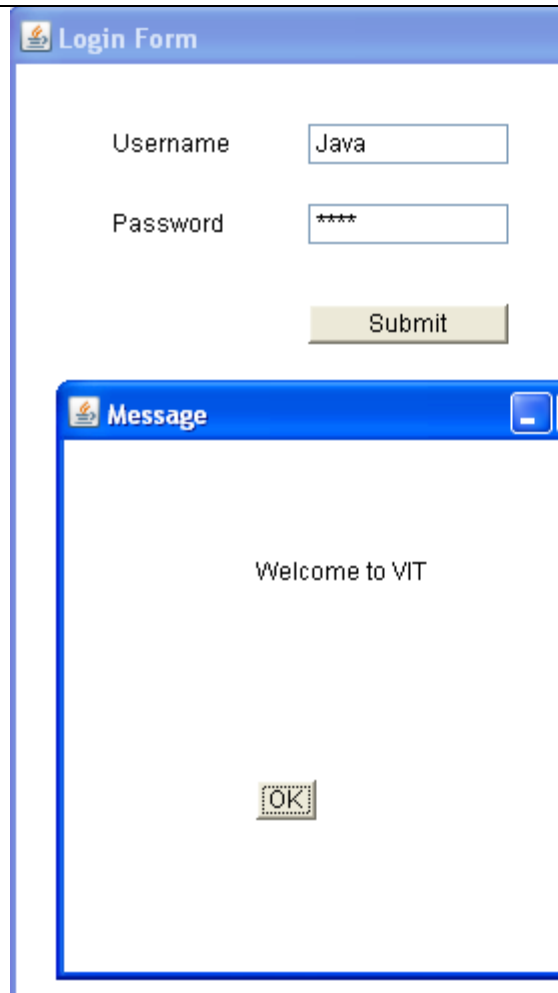
```

25) Program to create a login form with username, password field and submit button using Frame and display some information when the user hit (use event handling mechanism) the submit button. (If the username and password is correct then display “you are authorized user“ otherwise display “you are unauthorized user” –Use Frame Class).

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="authenticationapplet" height=400
width=400>
</applet>
*/
public class authenticationapplet extends Applet
implements ActionListener
{
    Frame f2,f3;
    Frame f;
    TextField t1,t2;
    Button b1;
    public void init()
    {
f=new Frame("Login Form");
    f2=new frame("Message");
    f3=new frame("Message");
    Label l1=new Label("Username");
    Label l2=new Label("Password");
    l1.setBounds(50,60,80,20);
    l2.setBounds(50,100,80,20);
    f.add(l1);
    f.add(l2);
    t1=new TextField();
    t2=new TextField();
    t2.setEchoChar('*');
    t1.setBounds(150,60,100,20);
    t2.setBounds(150,100,100,20);

```



Here, the predefined Frame class has

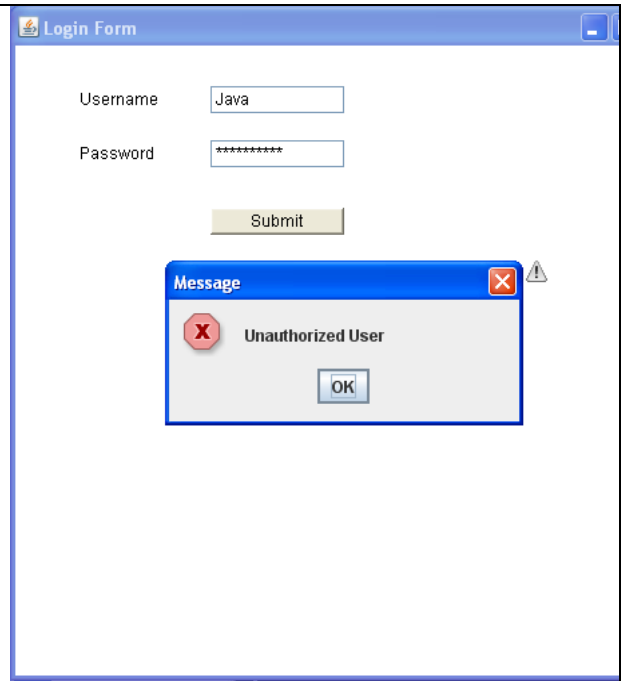
<pre> b1=new Button("Submit");     b1.setBounds(150,150,100,20); f.add(t1);     f.add(t2);     f.add(b1);     b1.addActionListener(this);     f.setLayout(null);      f.setSize(500,500);     f.setVisible(true);     }     public void actionPerformed(ActionEvent ae)     { String un=t1.getText(); String pw=t2.getText(); if((un.equals("Java")) &amp;&amp; (pw.equals("Java")))     {         Button b2=new Button("OK"); b2.setBounds(100,200,30,20); f2.add(b2);         f2.setLayout(null);         f2.setSize(300,300);         f2.setVisible(true);         f2.msg="Welcome to VIT";         f2.repaint();     }     else     { Button b3=new Button("OK");     b3.setBounds(100,200,30,20);     f3.add(b3);     f3.setLayout(null);     f3.setSize(300,300);     f3.setVisible(true);     f3.msg="Unauthorized User"; f3.repaint();     }     }     } class frame extends Frame {     String msg; frame(String title) {     super(title); }     public void paint(Graphics g) </pre>	<p>been used to create the login window and the user defined frame is used to create windows for displaying the login info'</p>
---	---

```
{
g.drawString(msg,100,100);
}
}
```

