Core Java Programming

Introduction to JAVA

Java is related to C++, which is a direct descendant of C. Much of the character of Java is inherited from these two languages. From C, Java derives its syntax. Many of Java's object-oriented features were influenced by C++. In fact, several of Java's defining characteristics come from—or are responses to—its predecessors. As you will see, each innovation in language design was driven by the need to solve a fundamental problem that the preceding languages could not solve. Java is the outcome of the same terminology. Which was initiated by James Gosling and released in 1995 developed by Sun Microsystems as Java 1.0,

Java programming language was originally developed by Sun Microsystems which was later acquired by the Oracle Corporation,

On November 13, 2006, Sun Microsystems made the bulk of its implementation of Java available under the GNU General Public License (GPL).

As of December 2008, the latest release of the Java Standard Edition is 6 (J2SE).
The latest version only supported version as of 2016.

It provides a system for developing application software and deploying it in a cross-platform computing environment. Java is used in a wide variety of computing platforms from embedded devices and mobile phones to enterprise servers and supercomputers. While they are less common than standalone Java applications, Java applets run in secure, sandboxed environments to provide many features of native applications and can be embedded in HTML pages is Java 8, the.

There are 3 levels of Java.

a. J2SE - Java 2 Standard Edition (Core Java)
b. J2EE - Java 2 Enterprise Edition (Adv. Ent. Java)
b. J2ME - Java 2 Micro Edition (java for mobile and hand held devices)

Java characteristics & features

11 Features of Java Programming Language

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

Simple

According to Sun, Java language is simple because:

- Most of the concepts and Syntax is based on C++ thus simple for programmers to learn it after C++.
- Java is Easy to write and more readable and eye catching.
- It removed many confusing and / or rarely used features e.g., explicit pointers, operator overloading, etc.
- Java has a concise, cohesive set of features that makes it easy to learn and use.
- No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

Object-oriented

- Java programming is object-oriented programming language.
- Like C++ java provides most of the object oriented features.
- Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Object-oriented programming (OOPs) is a methodology that simplify software development and maintenance by providing some rules.
- Java is pure OOP. Language. (while C++ is semi object oriented)

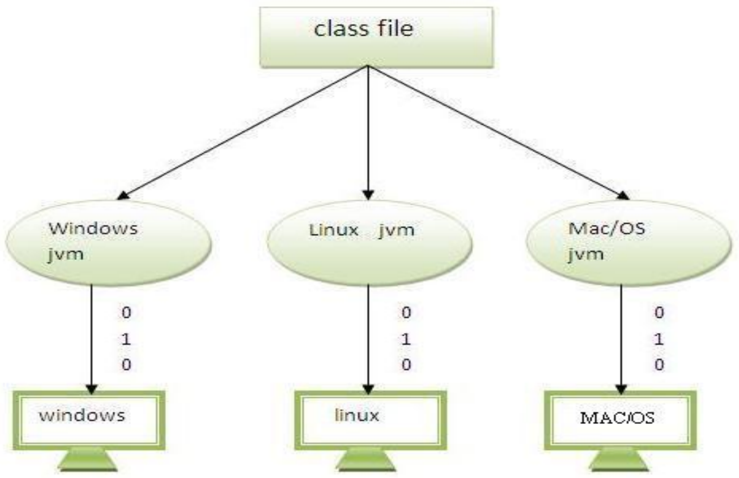
Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent

A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

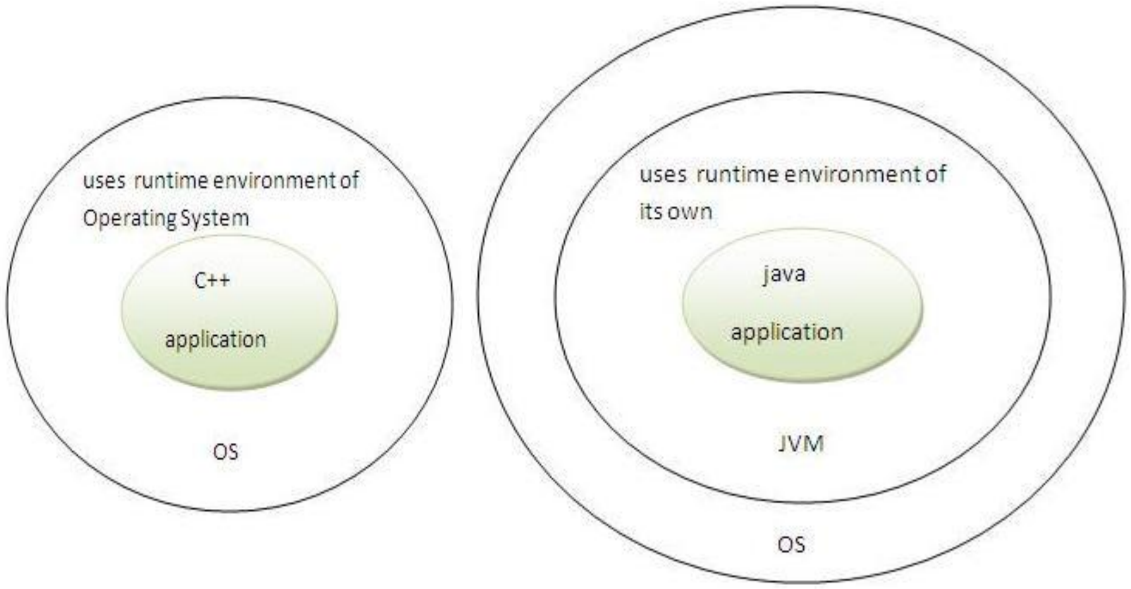


Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform independent code because it can be run on multiple platforms i.e. the main concept is “Write Once and Run Any Where (WORA)”.

