Java

BASICS : Quick guide to beginners

Hitesh Arjaria

Acknowledgments

I am writing this book to help teenagers & beginners in understanding the very basics concepts of java programming. Java offers many features and I will cover them along with Spring MVC, Springboot, Integration of AWS services etc. in my upcoming book 'Evolution in Java". As of now, I consider this effort as a start of learning of Java programming.

I am thankful to all my friends and relatives who supported me so far. My sisters Sonu and Swati, my parents and everyone around me. I am also thankful to some of my friends who inspired me to write Books at the time when I was not sure that I should start as a professional writer. For more than 15 years I am in habit of writing but never thought of becoming someone to publish the stuff in public. Apart from training programs, I was introvert by nature and usually never shared my personal work to people, anyhow they went through my work and were excited, then we talked for days on publishing. It took me 7 more years to finally start publishing. Now, I am publishing my work on my website and not to any publisher but soon I would go in that direction as well.

As I made a promise to Lord Shiva in Somnath temple, Gujrat that I would do all my educational writing work without accepting any money and will forward all the donation to Indian Army if it is coming from any of my educational writing work. I am really grateful that I am keeping that promise and hence there is no cost attached to this book. So, enjoy....

About the Author

Hitesh Arjaria

Currently, Senior Manager for an IT MNC in Bangalore, India. Leading teams that are working for fintech. applications. Area of expertise are Cloud migration, Java development, Database and Agile methodology.

Trained around 10 thousand people for IT industry and has vast experience in many programming languages and technologies. Apart from tech career, like to write novels, books and articles encouraging Holistic approach towards life and ability to think in terms of complete scenario.

“Dedicated to my Guru OSHO, Bhagwan shri Rajneesh. And, for the memory of the day – 19/05/2018 when I became his Sannyasin at Osho Amritdham, Jabalpur India.”

- Hitesh Arjaria

Chapters

Java capability area, Datatypes and Operators

Java was developed by a team of engineers in Sun Microsystems in 1991 as a part of a research project, which was led by James Gosling and Patrick Naughton. It took 18 months to develop the first working version. This language was initially called "Oak" but was renamed as "Java" in 1995.

Design goals and Features of JAVA-

According to the White Paper written by the author of java, 11 design goals of java are as follows-

Simple	**Portable**
Object Oriented	**Interpreted**
Distributed	**High Performance**
Robust	**Multithreaded**
Secure	**Dynamic**
Architecture Neutral	

Later they became the key features of java.

The simplest way to learn and master the Java programming is to understand one thing; EVERYTHING in java is in terms of class/object. For example, if you click using mouse, in java program, an object of Event class is created and further calls other functions. When you try to establish connection between server/client, for java program, it is Socket object or ServerSocket object. Even if an exception occurs, it is nothing but an object of any of the Exception classes.

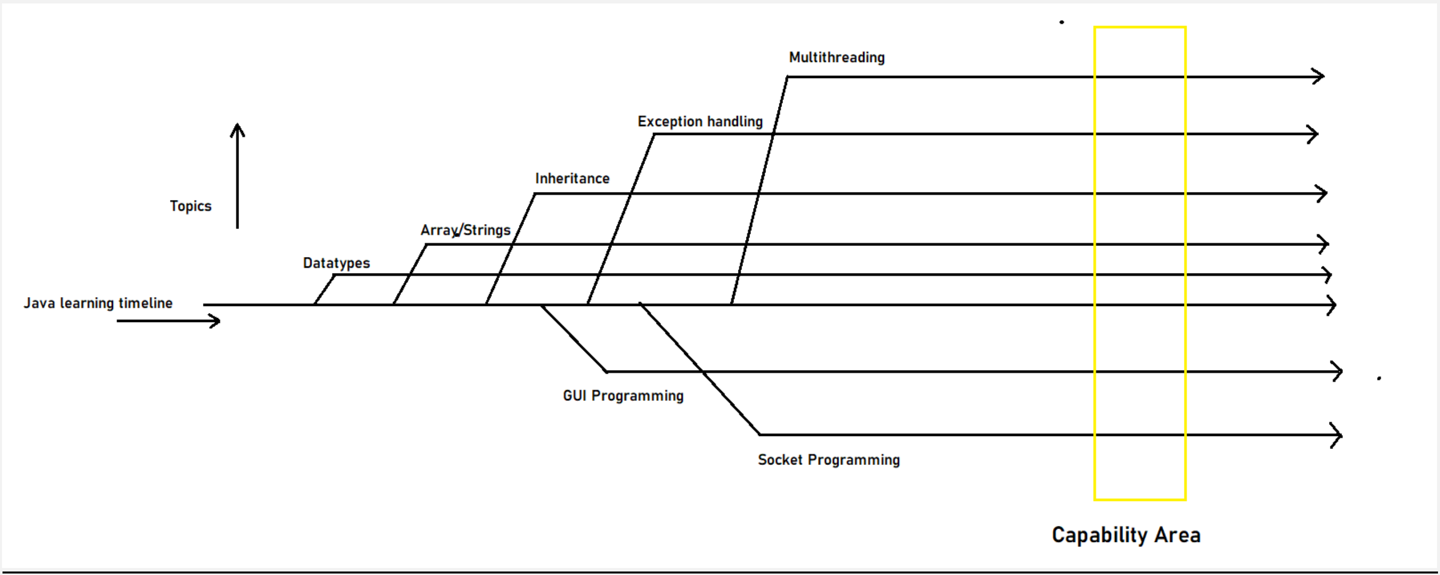
When you integrate any module or AWS service, just include dependency, create objects as per your requirements and calls the methods using those objects. SIMPLE NAAA….

All you have to do, is to understand the requirements, google the methods and classes that are there to do that task, then create the object of those classes and use the methods or functions. Just be careful for two things: Static and singleton classes. Because they are dealt in java differently.

Once you realise this, it is more convenient to understand Inheritance, polymorphism, and other OOPS concepts.

How to reach to capability area of Java programming?

Here capability area refers to the capability of solving problems in java and capability of developing the complex application with requirements demanding the multiple domains. Refer the following diagram that would help you to learn and master the java.



Once you start a topic like array, it starts but never ends, as you learn further, new topics will come like inheritance, GUI programming etc, where you will implement the array but in some different way, so definitely you will learn something new in array. Same applies for inheritance, after that when you start learning the Multithreading, don't consider that inheritance is completed and multithreading is started, assume that you are learning the multithreading along with inheritance and somewhere you will use it in multithreading, like Thread class is inherited to the class in which multithreading is supposed to be applied.

You can see yellow box in diagram which is called capability area. Eventually this area will expand as you proceed and include new topics in your journey, if you proceed with this approach. And that is the difference between learning the other subjects and programming. **You can never close a chapter in programming, they are always tending to expand.**

So be prepared to learn them simultaneously once you grow. This will lead you to be good developer and problem-solving technologist. When you start working in cross domains and integrating the other component and services this capability area will be extended larger in multiple dimensions

Let's Begin

Simple Java program looks like:

```
public class myapplication
{       public static void main(String args[])
        { System.out.println("Hey I am successful in writing my prog");
        }
}
```

This program has main() method inside of class myapplication. As java is fully supported OOP language so nothing can be put outside of classes, not even main function. Now let us understand the signature of the main() method:

The main() method is:

public: so that it can be accessed by anything, including the java technology interpreter.

static: As we know that in OOP, to call static member of a class, we don't have to create the object of that class, hence in java all the objects are created at runtime so there is no way to create them prior to start the execution of that program. The main() must be declared as static so that there is no need to create the object of myapplication class to call it, hence we will not trap in the circle.

void: Indicates that it does not return anything once called on command line. You can put some datatype if relevant values are to be returned by main.

String args[ ]: Declared as parameter in main(), it receives values for command line arguments. args[0] stores the first command line parameter. So args can be called as **reference of String array** (will be discussed later) which accepts the command line arguments. Each command line argument, separated by [SPACE] will be assigned to each of the String object of args[] array.

Object creation: In java, objects are created by explicitly calling the constructor of the class for which it holds the reference. Please refer the chapter "Object Oriented Programming Concepts in Java" (Java programs do not directly deal with objects. Rather it deals with the reference of the object, which the constructor returns).

Compiling & Running a Program

To compile the myapplication program, execute the compiler, javac, (jdk) specifying the name of the source file on the command line, as shown here:

```
C:\>javac myapplication.java
```

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

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

```
C:\>java myapplication
```

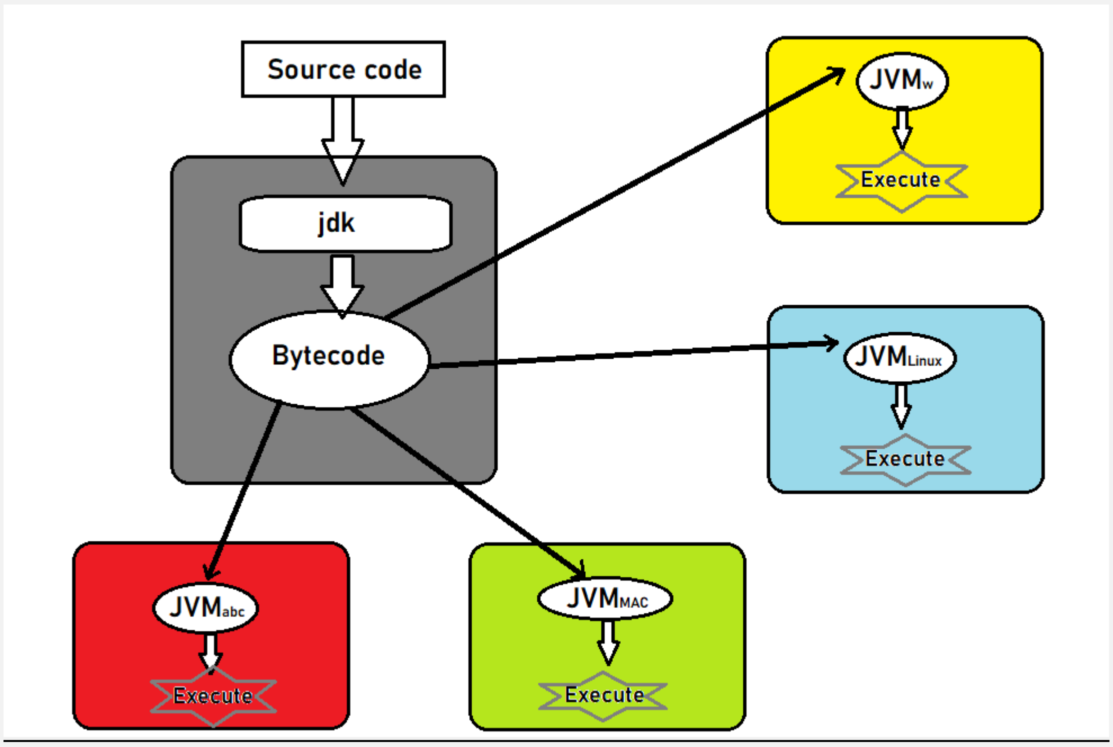
When the program is run, the following output is displayed:

```
Hey I am successful in writing my prog.
```

How Java is Cross-platform programming language?

First, java is not "Platform independent" language, rather it is better to call it "Cross-platform" language. Refer the following diagram:

Java source code compilation and execution



Once source code is compiled by jdk, java development kit, it is not getting converted into executable code. Rather then jdk converts it into an intermediate code which is called as Bytecode.

Command to compile the java code from CMD:

```
C:\> javac myapplication.java
```

Java vendors collaborated with almost all the popular platform providers such as Microsoft, Linux, MAC OS etc. and with the specification provided, they developed JVM for each of these platforms. JVM, java virtual machine, is a term used for those patches or programs that enables the machine for running the java programs, application.

It is clear, that each platform has its own JVM which is NOT platform independent. JVM is always platform dependent and JVM for one platform can not be installed and run on another platform. For example, JVM for windows can not work in place of JVM for Linux, although name is same JVM but they are 2 different things, that is why I mentioned them as JVM$_w$ and JVM$_{linux}$ in diagram.