## 25)b) Same program using Swing

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet    code="authentication2"    height=400
width=400>
</applet>
*/
public class authentication2 extends Applet
implements ActionListener
{
    Frame f;
    TextField t1,t2;
    Button b1;
    public void init()
    {
        f=new Frame("Login Form");
        Label l1=new Label("Username");
        Label l2=new Label("Password");
        l1.setBounds(50,60,80,20);
        l2.setBounds(50,100,80,20);
        f.add(l1);
        f.add(l2);
        t1=new TextField();
        t2=new TextField();
        t2.setEchoChar('*');
        t1.setBounds(150,60,100,20);
        t2.setBounds(150,100,100,20);
        b1=new Button("Submit");
        b1.setBounds(150,150,100,20);
        f.add(t1);
        f.add(t2);
        f.add(b1);
        b1.addActionListener(this);
        f.setLayout(null);

        f.setSize(500,500);
        f.setVisible(true);
    }
}
```



Import javax.swing package  
The showMessageDialog() method of JOptionPane class creates a modal dialog box. It takes 4 arguments

- the parent frame (in this program ,f)
  - message to be displayed in the dialog box
  - title for the dialog box
  - constant of JOptionPane class to denote the type of message. Accordingly the icon will be displayed.
- INFORMATION\_MESSAGE → 'i' symbol  
PLAIN\_MESSAGE → no icon  
ERROR\_MESSAGE → 'X' symbol  
WARNING\_MESSAGE → '!' symbol

JOptionPane.showMessageDialog(f,"Welcome to VIT", "Message", JOptionPane.INFORMATION\_MESSAGE);

```

public void actionPerformed(ActionEvent ae)
{
    String un=t1.getText();
    String pw=t2.getText();
    if((un.equals("Java")) && (pw.equals("Java")))
    {
        JOptionPane.showMessageDialog(f,
            "Welcome to VIT", "Message",
            JOptionPane.INFORMATION_MESSAGE);
    }
    else
    {
        JOptionPane.showMessageDialog(f,
            "Unauthorized User", "Message",
            JOptionPane.ERROR_MESSAGE);
    }
}
}

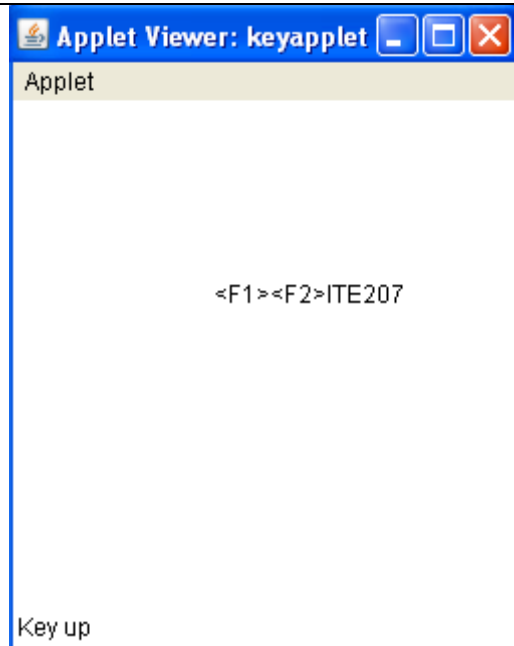
```

26) Program to work with KeyListener

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code=keyapplet width=250 height=250>
</applet>
*/
public class keyapplet extends Applet implements
KeyListener
{
    String msg="";
    int x=100,y=100;
    public void init()
    {
        addKeyListener(this);
    }
    public void keyPressed(KeyEvent ke)
    {
        showStatus("Key down");
        int key=ke.getKeyCode();
        switch(key)
        {
            case KeyEvent.VK_F1:
                msg+="<F1>";
                break;
            case KeyEvent.VK_F2:
                msg+="<F2>";

```



The applet class now implements KeyListener. So, all the 3 methods in KeyListener interface have to be defined in the program. Otherwise the applet class should be an abstract class.

```

break;
    case KeyEvent.VK_F3:
        msg+="<F3>";
break;
    case KeyEvent.VK_PAGE_DOWN:
        msg+="<PgDn>";
break;
    case KeyEvent.VK_RIGHT:
        msg+="<Right Arrow>";
break;
}
repaint();
}
public void keyReleased(KeyEvent ke)
{
    showStatus("Key up");
}
public void keyTyped(KeyEvent ke)
{
    msg +=ke.getKeyChar();
    repaint();
}
    public void paint(Graphics g)
    {
        g.drawString(msg,x,y);
    }
}

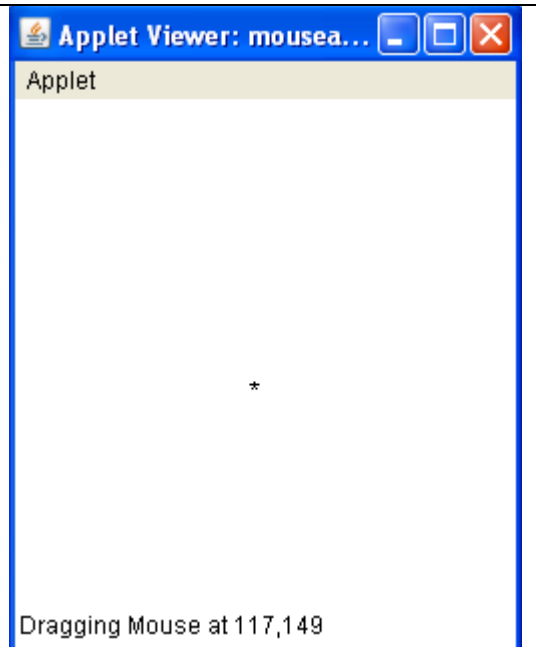
```