Secured

Java is secured because:

- No explicit pointer
- Programs run inside virtual machine sandbox.


uses runtime environment of Operating System
C++ application
OS
uses runtime environment of its own
java application
JVM
OS

- **Classloader-** adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier-** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager-** determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL, JAAS, cryptography etc.

Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

Architecture-neutral

There is no implementation dependent features e.g. size of primitive types is set.

Portable

We may carry the java bytecode to any platform.

High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

JDK (Java Development Kit)

JDK contains everything that will be required to ***develop and run*** Java application.
- Java Development kit is required to compile, develop and debug a java program
- It should be installed on programmer's or developer's computer.

JRE (Java Run time Environment)

JRE contains everything required to ***run*** Java application which has already been compiled. It doesn't contain the code library required to develop Java application.

JVM (Java Virtual Machine)

JVM is a virtual machine which work on top of your operating system to provide a recommended environment for your compiled Java code. JVM only works with bytecode. Hence you need to compile your Java application(.java) so that it can be converted to bytecode format (also known as the .class file). Which then will be used by JVM to run application. JVM only provide the environment It needs the Java code library to run applications.
JVM is a Java component that runs a Java program
- The JVM is part of all Operating Systems.
- It is already prtesent in all computers.

Structure of a java progarm

```
class SampleProgram
{
    public static void main(String args[])
    {
            System.out.println("Hello world, from Java");
    }
}
```

SampleProgram.java

1. A Java program must be written in a class
   Save the program by classname and .java Extension.

2. The class name is user defined and created by the user it begins with Capital letter

3. main() function is always written inside the class. It is the entry point function like C.
4. public static void main(String args[]) must be written in same order.

5. A program can be written inside main()

6. System.out.println() prints output on the screen, same as the printf() in C language.

7. The program can be written in any text editor like notepad and must be saved by the name of class and .java extension

8. The program is compiled on the dos prompt by javac compiler

```
d:\eveningj java>javac SampleProgram.java
```

9. If there are no errors then a .class file is created
ex. SampleProgram.class (Byte Code)

10. Finally the program can be run by java interpreter

```
d:\eveningj java>java SampleProgram
```

Data Types or Types:

The Java programming language has many built in data types. These are classified into two broad categories

1. Primitive Types
2. Reference Data Type

Primitive types are simple values and not objects. reference types are used for more complex types, includeing all types that we declare ourselves.

The Primitive Types

Java defines eight *primitive* types of data: **byte**, **short**, **int**, **long**, **char**, **float**, **double**, and **boolean**. The primitive types are also commonly referred to as *simple* types.

These can be put in four groups:

• **Integers** This group includes **byte**, **short**, **int**, and **long**, which are for whole-valued signed numbers. The most commonly used integer type is **int**. Variables of type **int** are often employed to
control loops, to index arrays, and to perform general-purpose integer math.

• **Floating-point numbers** This group includes **float** and **double**, which represent numbers with fractional precision.
There are two kinds of floating-point types, **float** and **double**, which represent single- and double-precision numbers, respectively. Type **float** is 32 bits wide and type **double**
is 64 bits wide.

Ex.

```
// Compute the area of a circle.
class Area {
public static void main(String args[]) {
double pi, r, a;
r = 10.8; // radius of circle
pi = 3.1416; // pi, approximately
a = pi * r * r; // compute area
System.out.println("Area of circle is " + a);
}
}
```

• **Characters :** This group includes **char**, which represents symbols in a character set, like letters and numbers. In Java, characters are not 8-bit quantities like they are in most other computer languages.
A character variable can be assigned a value by enclosing the character in single quotes.
For example, this assigns the variable **ch** the letter X:

```
char ch;
ch = 'X';
```

Ex.

```
// Demonstrate char data type.
class CharDemo {
public static void main(String args[]) {
char ch1, ch2;
ch1 = 88; // code for X
ch2 = 'Y';
System.out.print("ch1 and ch2: ");
System.out.println(ch1 + " " + ch2);
}
}
```

Output:

```
ch1 and ch2: X Y
```

• **Boolean:** This group includes **boolean**, which is a special type for representing true/false values.
Java defines the values true and false using the
reserved words **true** and **false**.
Ex.

```
// Demonstrate boolean values.
class BoolTest {
public static void main(String args[]) {
```

```
boolean b;
b = false;
System.out.println("b is " + b);
b = true;
System.out.println("b is " + b);
// a boolean value can control the if statement
if(b) System.out.println("This is executed.");
b = false;
if(b) System.out.println("This is not executed.");
// outcome of a relational operator is a boolean value
System.out.println("10 > 9 is " + (10 > 9));
}
}
```

Output:

```
b is false
b is true
This is executed.
10 > 9 is true
```

Reference Data Types:

- Reference variables are created using defined constructors of the classes. They are used to access objects.
- These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy,etc.
- Class objects and various types of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to any object of the declared type or any compatible type.

Example: Animal animal = new Animal("giraffe");

Identifiers:

All components in the Java program require names as their identity. The main components are classes, variables and methods and the Names used to identify them are called identifiers.

Rules for using Java Identifiers:

All identifiers should begin with alphabets (both upper case and lower case are allowed e.g. A to Z or a to z) or special characters i.e. currency character ($) or an underscore (_).
After the first character, identifiers can have any combination of characters.
A keyword cannot be used as an identifier.
Most importantly identifiers are case sensitive.

Example: Valid identifiers: age, $salary, _value
Invalid identifiers: 123abc, -salary

Escape Sequences:

A character preceded by a backslash (\) is an *escape sequence* and has special meaning to the compiler. When an escape sequence is encountered in a print statement, the compiler interprets it accordingly.
The following table shows the Java escape sequences:

Escape Sequences

Escape Sequence	**Description**
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.