It is a holistic approach, adopted world-wide to make java applications runnable in multiple platforms. Have separate JVM or JRE patch for each platform which will run/execute the Bytecode according to that platform.

So, it seems to a developer that his same code is executable across the platforms, provided each one of them is java-enabled (Having the JVM). That is how it works; OS or platform executes the JVM and then JVM runs the Bytecode of application. IF a platform abc is there but NO JVM is developed and installed in it then there is no way a java program is going run on it. First, the JVM_{abc} must be developed by vendors, then it must be installed in the machine then only your java program will be running on it. That is why we must not call java a platform independent as independent means you do not need anything like separate JVM for each machine at all.

Now, consider this command to run the program:

```
C:\> java myapplication
```

Here OS is executing the java, which is nothing but JVM and this java in turn is executing the file myapplication. All the bytecode are having the .class file extensions. In upcoming chapters, JVM will be discussed in detail.

Java virtual machine (JVM) in detail

It interprets compiled Java binary code (called bytecode) for a computer's processor (or "hardware platform") so that it can perform a Java program's instructions**.**

A Java technology runtime environment runs code compiled for a JVM and performs three main tasks:

- Loads Code - Performed by the class loader
- Verifies Code - Performed by the bytecode verifier
- Executes Code - Performed by the runtime interpreter

Class Loader

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

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

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

Verification Process

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

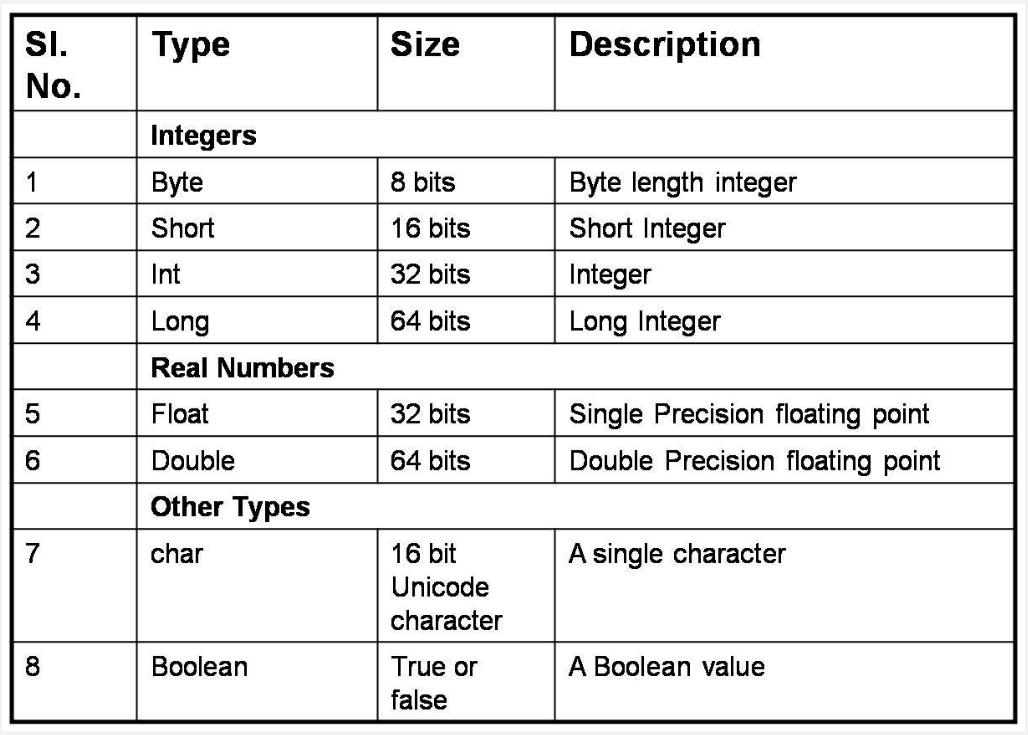
1. There are no access restriction violations
2. The code causes no operand stack overflows or underflows
3. The types of parameters for all operational codes are known to always be correct.

No illegal data conversions, such as converting integers to object references, have occurred.

Datatypes and Operators

Java is a strongly typed language. This means that every variable must have a declared type. You may already know that there are eight primitive types in Java. Let's look at them however, Four of them are integer types; two are floating-point number types; one is the character type char, used for code units in the Unicode encoding scheme, one is a boolean type for true/false values.

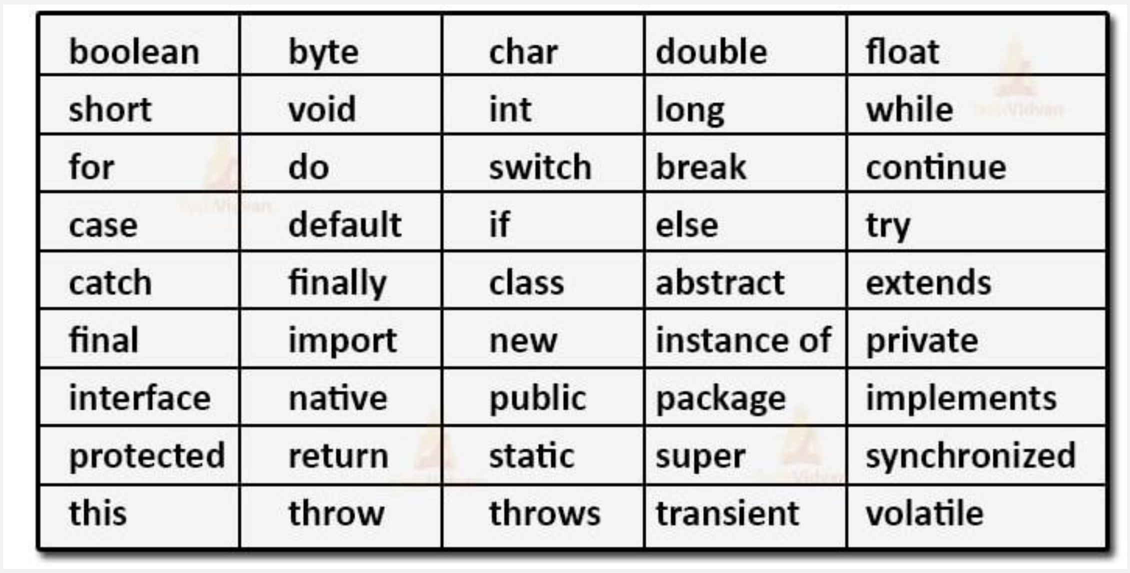
Data Types



Sl. No.	Type	Size	Description
	Integers		
1	Byte	8 bits	Byte length integer
2	Short	16 bits	Short Integer
3	Int	32 bits	Integer
4	Long	64 bits	Long Integer
	Real Numbers		
5	Float	32 bits	Single Precision floating point
6	Double	64 bits	Double Precision floating point
	Other Types		
7	char	16 bit Unicode character	A single character
8	Boolean	True or false	A Boolean value

Java Keywords

There are 52 reserved keywords defined in the Java language.



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

Operators

OPERATOR	RESULT

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

The Bitwise operators

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

<<=	Shift left assignment

Relational Operators

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

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

Boolean Logical operators

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

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

The ternary operator

Java includes a special ternary (three way) operator that can replace certain types of if-then-else statements.

expression1 ? expression2:expression3;

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

Type conversion/Type casting

Java supports two types of Type-castings –

1. Primitive data type casting:

It is of 3 types-

A. Implicit casting (widening conversion): A data type of lower size (occupying less memory) is assigned to a data type of higher size. This is done implicitly by the JVM. The lower size is widened to higher size. This is also named as automatic type conversion. Example-

```
int x = 10;             // occupies 4 bytes

double y = x;                   // occupies 8 bytes

System.out.println(y);  // prints 10.0
```

In the above code 4 bytes integer value is assigned to 8 bytes double value.

B. A data type of higher size (occupying more memory) cannot be assigned to a data type of lower size. This is not done implicitly by the JVM and requires explicit casting; a casting operation to be performed by the programmer. The higher size is narrowed to lower size

2. Reference type casting.

Reference type casting is nothing but assigning one Java object to another object. It comes with very strict rules and is explained clearly in Object Casting. In this type casting, object of derived class is assigned to the reference of Base class. It also means that for **this type casting, Inheritance is compulsory**.

Please refer to the chapter Exception handling for more details, later in this book.

Practice Programs:

- Install and start the JVM, JDK, and Netbeans to run java application.
- Write a program to print “Welcome to the java programming”
- Write a program for Addition, Subtraction, Division, and Multiplication of two variables.

- Write a program to find the remainder by dividing 225 by 7.

- Write a program to swap the values of two variables with/without using a third variable.

- Write a program for The type-conversion as given-

  Implicit/Explicit type-conservation

  a. int to float

  b. int to long

  c. int to char

  d. int to boolean

  e. char to float

  f. char to int

  g. char to long

  h. char to Boolean

  i. float to int

  j. float to char

  k. float to long

  l. float to Boolean

- WAP in which if the age of a person is less than 18 years than it will print "You are not eligible for voting" otherwise it will print "You are eligible for voting".
- A user wants to open an account in a bank that demands a user to be Indian, 20 years old, and have at least 2000/- for deposit. WAP to print "Your account is successfully opened" if all the above conditions are met.
- Write a program for a community in which people are differentiated on the basis of their monthly income. To get enrolled with such community one need to provide his monthly

income. As per the income your program should print his category in the society. Categories are-

Income	Category
Less than 1000/-	Poor
Between 1000/- and 2000/-	At the poverty line
Between 2000/- and 3000/-	Lower middle class
Between 3000/- and 10000/-	Middle class
Between 10000/- and 30000/-	Upper middle class
30000/- above	Rich

- Write a program to print numbers from 1 to 10 each number in a line.
- Write a program to print all the characters from a to z and from A to Z.
- Write a program to print the average from numbers 1 to 50.
- Write a program to print all the even numbers 1 to 50. Also print all the odd numbers 1 to 50. In which all the numbers should be separated with a space.
- Write a program to print all the numbers between 1 to 100 that are divisible by 7.
- Write a program to print all the numbers between 1 to 1000 that are divisible by 24 (checking the divisibility with 8 & 3).
- Write a program to find factorial of number 8.

- Write a program to check whether the sum of the digits of a number is divisible by 3 or not.
- Write a program to revert a number also check whether a number is a Palindrome or not.
- Write a program to check whether a number is an Armstrong or not.
- Write a program to check whether a number is a perfect number or not.
- Write a program to check whether a number is a prime number or not. Also print all the prime numbers from 1 to 50.
- Write a program to print the triangle of star.

```
*
**
***
```

Also in reverse order-

```
***
**
*
```

And

```
            *
        *       *
    *       *       *
*       *       *       *
```

- Write a program to find the sum up to n terms of series-

  1/2 + 1/3 + 1/4 + 1/5 + 1/6.......................

  1/1! + 2/2! + 3/3! + 4/4! + 5/5! + 6/6! +..............

  1!/1 + 2!/2 + 3!/3 + 4!/4 + 5!/5 ....

  1/2! + 3/4! + 5/6! + 7/8! ....

  1!/2! + 2!/3! + 3!/4! + 4!/5! +........................

- Switch case-

  Write a program to print-

  Monday if user enters 1.

  Tuesday if user enters 2.

  Wednesday if user enters 3.

  Thursday if user enters 4.

  Friday if user enters 5.

  Saturday if user enters 6.

  Sunday if user enters 7.

Write a program to check 226 is even or odd using switch case.

Write a program to print division.

Arrays, their properties and implementation

"Array is the finite, ordered, list of homogeneous data elements."

Let's list the properties of Arrays on the basis of above definition, such as;

Finite: Length of an array is finite that means it cannot be changed once declared.

Ordered: All the elements of an array are arranged in ascending order. Indexing from 0 to (n-1) for the size of n.

List: Array is used for continuous memory allocation; means they are one after another. There is no gap between any of them for any size.

Homogeneous: The data type of all the elements of an array must be same.