27) Program to implement MouseListener and MouseMotionListener

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code=mouseapplet width=250 height=250>
</applet>
*/
public class mouseapplet extends Applet implements
MouseListener, MouseMotionListener
{
    int mx=0,my=0;
    String msg;
    public void init()
    {
        addMouseListener(this);
        addMouseMotionListener(this);
    }
}

```



<pre> } public void mouseClicked(MouseEvent me) { mx=me.getX(); my=me.getY();     msg=" Mouse clicked"; repaint(); }     public void mouseEntered(MouseEvent me)     { mx=0; my=10;     msg=" Mouse Entered";     repaint(); }     public void mouseExited(MouseEvent me)     { mx=0; my=10; msg=" Mouse Exited"; repaint(); }     public void mousePressed(MouseEvent me)     { mx=me.getX(); my=me.getY();     msg=" Mouse Down";     repaint(); }     public void mouseReleased(MouseEvent me)     { mx=me.getX(); my=me.getY();     msg=" Mouse up";     repaint(); }     public void mouseDragged(MouseEvent me)     { mx=me.getX();     my=me.getY();     msg="*"; showStatus("Dragging Mouse at " +mx+", "+my);     repaint(); }     public void mouseMoved(MouseEvent me)     { showStatus("Moving mouse at "+me.getX() +", "+me.getY()); }     public void paint(Graphics g) { </pre>	<p>Define all the 5 methods of the <code>MouseListener</code> and 2 methods of the <code>MouseMotionListener</code> interfaces.</p> <p>Even if there is no need to process <code>mouseEntered</code> event, still atleast empty implementation of those methods must be provided. <code>getX()</code> and <code>getY()</code> methods of the <code>MouseEvent</code> class can be used to retrieve the x-y coordinates of the current mouse position</p>
--	--

```

g.drawString(msg,mx,my);
}
}

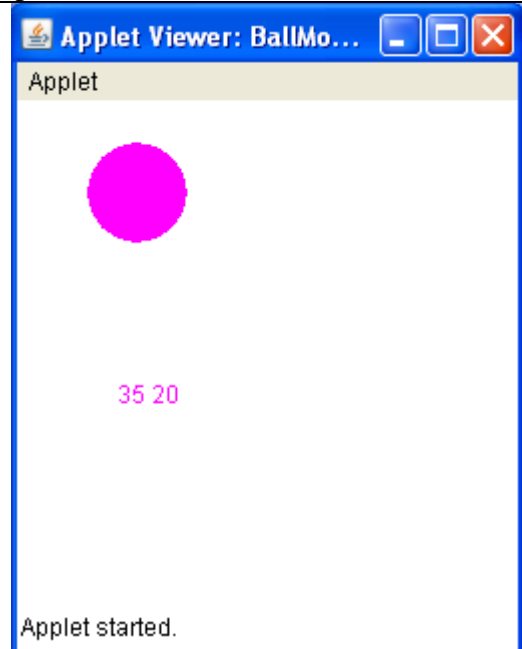
```

28) Program to move the ball as per the arrow key that is pressed

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet    code="BallMoveListener"    height=250
width=250>
</applet>
*/
public class BallMoveListener extends Applet
implements KeyListener
{
    int x,y;
    public void init()
    {
        x=y=100;
        addKeyListener(this);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.magenta);
        g.drawString(x+" "+y,50,150);
        g.fillOval(x,y,50,50);
    }
    public void keyPressed(KeyEvent ke)
    {
        int key=ke.getKeyCode();
        switch(key)
        {
            case KeyEvent.VK_UP: y-=5;break;
            case KeyEvent.VK_DOWN: y+=5;break;
            case KeyEvent.VK_LEFT: x-=5;break;
            case KeyEvent.VK_RIGHT: x+=5;break;
        }
        repaint();
    }
    public void keyReleased(KeyEvent ke) {}
    public void keyTyped(KeyEvent ke) {}
}

```



As we need to process only keyPressed() method, other methods are defined as empty methods