\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

Naming conventions in java:

Name	**Convention**
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

Keywords / Reserve Words in JAVA

There are some words that you cannot use as object or variable names in a Java program. These words are known as “reserved” words; they are keywords that are already used by the syntax of the Java programming language.

The table below lists all the words that are reserved:

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

For example, if you try and create a new class and name it using a reserved word:

```
 // you can't use finally as it's a reserved word!
 class finally {

    public static void main(String[] args) {

       //class code..

    }
 }
```

It will not compile, instead you will get the following error:
<identifier> expected

Variables

The variable in JAVA program work as the basic unit of storage. A variable is defined by the combination of an identifier, a type, a scope and an optional initializer. The scope of the variable defines their visibility, and a lifetime.

In Java, all variables must be declared before they can be used.

Syntax for declaring variables is:
datatype identifier [= *value*][, *identifier* [= *value*] …];

Here, *type* is one of Java’s atomic types, or the name of a class or interface. The *identifier* is the name of the variable. You can initialize the variable by specifying an equal sign and a value. Keep in mind that the initialization expression must result in a value of the same type as that specified for the variable. To declare more than one variable of the specified type, use a comma-separated list.

Here are several examples of variable declarations of various types. Note that some include an initialization.

```
int a, b, c; // declares three ints, a, b, and c.
int d = 3, e, f = 5; // declares three more ints, initializing
// d and f.
byte z = 22; // initializes z.
double pi = 3.14159; // declares an approximation of pi.
char x = 'x'; // the variable x has the value 'x'.
```

The identifiers that you choose have nothing intrinsic in their names that indicates their type. Java allows any properly formed identifier to have any declared type.

Types and Scope of the Variable

There are three kinds of variables in Java:
• Local variables
• Instance variables
• Class/static variables

Local variables:
• Local variables are declared in methods, constructors, or blocks.
• Local variables are created when the method, constructor or block is entered and the variable will be destroyed
once it exits the method, constructor or block.
• Access modifiers cannot be used for local variables.
Local variables are visible only within the declared method, constructor or block.
• Local variables are implemented at stack level internally.
• There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

Example:
Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to this method only.

```
public class Test{
public void pupAge(){
int age = 0;
age = age + 7;
System.out.println("Puppy age is : " + age);
}
public static void main(String args[]){
Test test = new Test();
test.pupAge();
}
}
```

This would produce the following result:

```
Puppy age is: 7
```

Example:
Following example uses *age* without initializing it, so it would give an error at the time of compilation.

```
public class Test{
public void pupAge(){
int age;
age = age + 7;
System.out.println("Puppy age is : " + age);
}
public static void main(String args[]){
Test test = new Test();
test.pupAge();
}
}
```

This would produce the following error while compiling it:

```
Test.java:4:variable number might not have been initialized
age = age + 7;
^
1 error
```

Instance variables:

• Instance variables are declared in a class, but outside a method, constructor or any block.
When a space is allocated for an object in the heap, a slot for each instance variable value is created.
• Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when
the object is destroyed.
• Instance variables hold values that must be referenced by more than one method, constructor or block, or
essential parts of an object's state that must be present throughout the class.
• Instance variables can be declared in class level before or after use.
• Access modifiers can be given for instance variables.
• The instance variables are visible for all methods, constructors and block in the class. Normally, it is

recommended to make these variables private (access level). However visibility for subclasses can be given for
these variables with the use of access modifiers.
• Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object
references it is null. Values can be assigned during the declaration or within the constructor.
• Instance variables can be accessed directly by calling the variable name inside the class. However within static methods and different class should be called using the fully qualified name . *ObjectReference.VariableName*.

Example:

```
import java.io.*;
public class Employee{
// this instance variable is visible for any child class.
public String name;
// salary variable is visible in Employee class only.
private double salary;
// The name variable is assigned in the constructor.
public Employee (String empName){
name = empName;
}
// The salary variable is assigned a value.
public void setSalary(double empSal){
salary = empSal;
}
// This method prints the employee details.
public void printEmp(){
System.out.println("name : " + name );
System.out.println("salary :" + salary);
}
public static void main(String args[]){
Employee empOne = new Employee("Ransika");
empOne.setSalary(1000);
empOne.printEmp();
}}
```
This would produce the following result:

```
name : Ransika
salary :1000.0
```

Class/static variables:

• Class variables also known as static variables are declared with the *static* keyword in a class, but outside a method, constructor or a block.
• There would only be one copy of each class variable per class, regardless of how many objects are created from it.
• Static variables are rarely used other than being declared as constants. Constants are variables that are
declared as public/private, final and static. Constant variables never change from their initial value.
• Static variables are stored in static memory. It is rare to use static variables other than declared final and used
as either public or private constants.
• Static variables are created when the program starts and destroyed when the program stops.
• Visibility is similar to instance variables. However, most static variables are declared public since they must be
available for users of the class.
• Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and
for object references, it is null. Values can be assigned during the declaration or within the constructor.
Additionally values can be assigned in special static initializer blocks.
• Static variables can be accessed by calling with the class name . *ClassName.VariableName*.
• When declaring class variables as public static final, then variables names (constants) are all in upper case. If
the static variables are not public and final the naming syntax is the same as instance and local variables.

Example:

```
import java.io.*;
public class Employee{
// salary variable is a private static variable
private static double salary;
// DEPARTMENT is a constant
public static final String DEPARTMENT = "Development ";
public static void main(String args[]){
salary = 1000;
System.out.println(DEPARTMENT+"average salary:"+salary);
}
}
```

This would produce the following result:

```
Development average salary:1000
```

Operators

Java provides a rich operator environment. These operators are used to manipulate variables. Most of its operators can be divided into the following four groups: arithmetic, bitwise, relational, and logical. Java also defines some additional operators that handle certain special situations.