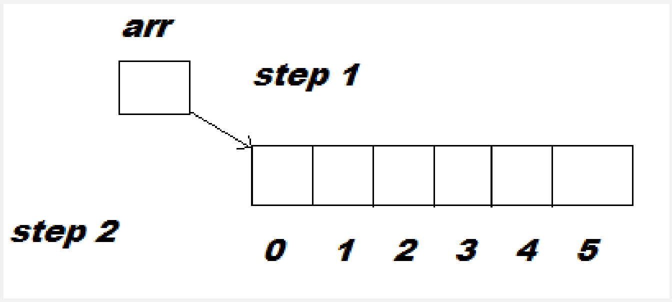
Creation of Arrays

Remember that in java, Memory to Arrays is allocated at run time.

Arrays are created in 2 steps. For example-

```
Int arr[];                  //step 1

arr=new int[6];             //step 2
```

In step 1 a reference is created that can hold the address of an array. While in step 2, a block of 6 integers is created at run-time with the help of new operator and its base address get assigned in the reference created in step 1.



Initialization of an array-

There are 3 types of initializations of an array-

1. Default Initialization- In java by default all the elements of an array are initialized by 0(in case of int,long),by 0.0 (in case of float, double), by NULL(in case of char of string), or by false(in case of boolean).

2. Implicit Initialization- When initialized at the declaration line. Ex-

```
int arr[5]={1,2,3,4,6};

char arr[]={'e','r','h','u','l'};
```

3. Explicit Initialization- After the declaration, values entered either by user or by the instructions for each individual member (usually in a loop). Ex-

```
 for(i=0;i<6;i++)

 { arr[i]=i;}
```

# Multidimensional Array-

```
Int arr[3][4]={{2,3,4,5},{8,9,7,6},{6,6,8,9}};
```

Here number of rows are 3 and number of columns are 4.

Please refer the below programs for your practices

- Write a program to print all the elements of an array.
- Write a program to find the average of all the elements of an array.
- Write a program to find the average of all the elements having even index.
- Write a program to find the average of all the elements having odd index.
- Write a program to swap the first and the last elements of an array.

- Write a program to revert an array and assign it-

  (a). In to the same array.

  (b). In to a different array.
- Write a program to find the sum of corresponding elements of two arrays and store them in to a third array.
- Write a program to assign all the elements of two array of size 5 in to another array of size 10 so that all the elements of second array will get assigned after the first array.
- Write a program to assign all the elements of two array of size 5 in to another array of size 10 so that all the elements of first and second array will get assigned alternatively(that is first array elements get assigned in even positions and second array elements get assigned in odd positions).
- Write a program to assign elements of three arrays in to another larger array alternatively.
- Write a program to search an integer in an array. If it is present then print "present" otherwise print "Not present".
- Write a program to print the occurrence of an integer in an array.
- Write a program to swap smallest element of an array with the first element.
- Write a program to swap largest element of an array with the last element.
- Write a program to sort an array using insertion sort.
- Write a program to sort an array using selection sort.
- Write a program to sort an array using bubble sort.
- Write a program to sort an array using quick sort.
- Write a program to sort an array using heap sort.
- Write a program to sort an array using merge sort.
- Write a program to insert and print elements in two dimensional arrays.
- Write a program a for matrix addition, subtraction.
- Write a program for matrix Multiplication.

Mixed problems of for, if-else and array/string

Please make programs for following problems:

1. 9 windmills are installed in 16 equal slots in a 4*4 form. Each slot is 100 sq ft in size. Snapshot of these windmill is taken from drone camera above 200 ft from ground is as follows –

c1	c2	c3	c4	NORTH	
1		0	0	1	
0	1	0	1		
0	0	1	0		
1		1	1	1	SOUTH

Where 1 represents the windmill and 0 represents a vacant slot. Wind is blown from north to south as shown in diagram above. Wind has certain strengths. If wind passes through a windmill its strength decreases by 1 and remains same if passes through vacant slots.

Mills of each column are interconnected and total energy generated is the sum of energy generated by each mill which is 4 if it is turned on by wind.

Wind coming from north at 2 instances T1 and T2 with their strengths applied on each wind-mill columns is given below –

T1= (1 2 0 1)

T2=(3 0 2 3)

Calculate the energy generate by winds of each column and print –

I. The Name of column generates highest energy after T1.
II. The Name of column generates highest energy after T2.

2. After 3 days in Olympic games, ranking of nations are as follows –

	Gold	Silver	Bronze	Total
USA	13	11	9	33
CHINA	11	12	7	30
UK	9	8	6	23
FRANCE	7	11	13	31
RUSSIA	6	9	5	20
ITALY	4	10	10	24

Ranking is decided on the basis of highest number of gold medal wins by a nation. If two nations have same number of gold medal then we see highest number of silver medal, if two nations have same number of gold medals as well as same number of silver medals then we see their bronze medal.

On day 4 many competitions takes place and medals get distributed as follows-

	Gold	Silver	Bronze
USA	3	0	4
CHINA	2	5	1
UK	1	0	2
FRANCE	4	2	1
RUSSIA	0	2	4
ITALY	2	1	0

Create a string array to store the countries name and 2d array for medals. Countries are assigned to string array according to their rank and it is corresponding to their medals stored in 2d array. For example if USA is at the top of string array followed by CHINA at second rank then in 2d array first row will hold the medals won by USA and second row will hold the medals won by CHINA and so on.

Now update the table and print the answers of following questions-

I. How many gold medals does the FRANCE won after 4 days.

II. Gold, silver and bronze medals won by CHINA along with total number of medals after 4 days.

III. What nation is at the bottom of the list ? show all the medals won by it.

IV. Ranking of nation after 4 days of games. (Create a separate table for the scores at day 4)

3. In PUBG, there are 6 squads available in map – “Miramar”, where each squad can have exact 4 members. 15 players provide their RP as –

   22,56,23,76,53,75,17,18,19,20,79,81,45,50,61

   In 6 groups a player can join –

   Squad Q : if his RP is divisible by 10

Squad W : if his RP is a prime number

Squad E : if his RP is divisible by 3

Squad R : if his RP is divisible by both 4 and 6

Squad T : if player didn't get joined in any squad Q,W,E,R

Squad Y : if player didn't get joined in T squad

If there is any vacant space in a squad then BOTs can join the group to complete the squad. Now print the squad /squads name that has –

I. Highest BOT players

II. NO BOT players at all

III. Players with highest sum of RP

IV. Groups with equal number of human players but less than 4

4. This question has two parts A & B. Write separate codes for both the parts.

(A) Jon, Merry and nick are cousins. One day their grandfather bring them 5 packets, each of them is full of candies of different colors. Packets have candies 64, 42, 70, 48, 160 respectively. Also Colors of the candies in packets are red, yellow, green, pink and orange respectively. He opens the packets one by one. Jon is eldest so he pick half of the candies, after that merry pick one third of the remaining candies and nick get the remaining candies.

I. Print how many orange candies jon, merry and nick get.

II. By what percent the pink candies of merry are greater than her red color candies.

III. Nick's candies are what percent of Jon's candies.

IV. Print all the candies of nick color wise.

(B) Rakhi maintains a list where she put items according to their order in dictionary. User enter 10 names one after another as –

{apple, ruby, ball, cat, cot, carbon, radio, Accenture, backup, robot }

She swap the items in the list if a word comes that should be before it according to dictionary. Print the list prepared by rakhi after all the inputs.(use array of pointer to string to maintain the list)

Object Oriented Programming Concepts in Java

Classes

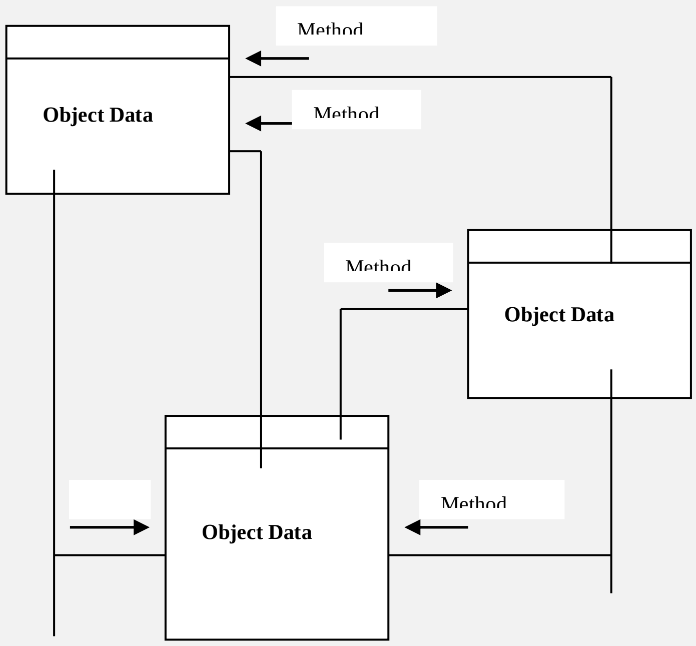
The Class is a user defined data type that holds both data & function together in a single unit. It is a template for objects. They behave like built in data types of a programming language. Once a class has been defined, we can create any number of objects belonging to that class. Ex: Fruit Grape, Fruit Water melon and Fruit Pine Apple. Here the class Fruit has different instances with the same properties but different values.

Remember: Class is not the collection of Object, it is like a template with prints the currency, collection of currency is not the template. Although, **A collection of objects is known as Array of Objects, not the class.**

Objects

Objects are the basic run-time entities in an object-oriented system. They represent the real-world entities. Ex: Person, Bank Account, Table of data or any item that the program must handle.

They contain data and code to manipulate that data. When a program is executed, the objects can interact with each other without knowing the details of each other's data or code.
Ex: A Customer Object requesting the Account Object for the Bank balance.



An object has both data and function in it. Data is known as data members and functions are known as member functions.

Object creation in java

In java objects are created by Constructors, it differs from C++ where constructors are used only to initialise the objects of the class.

Student s = new Student();

Here Student() constructor is being called at runtime using 'new' to create the object of Student class and the base address of the object will be assigned to 's' which is a reference of Student class. It is strange as compare to object creation in C++ but like array creation in java itself that we've discussed.

Constructors

Constructors are used to create and initialize the objects in java. (whereas in C++ constructors only initialize the objects).

Properties of constructors

1. Constructors are the member functions of a class that have the same name as the class name.

2. Constructors have no return type (not even void).

3. Constructors must be declared as public in a class.

4. Constructors can NOT be inherited. But base class constructors can be called in the derived class using “super” keyword.

5. In java, constructors are called at run time (using “new” keyword).

Student s=new Student();

6. Java provides a default constructor to all the classes created, but if a constructor created by the programmer, then the default one gets suppressed.

7. Constructor overloading takes place on the basis of parameters.

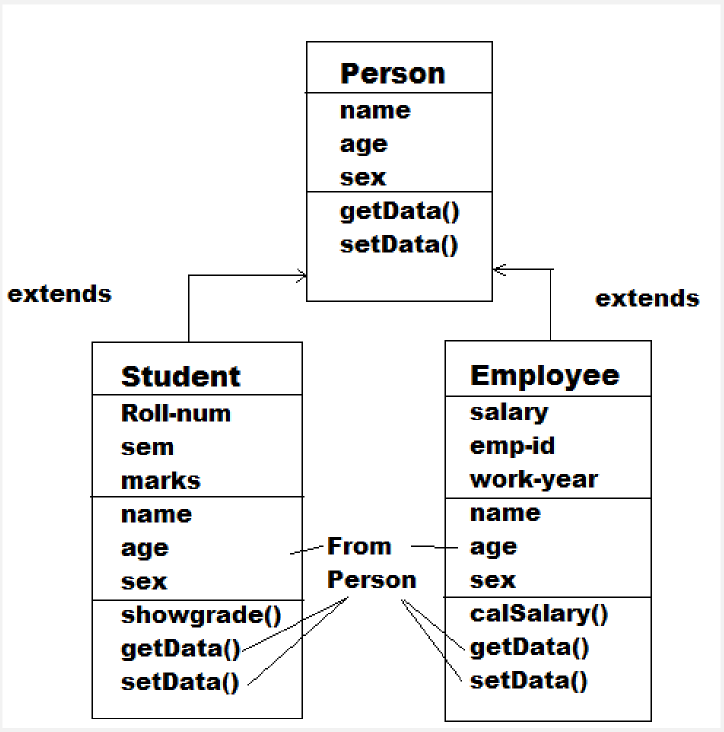
‘this’ keyword

It is used in any method to refer to the **current object** or **class instance.** That is, **this** is always a reference to the object on which the method was invoked.

```
fruit(mango m, apple a, grape g)
{       this.green=m;
        this.red=a;
        this.black=g;

}
```

Inheritance & Polymorphism in Java

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



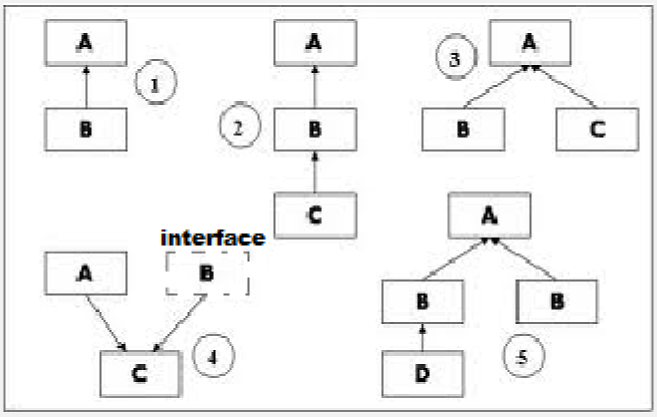
Class which is inherited by another class is called Base class, Parent class or Super class. The class which inherits another class is called Derived class, Child class or Sub class. In java Inheriting is also called **subclassed**.