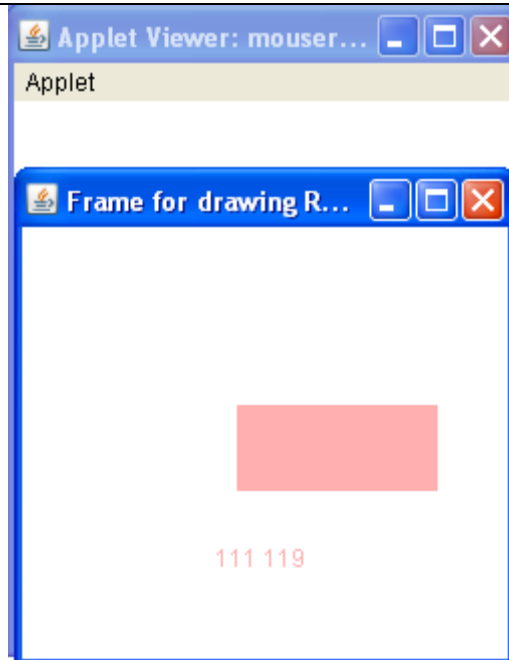


## 29) Program to draw the rectangle as per the mouse is moved

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="mouserectanglelistener" height=250
width=250>
</applet>
*/
public class mouserectanglelistener extends Applet
    // implements MouseListener
{
    newframe f;
    public void init()
    {
        f=new newframe("Frame for drawing
Rectangle");
        f.setSize(250,250);
        f.setVisible(true);
        f.repaint();
    }
}
class newframe extends Frame implements
MouseListener
{
    int x1,y1,x2,y2,height,width;
    newframe(String title)
    {
        super(title);
        x1=y1=x2=y2=height=width=0;
        addMouseListener(this);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.pink);
        g.drawString(x1+" "+y1,100,200);
        g.fillRect(x1,y1,width,height);
    }
    public void mousePressed(MouseEvent me)
    {
        x1=me.getX();
        y1=me.getY();
    }
    public void mouseReleased(MouseEvent me)
    {
        x2=me.getX();
        y2=me.getY();
    }
}

```



In the applet class only frame is made visible.

Newframe class implements MouseListener. So, define all the methods (at least as empty methods)

The point where the mouse is pressed is taken as the starting point for the rectangle and the point where the mouse is released is taken as the ending point for the rectangle and thus height and width are calculated from these and the rectangle drawn

<pre> width=x2-x1; height=y2-y1; repaint(); } public void mouseClicked(MouseEvent me) {} public void mouseExited(MouseEvent me) {} public void mouseEntered(MouseEvent me) {} } </pre>	
--	--

## Adapter classes

**Note:** Learn this only if you understand otherwise implement the Listener for all the programs.

Adapter classes provide default implementation of all methods in an event listener interface.

Useful when only some of the events handled by a particular event listener interface have to be processed

This avoids cluttering the program with several unwanted empty methods

-For this, extend one of the adapter classes & create a new class which will act as event listener & implement only those events in which you are interested

Listener interfaces and their equivalent Adapter classes

<b>KeyListener –</b>	<b>KeyAdapter</b>
<b>MouseListener-</b>	<b>MouseAdapter</b>
<b>MouseMotionListener-</b>	<b>MouseMotionAdapter</b>
<b>WindowListener-</b>	<b>WindowAdapter</b>

Ex.,

```

30) import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="adapterapplet" height=400 width=400>
</applet>
*/
public class adapterapplet extends Applet // this class once again cannot extend MouseAdapter
{
    //so create a separate class later by extending MouseAdapter
    public String msg="";
    public void init()
    {
        addMouseListener(new MyMouseAdapter(this)); // a ref to the object of the class which
        extends
        addKeyListener(new MyKeyAdapter(this)); // MouseAdapter should be passed for this
    }
    public void paint(Graphics g)
    {
        g.drawString(msg,10,40);
    }
}

```

```

class MyMouseAdapter extends MouseAdapter
{
    adapterapplet aa;
    MyMouseAdapter(adapterapplet aa)
    {
        this.aa=aa;                // in this class the status bar of the applet can only be
                                   // accessed through the object for the applet class
    }
    public void mouseClicked(MouseEvent me)
    {
        aa.showStatus("Mouse clicked");
    }
}