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators

Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:
Assume integer variable A holds 10 and variable B holds 20, then:

Operator	**Description**	**Example**
+ Addition	Adds values on either side of the operator A + B	A + B will give 30
- Subtraction	Subtracts right hand operand from left hand operand A - B	A - B will give -10
* Multiplication	Multiplies values on either side of the operator A * B	A * B will give 200
/ Division	Divides left hand operand by right hand operand B / A	B / A will give 2
% Modulus	Divides left hand operand by right hand operand and returns remainder B % A	B % A will give 0

The Relational Operators:

There are following relational operators supported by Java language:
Assume variable A holds 10 and variable B holds 20, then:

Operator	**Description**	**Example**
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Bitwise Operators

Java defines several *bitwise operators* that can be applied to the integer types: **long**, **int**, **short**, **char**, and **byte**. Bitwise operator works on bits and performs bit-by-bit operation. These operators act upon the individual bits of their operands.

Assume if a = 60; and b = 13; now in binary format
they will be as follows:
a = 0011 1100
b = 0000 1101
----
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011

The following table lists the bitwise operators:
Assume integer variable A holds 60 and variable B holds 13, then:

Operators	**Description**	**Example**
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators:

Assume Boolean variables A holds true and variable B holds false, then:

Operators	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

The Assignment Operators

There are following assignment operators supported by Java language:

=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A

%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Conditional Operator (?:):
Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable.
The operator is written as:
Syntax:

```
variable x =(expression)? value iftrue: value iffalse
```

example:

```
public class Test{
public static void main(String args[]){
int a , b;
a =10;
b =(a ==1)?20:30;
System.out.println("Value of b is : "+ b );
b =(a ==10)?20:30;
System.out.println("Value of b is : "+ b );
}
}
```

This would produce the following result:

```
Value of b is:30
Value of b is:20
```

Input data from keyboard

- To enter data from keyboard there are three important methods
- These methods are like scanf() function in C language.
- There are Input classes

1. DataInputStream 2. BufferdReader 3. Scanner

1. DataInputStream

- This class comes from java.io package
- A java package contains a large number of ready to use classes. A package is like header (.h) files of C
- A java package is used in a program by using import statement at the begining of program.

ex.
import java.io.*; -- import all classes of java.io package.
import java.io.DataInputStream;

- The object of DataInputStream class is declared and initialised using System.in property (System.in means keyboard or standard input device)

ex.

DataInputStream in; // variable of a class is called Object
in = new DataInputStream(System.in); // initalize or instantiate System.in is keywboard

DataInputStream in = new DataInputStream(System.in);

- readLine() function can be used to enter data from keyboard.
- The readLine() function inputs any data as a string data.

- The int and float data are converted using special conversion functions.
in.readLine() --- reads data from keyboard only String data.

ex.

1. Integer.parseInt()   --- String to int
2. Long.parseLong()   --- String to long
3. Float.parseFloat()   --- String to float
4. Double.parseDouble() --- String to double

- While entering data from keyboard the java must handle the runtime error or exception
- The statements that input data from keyboard should be written in try and catch block

printStackTrace();

ex.

```
try
{
  ----
  ----
}
catch(Exception ex)
{
    System.out.println("Some error occured " + ex);
    System.exit(0); // close the program
}
----
----
```

- All the input statements are written try block, if any error comes then control goes to catch block and displays an error message
- If NO error occurs then the program continues.

- A java program have many keyboard inputs at various stages of the program. In such cases a throws statement can be used which is applied to a function.
- If the throws statement is used then there is no need of try and catch blocks which reduces program size the program becomes easy to look and understand.

ex.

```
public static void main(String args[]) throws Exception
{

}
```

- HW programs

2. BufferedReader method

- This is a second method to read data from keyboard
- There are 2 classes used in this method

1. BufferedReader   2. InputStreamReader()

- Both classes come from java.io package

ex.

```
BufferdReader in = new BufferedReader(new InputStreamReader(System.in)));
```

- The in.readLine() reads data from keyboard.

3. Scanner

- It is newest method to input data from keyborad
- Scanner class comes from java.util
- This class does not require try and catch or throws exception.

ex.

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

there are many functions of Scanner class

1. next() -- to read string data
2. nextInt() -- read int
3. nextFloat()
4. nextLong()
5. nextDouble()

1. Convert cent to far

f = (c-32)* (5/9)

2. Eintsein formula

e = m * (c * c)

3. Volume of cylinder

vol = 3.14f * (rad * rad) * ht

Control Statements

A programming language uses *control* statements to cause the flow of execution to advance and branch based on changes to the state of a program. Java's program control statements can be put into the following categories: selection, iteration, and jump. *Selection* statements allow your program to choose different paths of execution based upon the outcome of an expression or the state of a variable. *Iteration* statements enable program execution to repeat one or more statements, that is, iteration statements form loops. *Jump* statements allow your program to execute in a nonlinear fashion.

1. **Java's Selection Statements /** Java Decision Making

There are two types of decision making statements in Java. They are:

- if statements
- switch statements

These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

1. if Statement:

The **if** statement is Java's conditional branch statement. It can be used to route program execution through two different paths.

Syntax:

```
if(condition)
{
//Statements will execute if the Boolean expression is true
}
```

Here, each *statement* may be a single statement or a compound statement enclosed in curly braces (that is, a *block*). The *condition* is any expression that returns a **boolean** value. If the Boolean expression evaluates to true, then the block of code inside the if statement will be executed. If not, the first set of code after the end of the if statement (if it exists after the closing curly brace) will be executed.