Inheritance in java is public in nature and we use "extends" keyword to inherit a class.

Ex ; class B extends A

Here class B is inheriting the class A that means A is super class and B is sub class or child class.

Types of Inheritance in java



1. Single Inheritance: Where one class (B) inherits one base class (A)

2. Multi-Level Inheritance: Where one class (C) inherits one base class (B) which also inherits another base class (A)

3. Hierarchical Inheritance: Where one class is inherited or subclassed by two or more derived classes.

4. Multiple inheritance (Java does not support multiple inheritance, but derived class can acquire the features by the implementing an interface which is called extending it, also by creating the object of one class into another class which is called containership or disinheritance).

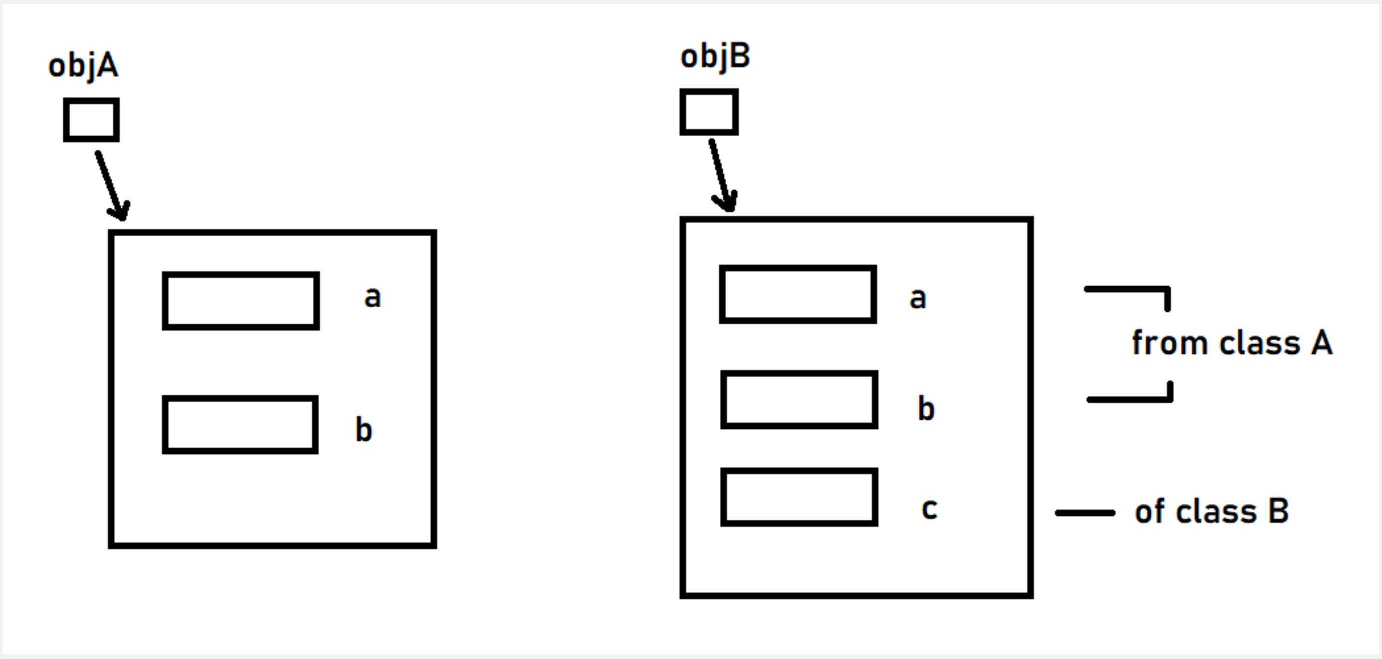
5. Hybrid inheritance: Where more than one type of inheritances are involved. Like multilevel and hierarchical inheritance are put together.

Ex:

```
Class A

{

Public:

int a;

char b;

}

Class B extends A

{
```

```
int c;

}
```

Now if we observe the memory for objects of class A & B, they would look like –



Here, objA is the object of class A and objB is the object of class B. As you can put the code for reference, objA has memory for it's data members a & b. Now, in case of objB we can see that it has memory for 3 data members but in class B we declare only 1 data member which is C, the memory for a & b is there in objB because the class B inherited them from class A so eventually they've become the data members of class B as well and hence got the memory inside the object objB of class B.

That is how code reusability is maintained, you reuse the code of class A in class B, without writing it in B, using the inheritance.

Polymorphism

Polymorphism is a feature that allows one interface to be used for a general class of actions. An operation may exhibit different behavior in different instances. The behavior depends on the types of data used in the operation.

It plays an important role in allowing objects having different internal structures to share the same external interface. Polymorphism is extensively used in implementing inheritance. Ex- Function overloading, virtual functions etc. Overloading is discussed separately in this book.

Access control-using Modifiers

Modifiers are Java keywords that give the compiler information about the nature & scope of code, data or classes. Modifiers specify, that a particular class or variable or a method is static, or final, or transient. With these access modifiers encapsulation is worked out according to the need of the program.

A member access level is determined by the access specifier, that modifies its declaration. Java's access specifiers are public, private and protected. Java also defines a default access level.

Public: It is the most generous access modifier. A public class, variable, or method may be used in any java program without restriction. In inheritance only public member take the participation it means **only public members can be inherited from a class in java.**

Private: The least generous access modifier is private. Top level classes may not be declared private.

Protected: A protected access specifier defines that only the members of the class and its sub class in which it is defined can access that feature.

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

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

- A final class may not be sub classed.
- If applied to a variable it means to say that the value is constant.
- A final object reference variable may not be changed but the data owned by the object that is referred as final can be changed.

A final method may not be overridden. This is done for security reasons and these methods are used for optimization.

Abstract:

The abstract modifier can be applied to classes and methods. A class that is abstract cannot be instantiated. Abstract classes provide way to defer implementation to sub classes.
A class must be declared abstract if any of the following conditions is true:

- The class has one or more abstract methods.

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

Static:

The static modifier can be applied to class variables, methods, and even a code that is not a part of method. It cannot be assigned to a top-level class definition but it can be assigned to an inner class level definition.

A static class member can be accessed directly by the class name and doesn't need any object. A single copy of a static member is maintained through out the program regardless of, the number of objects created.

Static variables are initialized only once and at the start of the execution during the lifetime of a class. These variables will be initialized first before the initialization of any instance variables.

Methods declared as static (class methods) have several restrictions:
- They can only call other static methods.
- They must only access static data. Attempt to access non-static data causes a compiler error.
- Non-static methods can access static data and methods.
- They cannot refer to **this** or **super** in anyway.
- These methods can be accessed using the class name rather than a object reference.
- main() is static because it must be accessible for an application to run, before any instantiation takes place.
- When main() begins, no objects are created, so if you have a member data, you must create an object to access it.

StaticExample.java

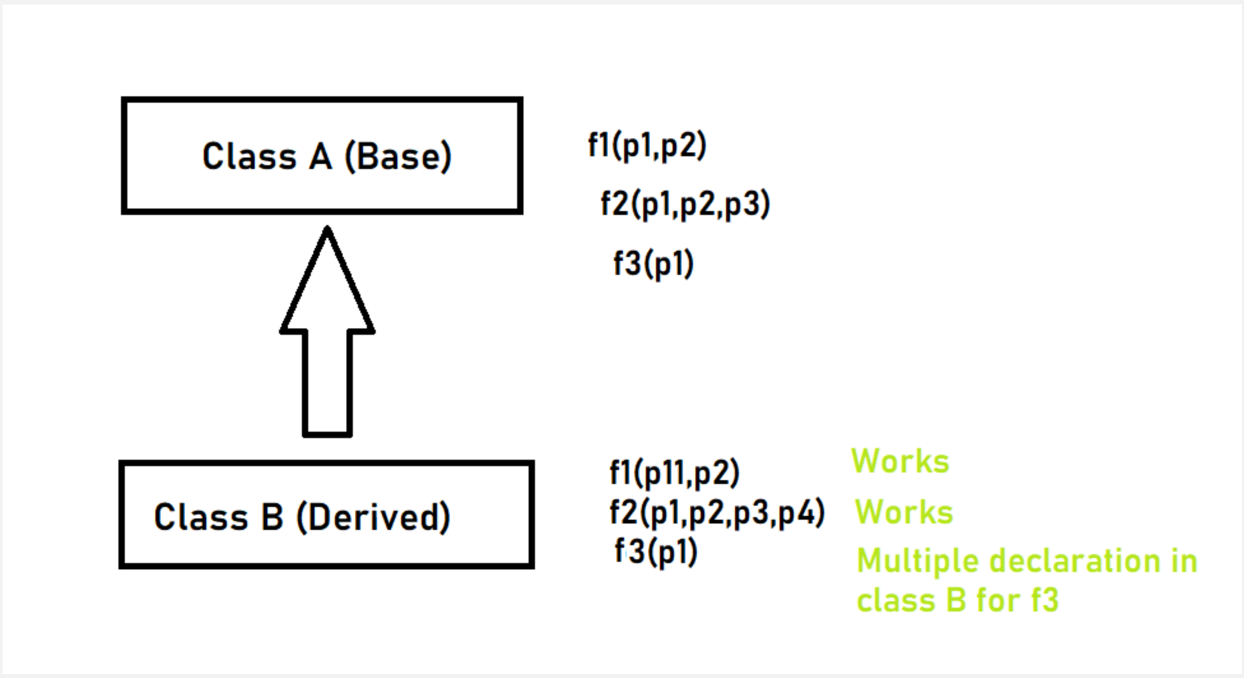
```
// STATIC INITIALIZERS
class StaticExample
{       static int x =8;
        static float y;
        double z;
        void display()
        {       System.out.println("Infosys");
        }
        static void inf(int i)
        {       System.out.println("i = "+i);
                System.out.println("x = "+x);
                System.out.println("y = "+y);
        }       static
        {       System.out.println("Initialising the Static Block");
```

```
          y=x*2;
     }
     public static void main(String args[])
     {    StaticExample se=new StaticExample();
          inf(25);
          se.display();
     }
}
```

Overloading and Overriding

Overloading

In inheritance, if base class has some function which has the same name as any other function in the derived class. Then they must differ in parameters in terms of types and number.