class MyKeyAdapter extends KeyAdapter
{
    adapterapplet aa;
    MyKeyAdapter(adapterapplet aa)
    {
        this.aa=aa;
    }
    public void keyTyped(KeyEvent me)
    {
        aa.msg+=me.getKeyChar();
        aa.showStatus("Key typed");
        aa.repaint(); // if repaint() is not called, paint() will not be called
    }
}

```

Programs 28 and 29 can be done using Adapter classes

<pre> 28) import java.awt.*; import java.applet.*; import java.awt.event.*; /* &lt;applet      code="BallMoveAsPerKey" height=400 width=400&gt; &lt;/applet&gt; */ public class BallMoveAsPerKey extends Applet {     int x,y;     public void init()     {         x=y=100;         addKeyListener(new MyKeyAdapter(this));     }     public void paint(Graphics g)     { </pre>	<pre> 29) import java.awt.*; import java.applet.*; import java.awt.event.*; /* &lt;applet  code="MouseRectangle"  height=400 width=400&gt; &lt;/applet&gt; */ public class MouseRectangle extends Applet {     newframe f;     public void init()     {         f=new newframe("Frame for drawing Rectangle");         f.setSize(700,700);         f.addMouseListener(new MyMouseAdapter(this));         f.setVisible(true); </pre>
---	---

<pre> g.setColor(Color.magenta); g.drawString(x+" "+y,100,200); g.fillOval(x,y,50,50); } } class MyKeyAdapter extends KeyAdapter {     BallMoveAsPerKey bk;     MyKeyAdapter(BallMoveAsPerKey bk)     {         this.bk=bk;         bk.showStatus("now in constructor");     }     public void keyPressed(KeyEvent ke)     {         int key=ke.getKeyCode();         bk.showStatus("Now in method");         switch(key)         {             case KeyEvent.VK_UP: bk.y-=5;                 break;             case KeyEvent.VK_DOWN:bk.y+=5;                 break;             case KeyEvent.VK_LEFT:bk.x-=5;                 break;             case KeyEvent.VK_RIGHT: bk.x+=5;                 break;         }         bk.repaint();     } } </pre>	<pre> f.repaint(); } } class newframe extends Frame {     int x1,y1,x2,y2,height,width;     newframe(String title)     {         super(title);         x1=y1=x2=y2=height=width=0;         addWindowListener(new         MyWindowAdapter(this));     }     public void paint(Graphics g)     {         g.setColor(Color.pink);         g.drawString(x1+" "+y1,100,200);         g.fillRect(x1,y1,width,height);     } } class MyWindowAdapter extends WindowAdapter {     newframe f;     MyWindowAdapter(newframe nf)     {         f=nf;     }     public void windowClosing(WindowEvent we)     {         f.setVisible(false);     } } class MyMouseAdapter extends MouseAdapter {     MouseRectangle mr;     MyMouseAdapter(MouseRectangle mr)     {         this.mr=mr;     }     public void mousePressed(MouseEvent me)     {         mr.f.x1=me.getX();         mr.f.y1=me.getY();     }     public void mouseReleased(MouseEvent me) </pre>
---	--

	<pre> {     mr.f.x2=me.getX();     mr.f.y2=me.getY();     mr.f.width=mr.f.x2-mr.f.x1;     mr.f.height=mr.f.y2-mr.f.y1;     mr.f.repaint(); } } </pre>
--	---

Use of inner classes:

If MyMouseListener or MyKeyAdapter class is defined within the scope of the applet (as inner classes), it can access all the methods & variables of the applet directly.

A reference variable of applet class is no longer needed.

Anonymous classes make the program still more concise by declaring and instantiating the class at the same time and the class is not given any name.

Pgm no 30 can be rewritten as follows using inner class and anonymous inner class.

<p>a)using inner class</p> <pre> import java.awt.*; import java.applet.*; import java.awt.event.*; /* &lt;applet    code="innerapplet"    height=400 width=400&gt; &lt;/applet&gt; */ public class innerapplet extends Applet {     public String msg="";     public void init()     {         addMouseListener(new         MyMouseListener());         // no reference is passed to the constructor         addKeyListener(new MyKeyAdapter());     }     public void paint(Graphics g)     {         g.drawString(msg,10,40);     }     class MyMouseListener extends     MouseAdapter     {         public void mouseClicked(MouseEvent me)         {             showStatus("Mouse clicked");         }     } } </pre>	<p>b)using anonymous inner class</p> <pre> import java.awt.*; import java.applet.*; import java.awt.event.*; /* &lt;applet    code="anonymousapplet"    height=400 width=400&gt; &lt;/applet&gt; */ public class anonymousapplet extends Applet {     public String msg="";     public void init()     {         addMouseListener(new MouseAdapter()         {             public void mouseClicked(MouseEvent             me)             {                 showStatus("Mouse clicked");             }         });         addKeyListener(new KeyAdapter()         {             public void keyTyped(KeyEvent me)             {                 msg+=me.getKeyChar();                 showStatus("Key typed");                 repaint(); // if repaint() is not called,                 paint() will not be called             }         });     } } </pre> <div data-bbox="1110 1199 1601 1325" style="border: 1px solid black; padding: 5px; width: fit-content;">         Anonymous inner class i.e., the class which extends the MouseAdapter is not given a name     </div>
--	---

<pre> } class MyKeyAdapter extends KeyAdapter {     public void keyTyped(KeyEvent me)     {         msg+=me.getKeyChar();         showStatus("Key typed");         repaint(); // if repaint() is not called,         paint() will not be called     } } } </pre>	<pre> } }); }     public void paint(Graphics g)     {         g.drawString(msg,10,40);     } } </pre>
--	---

## **JDBC (Java Database Connectivity)**

### Introduction to Databases:

A database is an organized collection of related data. Traditional file systems are inefficient in retrieving and managing data. A database server helps in organizing the data efficiently and retrieving it. Examples of database servers are DB2, mysql, Sybase etc.

Let us see how to work with mysql database. (Install mysql –an open source software)

Generally in any Relational database, data is stored in the form of a table. For example, if the student id, name, total and cgpa details of 10000 students have to be stored, they are stored in a table. The details of one student form a single row (termed tuple). So, for the example stated above, the table will be containing 10000 rows (tuples). The 4 fields that describe the student (id, name, total and cgpa) constitute the columns of the table and called as attributes.

In mysql, a database has to be created first, inside which any number of tables can be constructed.

Operations to be performed on a relational database are expressed in a language that was specifically designed for this purpose, the SQL (Structured Query Language).

### SQL Syntax:

- i) Creating a database: create database databasename;  
Eg., create database university;
- ii) To work inside the newly created database: use databasename;  
Eg., use university;
- iii) Creating a table: create table tablename(attribute1, datatype, attr2 datatype, attr3 datatype);  
Eg., create table student(studid int(3), studname varchar(20), total int(3), cgpa decimal(3,1));
- iv) Inserting values in to the table: insert into tablename values(value1, value2, value3, value);  
Eg., insert into student values(1,"Anu",89,9.3);  
insert into student values(2,"Ajay", 92, 9.5);
- v) A) Retrieving the data from the table: select \* from tablename;  
Eg., select \* from student;  
B) Retrieving the data based on condition: select \* from tablename where condition;  
Eg., select \* from student where studid=2;

Screenshot of mysql console:

```

H:\Program Files\wamp\bin\mysql\mysql5.0.51b\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.0.51b-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database university;
Query OK, 1 row affected (0.02 sec)

mysql> use university;
Database changed
mysql> create table student(studid int(3),studname varchar(20),total int(3),cgpa
decimal(3,1));
Query OK, 0 rows affected (0.16 sec)

mysql> insert into student values(1,"Anu",89,9.3);
Query OK, 1 row affected (0.17 sec)

mysql> insert into student values(2,"Ajay",92,9.5);
Query OK, 1 row affected (0.17 sec)

mysql> select * from student;
+-----+-----+-----+-----+
| studid | studname | total | cgpa |
+-----+-----+-----+-----+
|      1 | Anu      |    89 | 9.3 |
|      2 | Ajay     |    92 | 9.5 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> select * from student where studid=2;
+-----+-----+-----+-----+
| studid | studname | total | cgpa |
+-----+-----+-----+-----+
|      2 | Ajay     |    92 | 9.5 |
+-----+-----+-----+-----+
1 row in set (0.08 sec)

```

A Java application can be made to work with these databases. A Java program can be written to manipulate the tables that we have created. Eg., Another row of student details can be inserted or the table contents can be displayed through the Java code.

### JDBC (Java Database Connectivity)

- A library consisting of a set of classes and interfaces that help for establishing and maintaining the connection between the Database and the Java application program. Once the connection is established, SQL (Structured Query Language) can be used to access and process the contents of the database.
- These classes and interfaces (which constitute the JDBC library) are defined in **java.sql** package. So, import this.

Steps to follow:

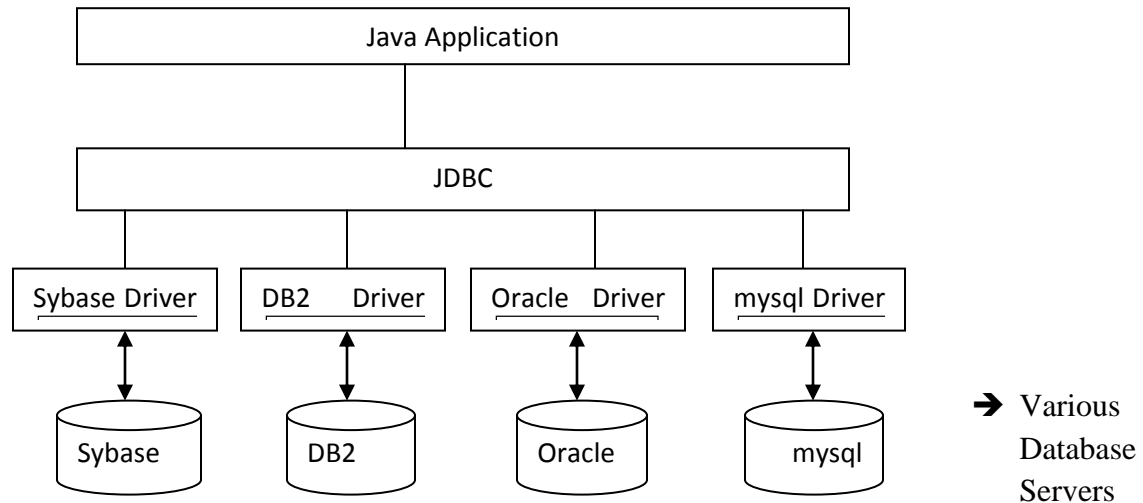
Step-I:

Load the appropriate Driver

JDBC application program does not have to be modified for connecting to various types of Database servers.

Then, how does JDBC manage to communicate with any type of database server?

- By implementing the JDBC interface for specific databases which are called Drivers. A specific Driver is available for each database. (i.e., individual drivers exist for mysql, Oracle, DB2 and other DB servers)



To load the appropriate driver explicitly, the static `forName()` method in the predefined class named 'Class' can be called by passing the driver class name as the String argument. `Class.forName("com.mysql.jdbc.Driver");` // this is the driver for my mysql database. This method can throw an exception of type `ClassNotFoundException`, if the driver class cannot be found. So, monitor it inside a try block and handle it.