Some programmers find it convenient to include the curly braces when using the **if**, even when there is only one statement in each clause. This makes it easy to add another statement at a later date, and you don't have to worry about forgetting the braces. In fact, forgetting to define a block when one is needed is a common cause of errors.

2. if...else Statement:

if statement can be followed by an optional *else* statement, which executes when the Boolean expression is false.

Syntax:

The syntax of an if...else is:

```
if(Boolean_expression){
//Executes when the Boolean expression is true
}else{
//Executes when the Boolean expression is false
}
```

3. if...else if...else Statement (Ladder Structure):

An if statement can be followed by an optional *else if...else* statement, which is very useful to test various conditions using single if...else if statement. When using if, else if , else statements there are few points to keep in mind. An if can have zero or one else's and it must come after any else if's. An if can have zero to many else if's and they must come before the else. Once an else if succeeds, none of the remaining else if's or else's will be tested.

Syntax:

The syntax of an if...else is:

```
if(Boolean_expression1){
//Executes when the Boolean expression 1 is true
}elseif(Boolean_expression2){
```

```
//Executes when the Boolean expression 2 is true
}elseif(Boolean_expression3){
//Executes when the Boolean expression 3 is true
}else{
//Executes when the none of the above condition is true.
}
```

The **if** statements are executed from the top down. As soon as one of the conditions controlling the **if** is **true**, the statement associated with that **if** is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final **else** statement will be executed. The final **else** acts as a default condition; that is, if all other conditional tests fail, then the last **else** statement is performed. If there is no final **else** and all other conditions are **false**, then no action will take place.

4. Nested if...else Statement:

It is always legal to nest if-else statements which means you can use one if or else if statement inside another if or else if statement.

Syntax:

The syntax for a nested if...else is as follows:

```
if(Boolean_expression1){
//Executes when the Boolean expression 1 is true
if(Boolean_expression2){
//Executes when the Boolean expression 2 is true
}
}
```

You can nest *else if...else* in the similar way as we have nested *if* statement.

II. The switch Statement:

The **switch** statement is Java's multiway branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression. A *switch* statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax:
The syntax of enhanced for loop is:

```
switch(expression){
case value :
//Statements
break;//optional
case value :
//Statements
break;//optional
//You can have any number of case statements.
default://Optional
//Statements
}
```

Each value specified in the **case** statements must be a unique constant expression (such as a literal value). Duplicate **case** values are not allowed. The type of each value must be compatible with the type of *expression*.

The **switch** statement works like this: The value of the expression is compared with each of the values in the **case** statements. If a match is found, the code sequence following that **case** statement is executed. If none of the constants matches the value of the expression, then the **default** statement is executed. However, the **default** statement is optional. If no **case** matches and no **default** is present, then no further action is taken.

The **break** statement is used inside the **switch** to terminate a statement sequence. When a **break** statement is encountered, execution branches to the first line of code that follows the entire **switch** statement. This has the effect of "jumping out" of the **switch**.

The following rules apply to a switch statement:

- The variable used in a switch statement can only be a byte, short, int, or char.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a *break* statement is reached.
- When a *break* statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will *fall through*to subsequent cases until a break is reached.
- A *switch* statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Nested switch Statements

You can use a **switch** as part of the statement sequence of an outer **switch**. This is called a *nested* **switch**. Since a **switch** statement defines its own block, no conflicts arise between the **case** constants in the inner **switch** and those in the outer **switch**.
For example, the following
fragment is perfectly valid:

```
switch(count) {
case 1:
switch(target) { // nested switch
case 0:
System.out.println("target is zero");
break;
case 1: // no conflicts with outer switch
System.out.println("target is one");
break;
}
break;
case 2: // ...
```

Here, the **case 1:** statement in the inner switch does not conflict with the **case 1:** statement in the outer switch. The **count** variable is compared only with the list of cases at the outer level. If **count** is 1, then **target** is compared with the inner list cases.

2. Iteration Statements

Java's iteration statements are **for**, **while**, and **do-while**. These statements create what we commonly call *loops*. As you probably know, a loop repeatedly executes the same set of instructions until a termination condition is met. As you will see, Java has a loop to fit any programming need.

i) For Loop

The **for** loop operates as follows. When the loop first starts, the *initialization* portion of the loop is executed. Generally, this is an expression that sets the value of the *loop control variable*, which acts as a counter that controls the loop. It is important to understand that the initialization expression is executed only once. Next, *condition* is evaluated. This must be a Boolean expression.

It usually tests the loop control variable against a target value. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates. Next, the *iteration* portion of the loop is executed. This is usually an expression that increments or decrements the loop control variable. The loop then iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass. This process repeats until the controlling expression is false.

Multiple values passing using Comma

There will be times when you will want to include more than one statement in the initialization and iteration portions of the **for** loop. For example, consider the loop in
the following program:

```
class Sample {
public static void main(String args[]) {
```

```
int a, b;
b = 4;
for(a=1; a<b; a++) {
System.out.println("a = " + a);
System.out.println("b = " + b);
b--;
}
}}
```

As you can see, the loop is controlled by the interaction of two variables. Since the loop is governed by two variables, it would be useful if both could be included in the **for** statement, itself, instead of **b** being handled manually. Fortunately, Java provides a way to accomplish this. To allow two or more variables to control a **for** loop, Java permits you to include multiple statements in both the initialization and iteration portions of the **for**. Each statement is separated from the next by a comma. Using the comma, the preceding **for** loop can be more efficiently coded, as shown here:

```
// Using the comma.
class Comma {
public static void main(String args[]) {
int a, b;
for(a=1, b=4; a<b; a++, b--) {
System.out.println("a = " + a);
System.out.println("b = " + b);
}
}}
```