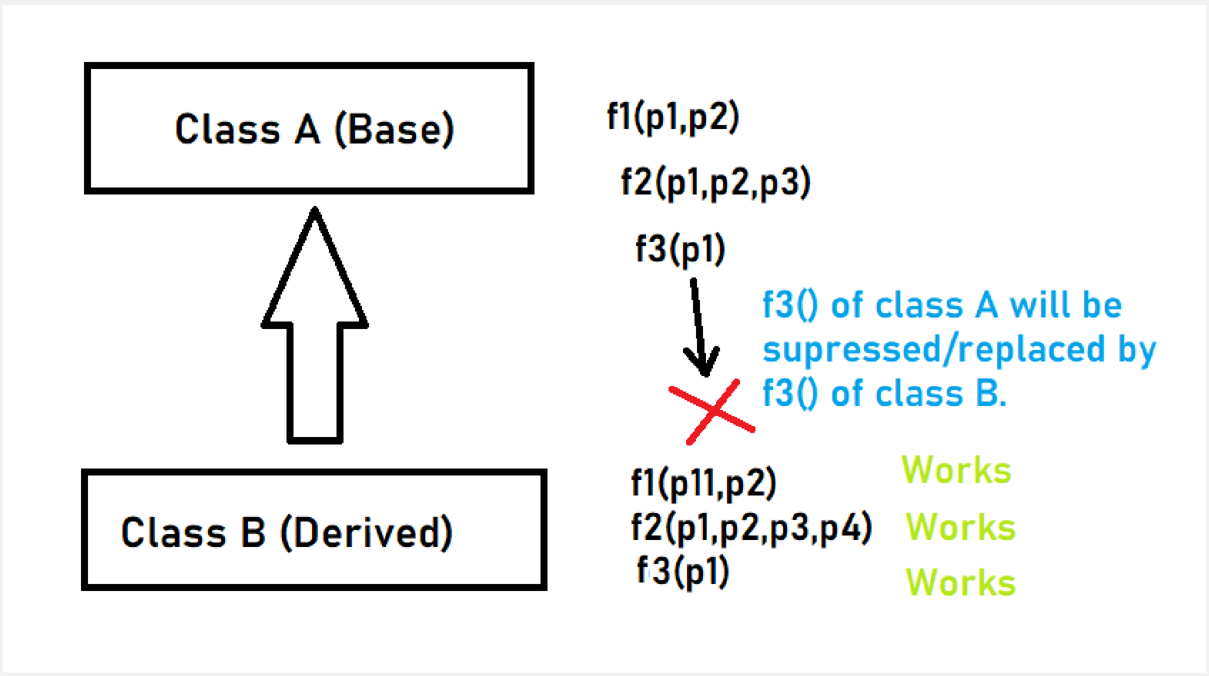


Here in class B there are 3 functions f1,f2,f3 having the same name as in it's base class A. f1 and f2 are called overloaded and will work fine as f1 has parameter of different type (p11) and f2 has 4 parameters whereas f2 of class A has 3 parameters. But, f3 will give the error of multiple declaration in class B for f3() because simply java can not differentiate them as they have name as well as parameters same, so this ambiguity will cause this error.

When we call f2 with 4 values it will call class B version of f2 whereas in case of 3 parameters it will call the class A version of f2 function present in class B (came via inheritance).

Overriding

Java was the first programming language which introduced the concept of Overriding. To continue with the ambiguity problem occurring in overloading mentioned above, if two functions in base and derived classes respectively are having the same name as well as same number and type of parameter, in overriding, base class version of function f3 will be replaced by class B version of f3 in class B. Make no mistake, f3 will be there in class A as it is. Only in class B it will not be there, that means if you call f3 with the object of class B then f3 of class B will be called, not the of class A. So above diagram will be:



Summary:

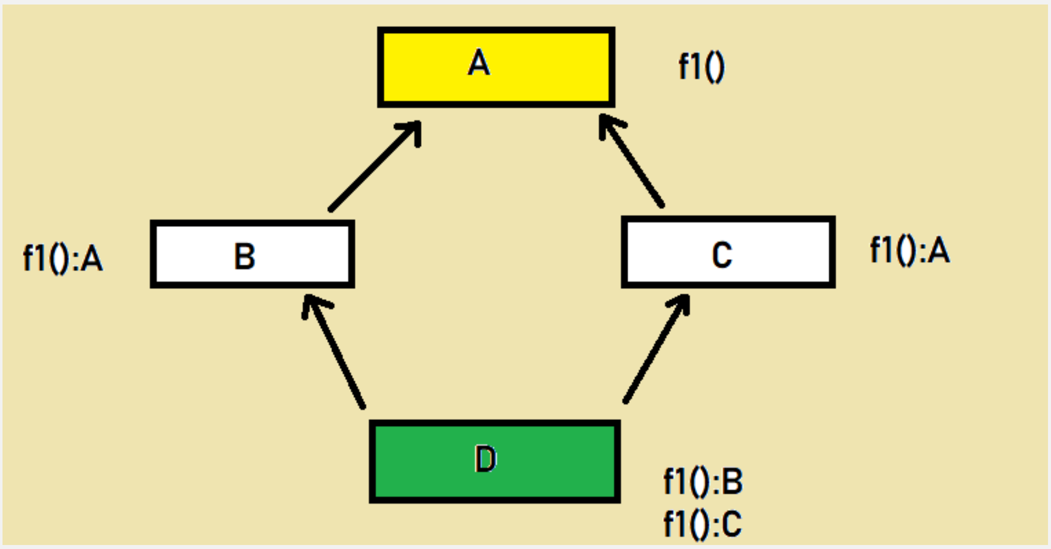
OverLoading	OverRiding
It is the relationship between the methods available in the same class.	It is the relationship between a superclass method and a subclass method.
It does not block inheritance from the superclass.	It blocks inheritance from the superclass.

In overloading separate methods share the same name.	In overriding subclass method replaces the superclass method.
It follows different method signatures.	It follows same method signature.
It may have different return types.	It must have matching return types.
It may have different declared exceptions.	It must have compatible declared exceptions.

Why Java does not support Multiple inheritance?

This can be answered as, to avoid the multiple declaration error of functions in the derived class.

As we know that some of the java features are bug free and simple language. Accordingly, java avoids the error and compile time which might occur in runtime. For example, consider the below scenario:



Here, if class A is inherited by B & C, the function f1() will be there in both the classes. Now, if another class D inherits both B & C classes, which is multiple inheritance, will have two copies of f1(); one from B class and one from C class. To accommodate this, in C++ we use virtual keyword, in runtime polymorphism, which will decide which of these two copies to be used in class D. But, for complex flow in large application, system might behave unpredictably. As java is a simple and bug-free language, it keeps avoiding such situations and hence prohibits such implementation from its root.

If you want to implement this hierarchy you can alternatively use interface, which is another implementation of what we called abstract class. That means if there is a situation when one class must acquire the properties of more than one class, then design your system in such a way that all of them except one class, are interfaces.

Interface in Java

Java is the first programming language that introduced the concept of interface. Interface can be considered as only declaration or blue-print of a class, it does not have any definition of member functions, just their declaration. (Declaration means only name, parameter list of a function without it's body or empty body) this way we can achieve abstraction.

As discussed, we can also achieve the functionality of multiple inheritance with eliminating the possibility of multiple declaration error. It can be used to achieve loose coupling while maintaining the uniformity throughout the application. Interfaces resides in packages in java.

It is declared by using the 'interface' keyword. All the member fields are public, static and final by default. It is compulsory for a class that implements an interface to implement all the methods declared in the interface.

Syntax :

```
interface <interface_name>{

 }
```

How to use an interface

We must use the keyword implements to the class in which we want to use the interface. Example:

```
Interface myinterface

{

//put all fields including function declaration
}

//Class definition of A

Class A implements myinterface

{

//define all methods of myinterface

}
```

It becomes mandatory to use interface if one class has to acquire the properties of more than one class like in multithreading to be done on an applet, you have to extend both Applet and Thread class, but it is not possible, then you have to adopt an alternative approach by implementing the Runnable interface to the class while extending the Applet class for the same.

Handling the Strings

Strings Character array is known as string. In java strings are either treated as the character arrays or as the objects of String, StringBuffer, and StingBuilder classes.

1. Strings as the character arrays-

```
Char arr[]={‘a’,’s’,’d’,’f’,’g’,’h’};
```

2. String as an Object-

There are String, StringBuffer, StringBuilder classes in java, we can create Objects of these classes and can store the multiple characters in it, which are to be formed the string. The benefit of this, is that now we have rich library of predefine functions in these classes that can perform many string operations such as reverting, concatenation, trimming etc. on those string objects we created using these classes. Otherwise we have to write entire logic for these operations. So proceeding further we would adopt the approach of String objects.

```
String s=new String(“Johnny”);
```

StringBuilder d=new StringBuilder(“Johnny”);

StringBuffer k=new StringBuffer(“Johnny”);

Description of various string methods:

String Methods with Description

1	char charAt(int index) Returns the character at the specified index.
2	int compareTo(Object o) Compares this String to another Object.
3	int compareTo(String anotherString) Compares two strings lexicographically.
4	int compareToIgnoreCase(String str) Compares two strings lexicographically, ignoring case differences.
5	String concat(String str) Concatenates the specified string to the end of this string.
6	boolean contentEquals(StringBuffer sb) Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	static String copyValueOf(char[] data) Returns a String that represents the character sequence in the array specified.
8	static String copyValueOf(char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
9	boolean endsWith(String suffix) Tests if this string ends with the specified suffix.
10	boolean equals(Object anObject) Compares this string to the specified object.
11	boolean equalsIgnoreCase(String anotherString) Compares this String to another String, ignoring case considerations.
12	byte getBytes() Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
13	byte[] getBytes(String charsetName Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.

14 void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
Copies characters from this string into the destination character array.

15 int hashCode()
Returns a hash code for this string.

16 int indexOf(int ch)
Returns the index within this string of the first occurrence of the specified character.

17 int indexOf(int ch, int fromIndex)
Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

18 int indexOf(String str)
Returns the index within this string of the first occurrence of the specified substring.

19 int indexOf(String str, int fromIndex)
Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

20 String intern()
Returns a canonical representation for the string object.

21 int lastIndexOf(int ch)
Returns the index within this string of the last occurrence of the specified character.

22 int lastIndexOf(int ch, int fromIndex)
Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.

23 int lastIndexOf(String str)
Returns the index within this string of the rightmost occurrence of the specified substring.

24 int lastIndexOf(String str, int fromIndex)
Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

25 int length()
Returns the length of this string.

26 boolean matches(String regex)
Tells whether or not this string matches the given regular expression.

27 boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
Tests if two string regions are equal.

28 boolean regionMatches(int toffset, String other, int ooffset, int len)
Tests if two string regions are equal.

29 String replace(char oldChar, char newChar)
Returns a new string resulting from replacing all occurrences of oldChar in this string with

newChar.

30 String replaceAll(String regex, String replacement
Replaces each substring of this string that matches the given regular expression with the given replacement.

31 String replaceFirst(String regex, String replacement)
Replaces the first substring of this string that matches the given regular expression with the given replacement.

32 String[] split(String regex)
Splits this string around matches of the given regular expression.

33 String[] split(String regex, int limit)
Splits this string around matches of the given regular expression.

34 boolean startsWith(String prefix)
Tests if this string starts with the specified prefix.

35 boolean startsWith(String prefix, int toffset)
Tests if this string starts with the specified prefix beginning a specified index.

36 CharSequence subSequence(int beginIndex, int endIndex)
Returns a new character sequence that is a subsequence of this sequence.

37 String substring(int beginIndex)
Returns a new string that is a substring of this string.

38 String substring(int beginIndex, int endIndex)
Returns a new string that is a substring of this string.

39 char[] toCharArray()
Converts this string to a new character array.

40 String toLowerCase()
Converts all of the characters in this String to lower case using the rules of the default locale.

41 String toLowerCase(Locale locale)
Converts all of the characters in this String to lower case using the rules of the given Locale.

42 String toString()
This object (which is already a string!) is itself returned.

43 String toUpperCase()
Converts all of the characters in this String to upper case using the rules of the default locale.

44 String toUpperCase(Locale locale)
Converts all of the characters in this String to upper case using the rules of the given Locale.

45 String trim()
Returns a copy of the string, with leading and trailing whitespace omitted.

46	<u>static String valueOf(primitive data type x)</u> Returns the string representation of the passed data type argument.

Write programs to : print all the elements of a string using while loop.

Find a character in a string. Print “present” if it is found otherwise print “Not present”.