```

try
{
    Class.forName("com.mysql.jdbc.Driver");
}
catch(ClassNotFoundException e)
{
    System.out.println(e);
}
  
```

#### Step-II:

Before executing any SQL statements, a Connection object must first be created, which represents an established connection (session) to a particular data source.

Creating a Connection object:

The DriverManager class helps in this regard.

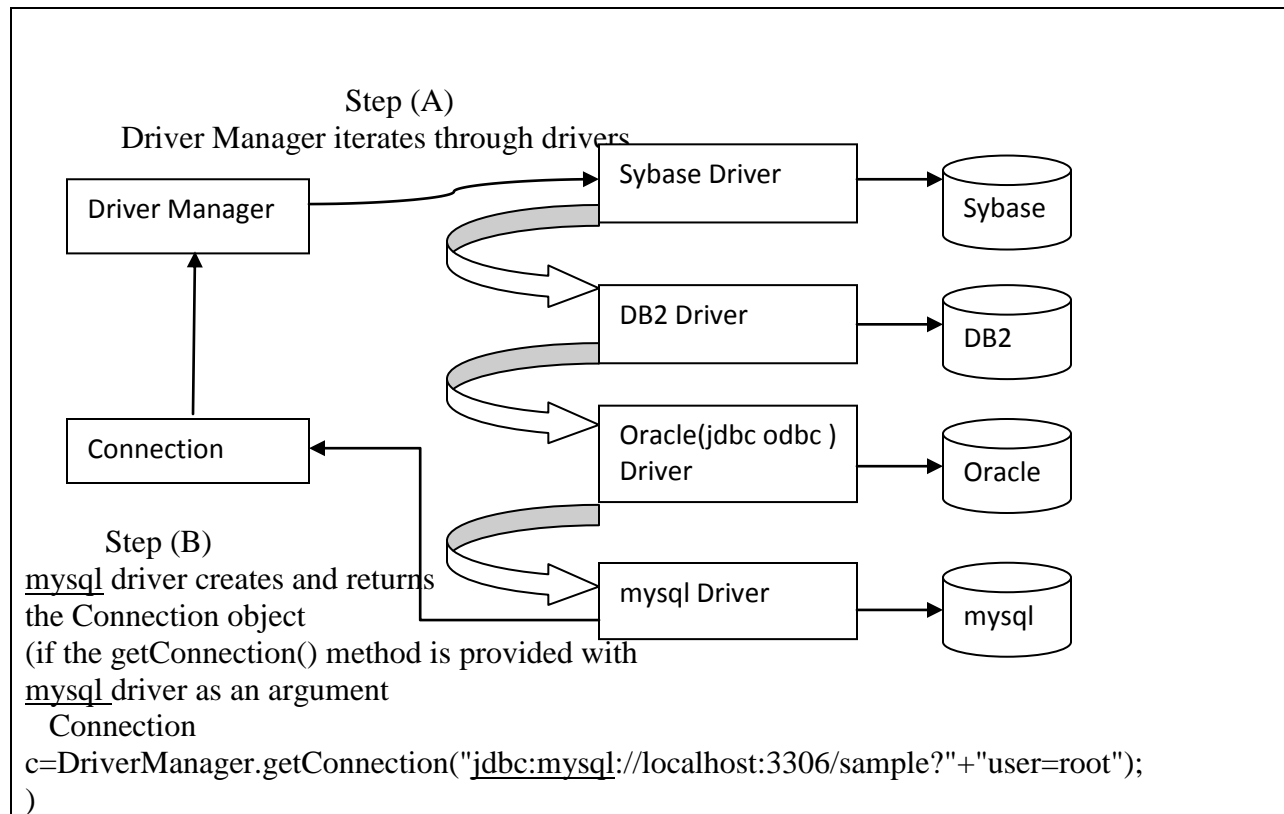
```
DriverManager.getConnection(datasourceURL);
```

Defines the URL that identifies where the database is located

A) When the `getConnection()` method of DriverManager class is called, the DriverManager iterates through the various drivers that are registered with the DriverManager and asks each one in turn, if it can handle the URL that has been passed to it.

B) The first driver that can handle the URL, passed to it, creates a Connection object and returns it to the application by way of DriverManager.





Note: The Java application need not implement code to interact with the various driver objects individually. The DriverManager class takes care of communicating with all the drivers. This method throws SQLException which has to be handled or forwarded. So, include this statement inside a try-catch block.

```
try
{
    Connection c=DriverManager.getConnection("jdbc:mysql://localhost:3306/sample?"+"user=root");
}
catch(SQLException e)
{
    System.out.println(e);
}
```

### Step-III:

Creation of Statement object which provides the workspace to create an SQL query, execute it and retrieve any results that are returned.

- Created by calling the createStatement() method of a valid Connection object.

Statement s=c.createStatement();

### Step-IV:

The results of executing an SQL query are returned in the form of an object that implements the ResultSet interface.

ResultSet rs=s.executeQuery("select \* from student");

This object contains a cursor that can be manipulated to refer to any particular row in the result set. It initially points to a position immediately preceding the first row. Calling the next() method for the ResultSet object will move the cursor to the next row.

The next() method returns true if the move is to a valid row and false if there are no more rows(i.e., has reached the end)

Step-V:

Accessing data in a ResultSet:

Using the ResultSet reference, the value of any column for the current row can be retrieved by name or by position. Class implementing the ResultSet interface contains methods to read all types of data.

-getBoolean(), getByte(), getDouble(), getFloat(), getInt(), getShort(), getLong(), getString().

Eg., String sid=rs.getString("studid");

1) Program to display the contents of the student table:

Download mysql connector and save it in the same folder which contains the Java program

Use executeQuery() method to execute the select SQL command

File name: jdbc1.java