In this example, the initialization portion sets the values of both **a** and **b**. The two commaseparated
statements in the iteration portion are executed each time the loop repeats. The program generates the following output:

```
a = 1
b = 4
a = 2
b = 3
```

Nested for loop:

- Two or more loops written and executed one inside another.
- There is an outer loop and inner loop
- The inner loop completes first and the outer loop continues.

```
 for(i=1;i<=4;i++) // outer loop
   {
      for(j=1;j<=3;j++) // inner loop
      {
          ----
      }
   }
```

- This is an imp. type of loop used in many programs like array sorting, matrix programs, multi domension program and number pyramids.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1

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

The while Loop:

Another of Java’s loops is the **while**. The general form of the **while** loop is while(*condition*) *statement*; where *statement* may be a single statement or a block of statements, and *condition* defines the condition that controls the loop, and it may be any valid Boolean expression. The loop repeats while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

Here is a simple example in which a **while** is used to print the alphabet:

```
// Demonstrate the while loop.
class WhileDemo {
public static void main(String args[]) {
char ch;
// print the alphabet using a while loop
ch = 'a';
while(ch <= 'z') {
System.out.print(ch);
ch++;
}}}
```

Here, **ch** is initialized to the letter a. Each time through the loop, **ch** is output and then incremented. This process continues until **ch** is greater than z. As with the **for** loop, the **while** checks the conditional expression at the top of the loop, which means that the loop code may not execute at all. This eliminates the need for performing a separate test before the loop. The following program illustrates this characteristic of the **while** loop. It computes the integer powers of 2, from 0 to 9.

```
// Compute integer powers of 2.
class Power {
public static void main(String args[]) {
int e;
int result;
for(int i=0; i < 10; i++) {
result = 1;
e = i;
while(e > 0) {
result *= 2;
e--;
}
System.out.println("2 to the " + i +
" power is " + result);
}
}
}
```

The output from the program is shown here:

```
2 to the 0 power is 1
2 to the 1 power is 2
2 to the 2 power is 4
2 to the 3 power is 8
2 to the 4 power is 16
2 to the 5 power is 32
2 to the 6 power is 64
2 to the 7 power is 128
2 to the 8 power is 256
2 to the 9 power is 512
```

do...while **Loops:**

The do loop is just like a while loop, except that do executes a given statement or block until a condition is false. The main difference is that while loops test the condition before looping, making it possible that the body of the loop will never execute if the condition is false the first time it's tested. do loops run the body of the loop at least once before testing the condition.

do loops look like this:

```
do {
bodyOfLoop;
} while (condition);
```

Here, the bodyOfLoop part is the statements that are executed with each iteration. It's shown here with a block statement because it's most commonly used that way, but you can substitute the braces for a single statement as you can with the other control-flow constructs.
The condition is a boolean test. If it returns true, the loop is run again. If it returns false, the loop exits. Keep in mind that with do loops, the body of the loop executes at least once.
Here's a simple example of a do loop that prints a message each time the loop iterates:

```
int x = 1;
do {
System.out.println("Looping, round " + x);
x++;
} while (x <= 10);
```

Here's the output of these statements:

```
Looping, round 1
Looping, round 2
Looping, round 3
Looping, round 4
Looping, round 5
```

```
Looping, round 6
Looping, round 7
Looping, round 8
Looping, round 9
Looping, round 10
```

3. Jump Statements

The **break** statement in Java programming language has the following two usages −

- When the **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- It can be used to terminate a case in the **switch** statement (covered in the next chapter).

The syntax of a break is a single statement inside any loop −

```
break;
```

Example

```
public class Test {
   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
            break;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

This will produce the following result −

Output

```
10
20
```

Continue Statement:

The **continue** keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes control to immediately jump to the update statement.
- In a while loop or do/while loop, control immediately jumps to the Boolean expression.

The syntax of a continue is a single statement inside any loop −

```
continue;
```

Example

```
public class Test {

  public static void main(String args[]) {
    int [] numbers = {10, 20, 30, 40, 50};

    for(int x : numbers ) {
      if( x == 30 ) {
        continue;
      }
      System.out.print( x );
      System.out.print("\n");
    }
  }
}
```

This will produce the following result −

Output

```
10
20
40
50
```

Java Methods

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println method, for example, the system actually executes seveal statements in order to display a message on the console.

Creating a Method:

In general, a method has the following syntax:

```
modifier returnValueType methodName(list of parameters){
// Method body;
}
```

A method definition consists of a method header and a method body. Here are all the parts of a method:

Modifiers: The modifier, which is optional, tells the compiler how to call the method. This defines the access type of the method.

Return Type: A method may return a value. The returnValueType is the data type of the value the method returns. Some methods perform the desired operations without returning a value. In this case, the returnValueType is the keyword **void**.

Method Name: This is the actual name of the method. The method name and the parameter list together constitute the method signature.

Parameters: A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.

Method Body: The method body contains a collection of statements that define what the method does.

Example:

Here is the source code of the above defined method called max(). This method takes two parameters num1 and
num2 and returns the maximum between the two:

```
/** Return the max between two numbers */
public static int max(int num1,int num2){
int result;
if(num1 > num2)
result = num1;
else
result = num2;
return result;
}
```

Calling a Method:

In creating a method, you give a definition of what the method is to do. To use a method, you have to call or invoke it. There are two ways to call a method; the choice is based on whether the method returns a value or not. When a program calls a method, program control is transferred to the called method. A called method returns control to the caller when its return statement is executed or when its method-ending closing brace is reached.
If the method returns a value, a call to the method is usually treated as a value. For example:

```
int larger = max(30,40);
```

If the method returns void, a call to the method must be a statement. For example, the method println returns void.
The following call is a statement:

```
System.out.println("Welcome to Java!");
```

Example:
Following is the example to demonstrate how to define a method and how to call it:

```
public class TestMax{
```

```
/** Main method */
public static void main(String[] args){
int i =5;
int j =2;
int k = max(i, j);
System.out.println("The maximum between "+ i +
" and "+ j +" is "+ k);
}


/** Return the max between two numbers */
public static int max(int num1,int num2){
int result;
if(num1 > num2)
result = num1;
else
result = num2;
return result;
}
}
```

This would produce the following result:

```
The maximum between 5and2is5
```

This program contains the main method and the max method. The main method is just like any other method except that it is invoked by the JVM.
The main method's header is always the same, like the one in this example, with the modifiers public and static, return value type void, method name main, and a parameter of the String[] type. String[] indicates that the parameter is an array of String**.**

Passing Parameters by Values:

When calling a method, you need to provide arguments, which must be given in the same order as their respective parameters in the method specification. This is known as parameter order association.

For example, the following method prints a message n times:

```
public static void nPrintln(String message,int n){
for(int i =0; i < n; i++)
System.out.println(message);
}
```

Here, you can use nPrintln("Hello", 3) to print "Hello" three times. The nPrintln("Hello", 3) statement passes the actual string parameter, "Hello", to the parameter, message; passes 3 to n; and prints "Hello" three times. However, the statement nPrintln(3, "Hello") would be wrong.

When you invoke a method with a parameter, the value of the argument is passed to the parameter. This is referred to as pass-by-value. If the argument is a variable rather than a literal value, the value of the variable is passed to the parameter. The variable is not affected, regardless of the changes made to the parameter inside the method.

For simplicity, Java programmers often say passing an argument x to a parameter y, which actually means passing
the value of x to y.

Example:

Following is a program that demonstrates the effect of passing by value. The program creates a method for
swapping two variables. The swap method is invoked by passing two arguments. Interestingly, the values of the
arguments are not changed after the method is invoked.

```
public class TestPassByValue{
public static void main(String[] args){
int num1 =1;
int num2 =2;
System.out.println("Before swap method, num1 is "+num1 +" and num2 is "+ num2);
// Invoke the swap method
swap(num1, num2);
System.out.println("After swap method, num1 is "+num1 +" and num2 is "+ num2);
}
/** Method to swap two variables */
public static void swap(int n1,int n2){
System.out.println("\tInside the swap method");
System.out.println("\t\tBefore swapping n1 is "+ n1+" n2 is "+ n2);
// Swap n1 with n2
int temp = n1;
n1 = n2;
n2 = temp;
System.out.println("\t\tAfter swapping n1 is "+ n1+" n2 is "+ n2);
}
}
```

This would produce the following result:

```
Before swap method, num1 is1and num2 is2
Inside the swap method
Before swapping n1 is1 n2 is2
After swapping n1 is2 n2 is1
After swap method, num1 is1and num2 is2
```

Overloading Methods:

The max method that was used earlier works only with the int data type. But what if you need to find which of two floating-point numbers has the maximum value? The solution is to create another method with the same name butdifferent parameters, as shown in the following code:

```
public static double max(double num1,double num2){
if(num1 > num2)
return num1;
else
return num2;
}
```

If you call max with int parameters, the max method that expects int parameters will be invoked; if you call max with double parameters, the max method that expects double parameters will be invoked. This is referred to as **method overloading**; that is, two methods have the same name but different parameter lists within one class.

The Java compiler determines which method is used based on the method signature. Overloading methods can make programs clearer and more readable. Methods that perform closely related tasks should be given the same name.
Overloaded methods must have different parameter lists. You cannot overload methods based on different modifiers or return types. Sometimes there are two or more possible matches for an invocation of a method due to similar method signature, so the compiler cannot determine the most specific match. This is referred to as ambiguous invocation**.**

The finalize() Method:

It is possible to define a method that will be called just before an object's final destruction by the garbage collector.
This method is called **finalize( )**, and it can be used to ensure that an object terminates cleanly.
For example, you might use finalize( ) to make sure that an open file owned by that object is closed.
To add a finalizer to a class, you simply define the finalize( ) method. The Java runtime calls that method whenever
it is about to recycle an object of that class.
Inside the finalize( ) method, you will specify those actions that must be performed before an object is destroyed.

The finalize( ) method has this general form:

```
protected void finalize()
{
// finalization code here
}
```

Here, the keyword protected is a specifier that prevents access to finalize( ) by code defined outside its class.
This means that you cannot know whenor even iffinalize( ) will be executed. For example, if your program ends
before garbage collection occurs, finalize( ) will not execute.

Constructors

Constructor in java is a *special type of method* that is used to initialize the object. This can also be called creating an instance. Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

The **constructor** return any value that is current class instance. One cannot use return type yet it returns a value. The constructor can perform other tasks instead of initialization, like object creation, starting a thread, calling method etc. You can perform any operation in the constructor as you perform in the method.

Rules for creating java constructor

- Constructor name must be same as its class name
- Constructor is invoked implicitly.
- Constructor must have no explicit return type
- The java compiler provides a default constructor if you don't have any constructor

Types of java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

1. **Default constructor:**

A constructor that have no parameter is known as default constructor.

If you do not define any constructor in your class, java generates one for you by default. This constructor is known as default constructor. You would not find it in your source code but it would present there.

Default constructor provides the default values to the object like 0, null etc. depending on the type.

Syntax:

```
<class_name>()
{
}
```

Example:

```
class Constru{
Bike1()
{
System.out.println("Default constructor is created");
}
public static void main(String args[])
{
Bike1 b=new Bike1();
}
}
```

Output:

```
Default constructor is created
```

Parameterized constructor

A constructor that have parameters is known as parameterized constructor. Parameterized constructor is used to provide different values to the distinct objects.