Find the length of a string. Ex- strlen();

Compare two strings. Ex- strcmp();

Assign all the characters of a string to another. Ex- strcpy();

Concatenate two strings. Ex- strcat();

Revert a string.

Find whether “cre” is present in “icecream”

Java Classes & Packages

A class is a term for a **template that provides data and the functionality,** which will work on that data.

The Java Development Kit comes with a standard set of class (Java Class Library) that implements most of the basic behaviors needed - not only for programming tasks, but also for graphics and networking database etc.

A **Package** is a group of related classes. The class library is organized in to many packages, each containing several classes. The following packages are prominent:

1. **java.lang:** contains classes that form the core of the language, such as String, Math, Integer and thread. It is automatically imported by default into all the user defined Source files.
2. **java.awt:** contains classes that make up the Abstract Window Toolkit (AWT). This package is used for constructing and managing the graphical user interface of the application.
3. **java.net:** Contains classes for performing network related operations and dealing with sockets and uniform resource locators (URLs).
4. **java.io:** Contains classes that deal with file Input/Output.
5. **java.util:** Contains utility classes for tasks, such as random number generation, defining system properties, and using date and calendar related functions. etc.

The source file layout:

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

- Package Declaration (optional – only one)
- Import Statements (optional – any number)
- Class Definitions (Any number – only one public class)

Role of Access specifier can be describe as :

	Private	" " i.e. None Specified	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	No	Yes	Yes	Yes
Same Package non-Subclass	No	Yes	Yes	Yes
Different Package Subclass	No	No	Yes	Yes
Different Package Non-Subclass	No	No	No	Yes

Nested Class

A class written within another class is known as **nested class.** The scope of the nested class is bounded by the scope of its enclosing class. A nested class has access to the members, including private members, of the class in which it is nested. However, the enclosing class does not have access to the members of the nested class. The access to the enclosing class's scope is possible because the inner class actually has a hidden reference to the outer class context.

There are two types of nested classes **static** and **non-static**. If a inner class is static, it must access the members of its enclosing class through an object. That is, it cannot refer to members of its enclosing class directly.

The most important type of nested class is the **non-static inner class.** It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other **non-static** members of the outer class do. Thus an inner class is fully within the scope of its enclosing class.

Nested class example :

```
InnerClassExample.java
class InnerClass
{       int far_x=45;
        void plan()
        {       Inner inner=new Inner();
                inner.display();
        }     class Inner
        {     void display()
                {       System.out.println("Displaying : far_x : "+far_x);
                }
        }
}     class InnerClassExample
{        public static void main(String args[])
        {       InnerClass ic=new InnerClass();
                ic.plan();
        }
}
```

Classes Defined Inside Methods

Java allows a class to be defined inside a method. Any variable, either a local variable or a formal parameter, can be accessed by methods within an inner class, provided that variable is marked **final**. An object created inside a method is likely to outlive the method invocation. Since local variables and method arguments are conventionally destroyed with their method exits, these variables would be invalid for access by inner class methods after the enclosing method exits. By allowing access only to 'final' variables, it becomes possible to copy the values of those variables into the object itself, thereby extending their lifetime.

```
MOuter.java
// CLASSES DEFINED INSIDE METHODS
public class MOuter
{       private int i=10;
        final int j=20;
        public static void main(String args[])
        {       MOuter that=new MOuter();
                that.go((int)(Math.random()*100),(int)(Math.random() * 100));
        }
        public void go(int x, final int y)
        {       int a=x+y;
                final int b=x-y;
                class MInner
                {       public void method()
                        {       // System.out.println(("X : "+x); // Illegal
                                System.out.println("Y : "+y);
                                // System.out.println("a : "+a); // Illegal
                                System.out.println("b : "+b);
                                System.out.println("i : "+i);   //accessible
                                System.out.println("j : "+j);   // accessible
                        }
                }   MInner that=new MInner();
                that.method();
        }
}
```

Anonymous classes

Some classes that you define inside a method do not need a name. A class defined in this way without a name is an **anonymous class**. In fact, they are defined in the place they are constructed.

```
public void amethod()
        {
        theButton.addActionListener(new ActionLIstener()
                {
                public void actionPerformed(ActionEvent ae)
                        {
                        System.out.println("Action performed");
                        }
                }
        );
```

}

An anonymous class cannot have a constructor. Since you do not specify a name for the class, you cannot use the name to specify a constructor.

Inner classes

- The class name can be used only within the defined scope.
- The inner class can also be defined inside a method generally known as anonymous class.
- The inner class can use both static and instance variables of enclosing classes and final local variables of enclosing blocks.
- An inner class can act as an interface implemented by another inner class.
- Inner classes that are declared static automatically become top-level classes.
- Inner classes cannot declare any static members; only top-level classes can declare static members.
- Public, private, static are all legal access modifiers for this inner class.
- Inner classes can extend any class or interface
- They can access all static, final and instance variables.

Object class:

This is one special class, called Object, defined by java. All other classes are subclasses of **object**. That is, object is a superclass of all other classes. This means that a reference variable of type **object** can refer to an object of any other class. Also, since arrays are implemented as classes, a variable of type object can also refer to any array.

Abstract Classes:

Any class containing one or many abstract methods should also be declared abstract. The keyword abstract is used before the keyword class to declare a class as abstract. The abstract class cannot have objects of its own, i.e, it cannot be directly instantiated using the new operator. Abstract constructors and abstract static methods also cannot be declared in a abstract class. Any subclass of an abstract class must either implement all the abstract methods in the superclass, or it should be declared abstract.

Input/Output in Java

A **stream** is a path of communication between the source of some information and its destination. The source and the destination can be a file, the computer's memory, or even from the Internet. Java works with two types of I/O streams:

1. Byte-Oriented streams
2. Character Oriented streams.

But at the low-level everything is byte oriented.

Byte-Oriented Streams

A general stream, which receives and sends information as bytes, is called a byte-oriented stream. At the top of the hierarchy are the two abstract classes: **InputStream and OutputStream** for input and output respectively.

Character Oriented Streams

Streams, which receive and send information as characters are called character oriented streams. The abstract classes Reader and Writer are the base classes for all the character-oriented streams.

Byte Stream Classes and Interfaces examples

Stream Class	**Meaning**
BufferedInputStream	Buffered input stream
BufferedOutputStream	Buffered output stream
ByteArrayInputStream	Input stream that reads from a byte

Stream Class	Meaning
	array
ByteArrayOutputStream	Output stream that writes to a byte array
DataInputStream	An output stream that contains methods for writing the Java Standard data types.

Character Stream Classes and Interfaces

BufferedReader	**Buffered input character stream**
BufferedWriter	Buffered output character stream
CharArrayReader	Input stream that reads from a character array
CharArrayWriter	Output stream that writes to a character array
FileReader	Input stream that reads from a file
FileWriter	Output stream that writes to a file

This will be discussed in detail in Multithreading chapter.

Exception Handling

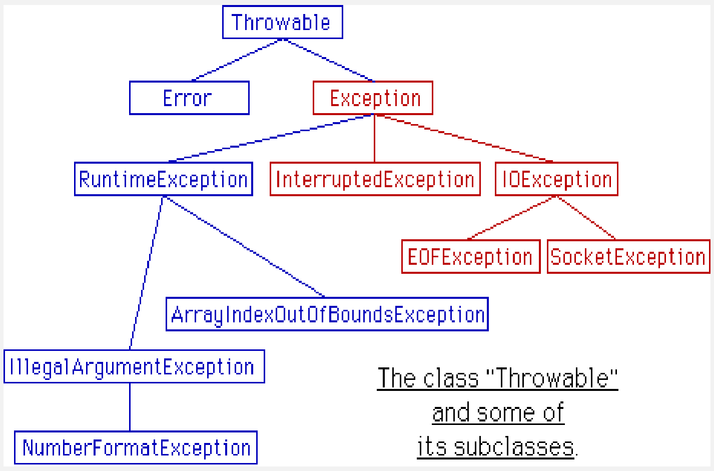
When an exception occurs, for java, it is just an object of any of the Exception classes. And it is handled by calling the appropriate functions by that object. If we go through terminology, then an exception is a run-time error. **Exception Handler** is a set of instructions that handles an exception. Default handler, provided by the Java run time system handles a few exceptions. Exceptions can occur when

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

A few objectives can be pursued when dealing with processing in a program.

- Determine if modifying the existing code will eliminate the problem.
- Inform the User.
- Save the data whenever possible.
- Offer the user a choice of subsequent action.

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



Java exception handling is managed via try, catch, throw, throws, and finally. Program statements that you want to monitor for exceptions are contained within a try block. If an exception occurs within the try block, it is thrown. Your code can catch these exceptions that are automatically thrown by the Java run-time system. To manually throw an exception we use the keyword throw. Any exception that is thrown out of a method must be specified by a throws clause. Any code that absolutely must be executed before a method return is put in a **finally** block. We are not taking example codes here as it is upto the programmer to where to put them, although it is somehow will be covered in socket programming and multithreading.

GUI Programming and Event handling

There are two sets of Java APIs for graphics programming: AWT (Abstract Windowing Toolkit) and Swing.

1. AWT API was introduced in JDK 1.0. Most of the AWT components have become obsolete and should be replaced by newer Swing components.
2. Swing API, a much more comprehensive set of graphics libraries that enhances the AWT, was introduced as part of Java Foundation Classes (JFC) after the release of JDK 1.1. JFC consists of Swing, Java2D, Accessibility, Internationalization, and Pluggable Look-and-Feel Support APIs. JFC was an add-on to JDK 1.1 but has been integrated into core Java since JDK 1.2.

AWT Packages

AWT is huge! It consists of 12 packages (Swing is even bigger, with 18 packages as of JDK 1.7!). Fortunately, only 2 packages - java.awt and java.awt.event - are commonly-used.

1. The java.awt package contains the *core* AWT graphics classes:

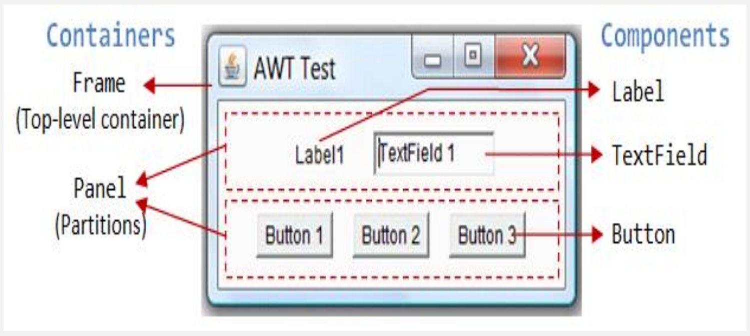
- GUI Component classes (such as Button, TextField, and Label),
- GUI Container classes (such as Frame, Panel, Dialog and ScrollPane),
- Layout managers (such as FlowLayout, BorderLayout and GridLayout),
- Custom graphics classes (such as Graphics, Color and Font).

2. The java.awt.event package supports event handling:

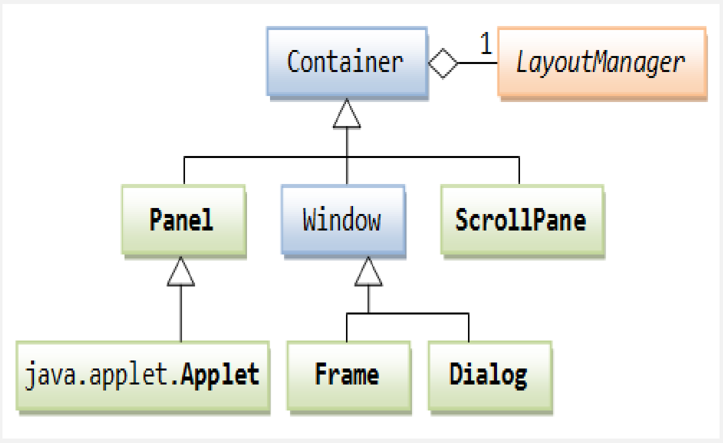
- Event classes (such as ActionEvent, MouseEvent, KeyEvent and WindowEvent),
- Event Listener Interfaces (such as ActionListener, MouseListener, KeyListener and WindowListener),
- Event Listener Adapter classes (such as MouseAdapter, KeyAdapter, and WindowAdapter).