```
Import java.sql.*;
public class jdbc1
{
    public static void main(String[] args)
    {
        Connection c=null;
        Statement s=null;
        ResultSet rs=null;
        try
        {
            Class.forName("com.mysql.jdbc.Driver"); // ➔ loads the appropriate driver
            c=DriverManager.getConnection("jdbc:mysql://localhost:3306/university?"+"user=root");
            //➔ establish the connection by creating connection object
            s=c.createStatement(); //➔ provides the workspace to create and execute the query
            rs=s.executeQuery("select * from student");
            while(rs.next())
            {
                String sid=rs.getString("studid");
                String sname=rs.getString("studname");
                String total=rs.getString("total");
                String cgpa=rs.getString("cgpa");
                System.out.println(sid+" "+sname+" "+total+" "+cgpa);
            }
            rs.close();
            s.close();
            c.close();
        }
        catch(SQLException | ClassNotFoundException e)
        {
            System.out.println(e);
        }
    }
}
```

```
}  
}
```

Compilation:

C:\Desktop\Java\Applets> javac jdbc1.java

Execution:

C:\Desktop\Java\Applets >java -cp .;" C:\Desktop\Java\Applets \mysql-connector-java-5.1.10-bin.jar" jdbc1

1 Anu 89 9.3

2 Ajay 92 9.5

Note: include the current directory as well as the path for the jar file which contains the mysql driver class.

2) Program to insert two more rows in to the student table:

Use executeUpdate() method for executing the insert SQL command.

File name: jdbc2.java

```
import java.sql.*;  
public class jdbc2  
{  
    public static void main(String[] args)  
    {  
        Connection c=null;  
        Statement s=null;  
        ResultSet rs=null;  
        try  
        {  
            Class.forName("com.mysql.jdbc.Driver");  
            c=DriverManager.getConnection("jdbc:mysql://localhost:3306/university?"+"user=root");  
            s=c.createStatement();  
            s.executeUpdate("INSERT INTO student VALUES(3,'Akshay',80,9.0)");  
            s.executeUpdate("INSERT INTO student VALUES(4,'Ajit',73,8.6)");  
            rs=s.executeQuery("select * from student");  
            while(rs.next())  
            {  
                String sid=rs.getString("studid");  
                String sname=rs.getString("studname");  
                String total=rs.getString("total");  
                String cgpa=rs.getString("cgpa");  
                System.out.println(sid+" "+sname+" "+total+" "+cgpa);  
            }  
            rs.close();  
            s.close();  
            c.close();  
        }  
        catch(SQLException | ClassNotFoundException e)
```

```
{  
System.out.println(e);  
}  
}  
}
```

Compilation:

C:\Desktop\Java\Applets> javac jdbc2.java

Execution:

C:\Desktop\Java\Applets >java -cp .;" C:\Desktop\Java\Applets \mysql-connector-java-5.1.10-bin.jar" jdbc2

1 Anu 89 9.3

2 Ajay 92 9.5

3 Akshay 80 9.0

4 Ajit 73 8.6

3) Applet Program to get the student details through text boxes and insert them in to the table.

```
import java.sql.*;  
import java.awt.*;  
import java.awt.event.*;  
import java.applet.*;  
/*  
<applet code="jdbcapplet" width=400 height=400>  
</applet>  
*/  
  
public class jdbcapplet extends Applet implements ActionListener  
{  
    TextField t1,t2,t3,t4;  
    Button b;  
    public void init()  
    {  
        t1=new TextField(5);  
        t2=new TextField(20);  
        t3=new TextField(3);  
        t4=new TextField(3);  
        add(t1);  
        add(t2);  
        add(t3);  
        add(t4);  
        b=new Button("Insert");  
        add(b);  
        b.addActionListener(this);  
    }  
    public void actionPerformed(ActionEvent ae)  
    {
```

```

    workWithDatabase();
}
public void workWithDatabase()
{
    Connection c=null;
    Statement s=null;
    ResultSet rs=null;
    try
    {
        Class.forName("com.mysql.jdbc.Driver");
        c=DriverManager.getConnection("jdbc:mysql://localhost:3306/university?"+"user=root");
        s=c.createStatement();
        String studid=t1.getText();
        int id=Integer.parseInt(studid);
        String studname=t2.getText();
        String studtotal=t3.getText();
        int tot=Integer.parseInt(studtotal);
        String studcgpa=t4.getText();
        double cgpa=Double.parseDouble(studcgpa);
        s.executeUpdate("INSERT INTO student VALUES(id,studname,tot,cgpa)");
        rs=s.executeQuery("select * from student");
        while(rs.next())
        {
            String sid=rs.getString("studid");
            String sname=rs.getString("studname");
            String total=rs.getString("total");
            String cgpa=rs.getString("cgpa");
            System.out.println(sid+" "+sname+" "+total+" "+cgpa);
        }
        rs.close();
        s.close();
        c.close();
    }
    catch(SQLException | ClassNotFoundException e)
    {
        System.out.println(e);
    }
}
}

```

Note: While executing this program, the appropriate security policy file should be included.