There are two types of GUI elements:

1. **Component**: Components are elementary GUI entities (such as Button, Label, and TextField.)
2. **Container**: Containers (such as Frame, Panel and Applet) are used to hold components in a specific layout (such as flow or grid). A container can also hold sub-containers.

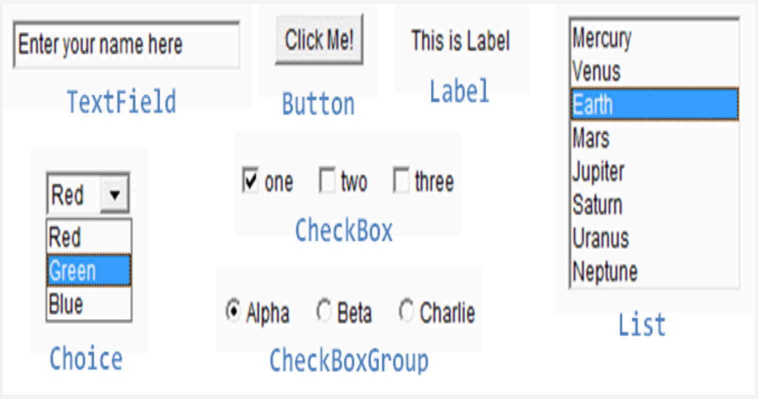


Hierarchy of the AWT Container Classes



AWT Component Classes

AWT provides many ready-made and reusable GUI components. The frequently-used are: Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons), List, and Choice, as illustrated below.



Swings

Swing is the primary Java GUI widget toolkit. It is part of Oracle's Java Foundation Classes (JFC) — an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform.

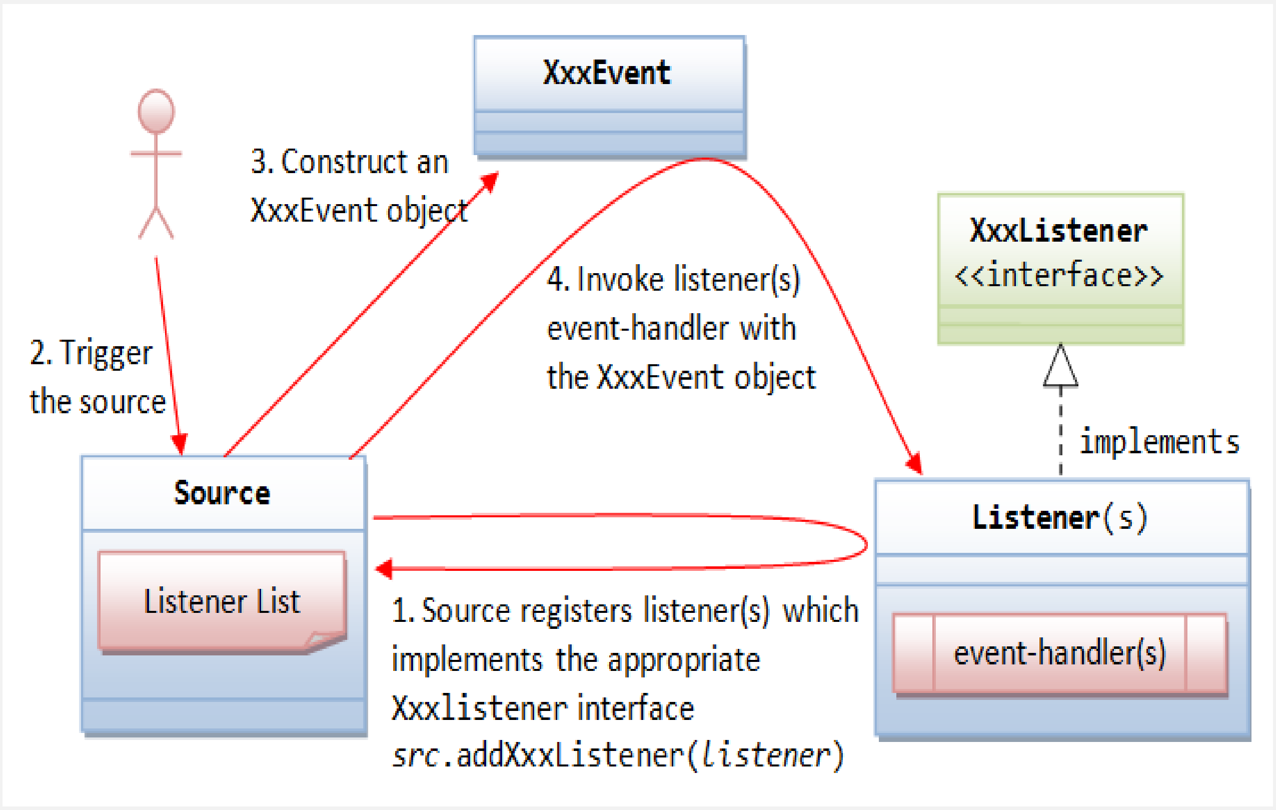
It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

AWT Event-Handling

Java adopts the so-called "Event-Driven" (or "Event-Delegation") programming model for event-handling, similar to most of the visual programming languages (such as Visual Basic and Delphi).

In event-driven programming, a piece of event-handling codes is executed (or called back by the graphics subsystem) when an event has been fired in response to an user input (such as clicking a mouse button or hitting the ENTER key). This is unlike the procedural model, where codes are executed in a sequential manner.

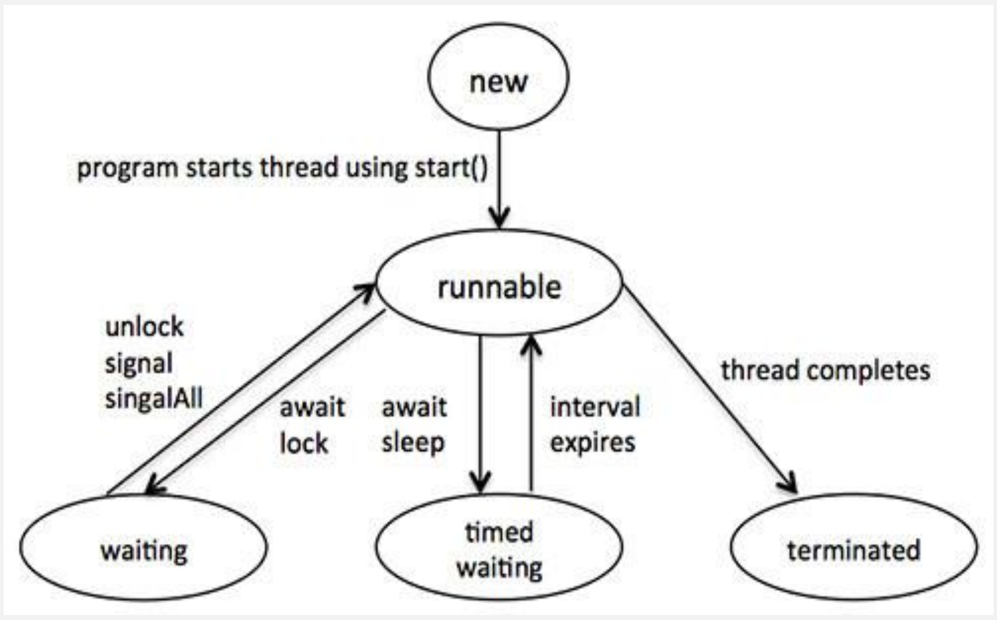
- The AWT's event-handling classes are kept in package java.awt.event.
- Three objects are involved in the event-handling: a *source*, *listener*(s) and an *event* object.
- The source object (such as Button and Textfield) interacts with the user. Upon triggered, it creates an event object


XxxEvent
3. Construct an XxxEvent object
4. Invoke listener(s) event-handler with the XxxEvent object
XxxListener <<interface>>
implements
2. Trigger the source
Source
Listener List
Listener(s)
event-handler(s)
1. Source registers listener(s) which implements the appropriate Xxxlistener interface
src.addXxxListener(listener)

Multithreading in Java

Java is a multithreaded programming language which means we can develop multithreaded program using Java. A multithreaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.



Above-mentioned stages are explained here:

- **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

- **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

- **Waiting:** Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

- **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

- **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Create Thread by Implementing Runnable Interface:

You will need to follow three basic steps:

Step 1

As a first step you need to implement a run() method provided by **Runnable** interface. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run() method:
public void run( );

Step 2

At second step you will instantiate a **Thread** object using the following constructors:
1. Thread(Runnable threadObj)
2. Thread(Runnable threadObj, String threadName);

Where, *threadObj* is an instance of a class that implements the **Runnable** interface and **threadName** is the name given to the new thread.

Step 3

Once Thread object is created, you can start it by calling **start( )** method, which executes a call to run( ) method. Following is simple syntax of start() method:

void start( );

Create Thread by Extending Thread Class:

Step 1

You will need to override **run( )** method available in Thread class. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run() method:
public void run( )

Step 2

Once Thread object is created, you can start it by calling **start( )** method, which executes a call to run( ) method. Following is simple syntax of start() method:
void start( );

Thread Methods:

N	Methods with Description
1	**public void start()** Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	**public void run()** If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	**public final void setName(String name)** Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**public final void join(long millisec)** The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.

7 **public void interrupt()**
Interrupts this thread, causing it to continue execution if it was blocked for any reason.

8 **public final boolean isAlive()**
Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

The following methods in the Thread class are static.

N Methods with Description

1 **public static void yield()**
Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled.

2 **public static void sleep(long millisec)**
Causes the currently running thread to block for at least the specified number of milliseconds.

3 **public static boolean holdsLock(Object x)**
Returns true if the current thread holds the lock on the given Object.

4 **public static Thread currentThread()**
Returns a reference to the currently running thread, which is the thread that invokes this method.

5 **public static void dumpStack()**
Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

Socket programming

Networking in Java is implemented usually with server-client architecture using the TCP/IP model.

Socket is the combination of IP address and port address. Both in combined ensure the end-to-end delivery in the network. IP is used to uniquely identifies a computer system on a network whereas port addressing identifies a process among the processes running in a system.

Socket and ServerSocket are the two classes used in networking that are present in **java.net** package. The steps for networking can be described as follows-

1. Server open its socket While creating an object of ServerSocket class, the constructor of which accepts int value specifies its communicating port.

2. At the client side, when object of Socket class is created by the constructor of Socket class, that accepts IP address as string and port as an int value, sends a request to the server for connection establish.

3. The accept() method at the server side, blocks the execution of server process and waits until the client request arrives. When a request arrives, it unblocks the execution and send a response back to the server. Also returns the reference of request that can be stored at Socket object, created at the server side.

4. When the response arrives at client, the constructor that sent the request, finally creates the object of Socket.

5. After that, at both the sides, Objects of DataInputStream and DataOutputStream are created by the input and output streams retrieved

The constructor of a Socket class accepts IP address (as a string), to request to the server.

Now in the below example server-client connection and messaging will take place, basically here two java programs are communicating with each other which are running in two different machines with different sockets.

Server-side Java program –

```
import java.net.*;
import java.io.*;

public class DemoServer {

    public static void main(String[] args) {

      String sq="quit";
    try
    {
        ServerSocket ssock=new ServerSocket(6303);
        Socket s=ssock.accept();
        DataInputStream inp=new DataInputStream(s.getInputStream());
        DataOutputStream oup=new DataOutputStream(s.getOutputStream());
        System.out.println("Connection established");
        oup.writeUTF("Hello from server");
        for(;;)
        {
           String line= inp.readUTF();
           if(line.equals(sq))
           {
               System.out.println("Quiting");
           return;
           }
           else
                  System.out.println(line);
         }

        }
    catch(Exception e)
    {
        System.out.println("Exiting");
    }
    }
}
```

Client-side Java program –

```
import java.net.*;
import java.io.*;
import java.util.*;
```

```
public class DemoClient {

    public static void main(String[] args) {
        String sq="quit";
        Scanner s=new Scanner(System.in);
        try
        {
        Socket sock=new Socket("localhost",6303);
        DataInputStream inp=new DataInputStream(sock.getInputStream());
        DataOutputStream oup=new DataOutputStream(sock.getOutputStream());
        System.out.println("Connection Established");
        for(;;)
        {
            String line=s.nextLine();
            if(line.equals(sq))
           {
           return;
           }
            else
            {
                oup.writeUTF(line);
            }
        }

        }
        catch(Exception e)
        {
            System.out.println("Exiting");
        }
    }
}
```

Consider the below program to implement multithreading with Socket programming to create a multithreaded server:

```
import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;

public class MultiThreadServer implements Runnable {
   Socket csocket;
   MultiThreadServer(Socket csocket) {
      this.csocket = csocket;
   }

   public static void main(String args[])
   throws Exception {
      ServerSocket ssock = new ServerSocket(1234);
      System.out.println("Listening");
      while (true) {
         Socket sock = ssock.accept();
         System.out.println("Connected");
         new Thread(new MultiThreadServer(sock)).start();
      }
   }
   public void run() {
      try {
         PrintStream pstream = new PrintStream(csocket.getOutputStream());
         for (int i = 100; i >= 0; i--) {
            pstream.println(i +" bottles of beer on the wall");
         }
         pstream.close();
         csocket.close();
      }
      catch (IOException e) {
         System.out.println(e);
      }
   }
}
```

Applets and Servlet

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following:

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet

Four methods in the Applet class give you the framework on which you build any serious applet:

- **init:** This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.

- **start:** This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.

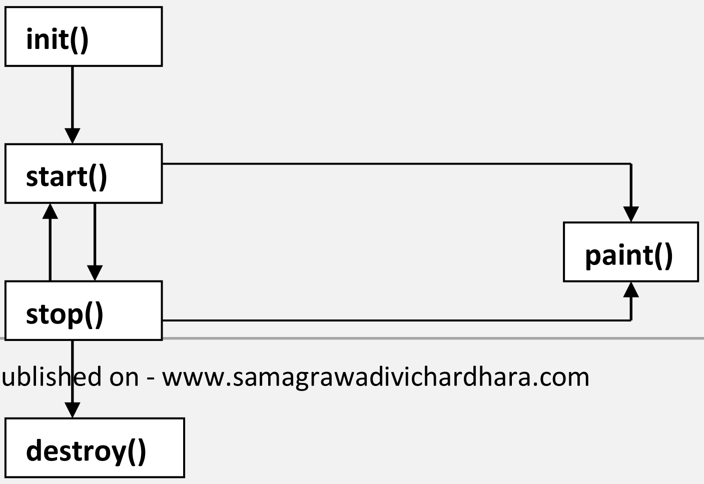
- **stop:** This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

-

- **destroy:** This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

- **paint:** Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

  Refer the below diagram :



Example of applet program

```
class SampleBanner extends Applet
implements Runnable{
   String str = "This is a simple Banner ";
   Thread t ;
   boolean b;
   public void init() {
      setBackground(Color.gray);
      setForeground(Color.yellow);
   }
   public void start() {
      t = new Thread(this);
      b = false;
      t.start();
   }
   public void run () {
      char ch;
      for( ; ; ) {
      try {
         repaint();
         Thread.sleep(250);
         ch = str.charAt(0);
         str = str.substring(1, str.length());
         str = str + ch;
      }
      catch(InterruptedException e) {}
      }
   }
   public void paint(Graphics g) {
      g.drawRect(1,1,300,150);
      g.setColor(Color.yellow);
      g.fillRect(1,1,300,150);
      g.setColor(Color.red);
      g.drawString(str, 1, 150);
   }
}
```

Servlet

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

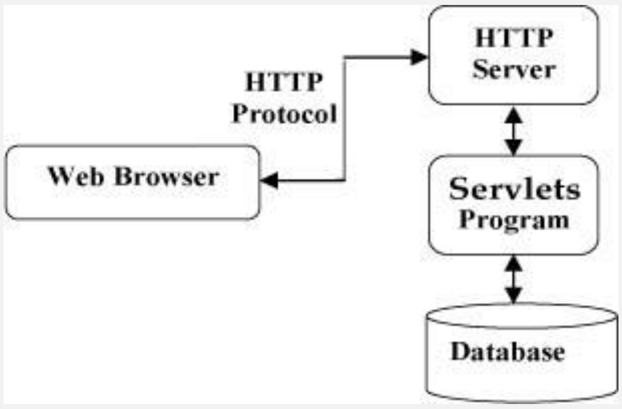
- Performance is significantly better.

- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.

- Servlets are platform-independent because they are written in Java.

- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.

- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.



Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Servlet life cycle

It can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

- The servlet is initialized by calling the **init ()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.

Finally, servlet is garbage collected by the garbage collector of the JVM.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
Public void doGet(HttpServletRequest request , HttpServletResponse
response) throws ServletException, IOException
{
// Servlet code
}
```

The doPost() Method

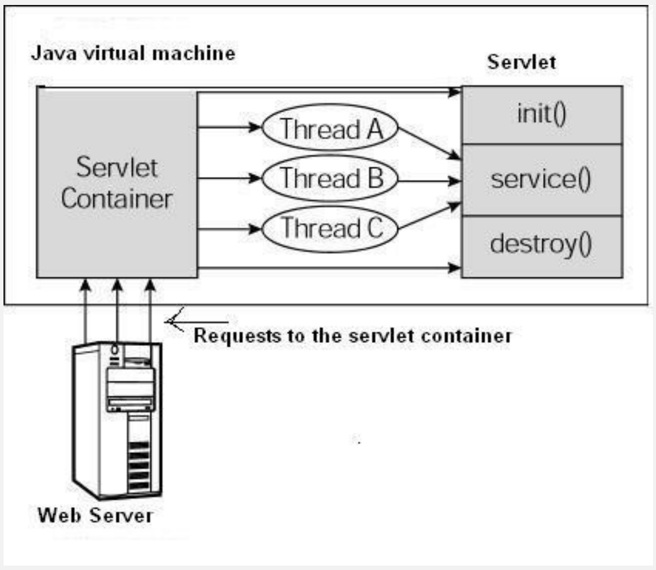
A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled
by doPost() method.

```
Public void doPost( HttpServletRequest request, HttpServletResponse response) throws
ServletException,

IOException
{
// Servlet code
}
```

Simple Program to print Hello World by servlet

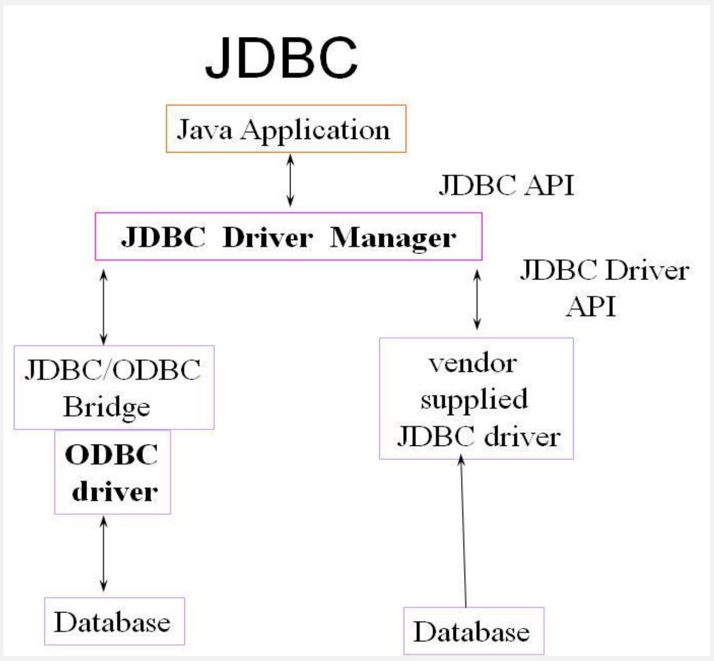
```
// Import required java libraries
Import java.io.*;
Import javax.servlet.*;
Import javax.servlet.http.*;
// Extend HttpServlet class
Publicclass HelloWorld extends HttpServlet
{
Private String message;
Publicvoid init() throws ServletException
{
// Do required initialization
message="Hello World";
}
Publicvoid doGet(HttpServletRequest request
, HttpServletResponse response) throws ServletException
,IOException
{
// Set response content type
response
.
setContentType("text/html");
// Actual logic goes here.
PrintWriter out=response.getWriter();
out.println("<h1>"+message+"</h1>");
}
Public void destroy()
{
// do nothing.
}
```

Servlet Working



Database Connectivity in JAVA

JDBC is a Java-based data access technology (Java Standard Edition platform) from Oracle Corporation. This technology is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases. A JDBC-to-ODBC bridge enables connections to any ODBC-accessible data source in the JVM host environment.



The database programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps:

- **Import JDBC Packages:** Add **import** statements to your Java program to import required classes in your Java code.

```
import java.sql.* ;  // for standard JDBC programs
```

- **Register JDBC Driver:** This step causes the JVM to load the desired driver implementation into memory so it can fulfill your JDBC requests.

Approach (I) - Class.forName():

The most common approach to register a driver is to use Java's **Class.forName()** method to dynamically load the driver's class file into memory, which automatically registers it. This method is preferable because it allows you to make the driver registration configurable and portable.

The following example uses Class.forName( ) to register the Oracle driver:

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

You can use **getInstance()** method to work around noncompliant JVMs, but then you'll have to code for two extra Exceptions as follows:

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
catch(IllegalAccessException ex) {
   System.out.println("Error: access problem while loading!");
   System.exit(2);
catch(InstantiationException ex) {
   System.out.println("Error: unable to instantiate driver!");
   System.exit(3);
}
```

Approach (II) - DriverManager.registerDriver():

The second approach you can use to register a driver is to use the static **DriverManager.registerDriver()** method.

You should use the registerDriver() method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.

The following example uses registerDriver() to register the Oracle driver:

```
try {
   Driver myDriver = new oracle.jdbc.driver.OracleDriver();
   DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

- **Database URL Formulation:** This is to create a properly formatted address that points to the database to which you wish to connect.

We can establish a connection using the **DriverManager.getConnection()** method. For easy reference, let me list the three overloaded DriverManager.getConnection() methods:

- getConnection(String url)
- getConnection(String url, Properties prop)
- getConnection(String url, String user, String password)

Here each form requires a database **URL**. A database URL is an address that points to your database.

Formulating a database URL is where most of the problems associated with establishing a connection occur.

Following table lists down popular JDBC driver names and database URL.

MySQL	com.mysql.jdbc.Driver	**jdbc:mysql://**hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.S	**jdbc:sybase:Tds:**host name: port

	ybDriver	Number/databaseNa me

•

- **Create Connection Object:** Finally, code a call to the DriverManager object's getConnection( ) method to establish actual database connection.

The Next Step-

Creation of Statement Object

The Statement object, which is created using a Connection object, allows us to execute both Data Definition Language (DDL) and Data Manipulation Language (DML) statements by using its execute( ), executeUpdate( ), and executeQuery( ) method to execute any valid SQL statement. The java instruction is-

Statement stmt=con.createStatement();

(Where con is a connection object)

Finally we can execute query like-

stmt.execute("select 'Hello '||USER from dual");

Select the execute method that best suits your needs:

boolean execute(String SQL)

Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.

int executeUpdate(String SQL)

Returns the numbers of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected -- for example, an INSERT, UPDATE, or DELETE statement.

ResultSet executeQuery(String SQL)

Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

The Robot Class

This class is used to generate native system input events for the purposes of test automation, self-running demos, and other applications where control of the mouse and keyboard is needed. The primary purpose of Robot is to facilitate automated testing of Java platform implementations.

Using the class to generate input events differs from posting events to the AWT event queue or AWT components in that the events are generated in the platform's native input queue. For example, Robot.mouseMove will actually move the mouse cursor instead of just generating mouse move events.

Note that some platforms require special privileges or extensions to access low-level input control. If the current platform configuration does not allow input control, an AWTException will be thrown when trying to construct Robot objects. For example, X-Window systems will throw the exception if the XTEST 2.2 standard extension is not supported (or not enabled) by the X server.

Applications that use Robot for purposes other than self-testing should handle these error conditions gracefully.