```
class Example2
{
    private int var;

    public Example2(int num)
    {
        //code for parameterized one
        var = num;
    }
    public int getValue()
    {
        return var;
    }
    public static void main(String args[])
    {
        Example2 obj2 = new Example2(10);
        System.out.println("var is: "+obj2.getValue());
    }
}
```

Output:

```
var is: 10
```

Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

```
    class Student5{
        int id;
        String name;
        int age;
        Student5(int i,String n){
        id = i;
        name = n;
        }
        Student5(int i,String n,int a){
        id = i;
        name = n;
        age=a;
        }
        void display(){System.out.println(id+" "+name+" "+age);
    }
          public static void main(String args[]){
```

```
        Student5 s1 = new Student5(111,"Karan");
        Student5 s2 = new Student5(222,"Aryan",25);
        s1.display();
        s2.display();
       }
    }
```
Output:
```
111 Karan 0
222 Aryan 25
```

Difference between constructor and method in java

There are many differences between constructors and methods. They are given below.

Java Constructor	**Java Method**
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

Destructor in java

- The destructor is opposite of the constructor, they are executed at the end of the program or when the object goes out of scope.
- A destructor destroys the objects made in the program and clears the memory occupied by the objects.
- The destructors are executed automatically.
- The concept of destructor is called as "Garbage Collection" or GC.
- The Java Virtual Machine automatically calls destructors to destroy the unused and out of scope objects or dead objects, periodically.
- The destructor is called a Finalizer in java and can used in class using finalize() function.
- The destructors never return any value or pass any parameters.

ex.

```
protected void finalize()
{

}
```

- The destructor need not be written in a program. As far as possible do not write the finalizer in java.
- The Finalizer is automatically CALLed by JVM.

Static Keyword

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.
The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

1) Java static variable

If you declare any variable as static, it is known static variable.

- The static variable can be used to refer the common property of all objects e.g. company name of employees, college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.

```
class Student8{
   int rollno;
   String name;
   static String college ="ITS";
   Student8(int r,String n){
   rollno = r;
   name = n;
   }
 void display (){System.out.println(rollno+" "+name+" "+college);}
   public static void main(String args[]){
 Student8 s1 = new Student8(111,"Karan");
 Student8 s2 = new Student8(222,"Aryan");
 s1.display();
 s2.display();
 }
}
```

Output:
```
111 Karan ITS
222 Aryan ITS
```

// Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
class Counter2{
static int count=0;//will get memory only once and retain its value

Counter2(){
count++;
System.out.println(count);
}
  public static void main(String args[]){
Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2();
   }  }
```

```
Output:1
       2
       3
```

2) Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- Static method can access static data member and can change the value of it.

Restrictions for static method

1. The static method cannot use non static data member or call non-static method directly.
2. This and super cannot be used in static context.

2) Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.

Example of static method

```
//Program of changing the common property of all objects(static field).

class Student9{
     int rollno;
     String name;
     static String college = "ITS";
       static void change(){
     college = "BBDIT";
     }
     Student9(int r, String n){
     rollno = r;
     name = n;
     }
    void display (){System.out.println(rollno+" "+name+" "+college);}
     public static void main(String args[]){
    Student9.change();
     Student9 s1 = new Student9 (111,"Karan");
    Student9 s2 = new Student9 (222,"Aryan");
    Student9 s3 = new Student9 (333,"Sonoo");
     s1.display();
    s2.display();
    s3.display();
    }
}
```

```
Output:111 Karan BBDIT
       222 Aryan BBDIT
       333 Sonoo BBDIT
```

3) Java static block

- Is used to initialize the static data member.
- It is executed before main method at the time of classloading.

Example of static block

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

```
Output:static block is invoked
       Hello main
```

Arrays in JAVA

- An array is a collection of data with same data type or similar data type.
- An array is useful to store data of same type in the memory and then process the data.
- An array a basic collection.
- Array is declared using data type, name of array and [ ]
- Each data of array is called Element and every element has an array address or index that begins with 0 and left to right.
- There are two types of arrays

1. Single dimension 2. Multi dimension array

1. Single dimension array

- In a single dimension array the data is stored on one dimension only
- Every element of Single dimension array has a single index that begins with 0.

ex.

```
            0 1 2 3 4 5 6 7 8
int a[] = { 1,2,3,4,5,6,7,8,9 }; // predefined array.
```

```
// array declaration
int a[], i, j, t;   ----- array and simple variable.
int[] a, b, c;    ----- all are considered as arrays. Not possible in C/C++

int a[100] ---- in C and C++
```

- In Java, the array size, at the time declaration is unknown
- Later on array can be initialized by using new operator.

ex.

- Eclipse is a world class IDE (Integrated Development Environment) it is one the popular and widely used software for Java development. There are many such IDE like NetBeans, Sun One, Java Studio, JBuilder, Sun Studio, JCreater
- There are many versions of eclipse like Helios, Juno, Indigo, Kepler, Luna, Mars (***)
- It is fully open source and free software.

```
int a[];
a = new int[10]; -- array of 10 elements [ 0 to 9 ]
a = new int[n];  -- array of n elements [ 0 to n-1 ]
```

- Finally a single dimension array can be processed by a simple for loop.
- Arrays are very important and commonly used in programs java programs.
- Each SDA has a built-in property called length which returns an int value that the length or size of array.

```
int a[]; // declare an array
a = new int[10]; // initialize the array

System.out.println("Length of array is " + a.length );
```

length is the property of any type of array which gives the size of array.

Single dimension array programs

1. Sorting of array
    1. Selection Sort
    2. Bubble sort

2. Searching of array
    1. Linear Search
    2. Binary Search

a. Sorting of array

- Compare two elements and exchange using a temp variable.

1. Selection Sort

```
       0   1   2    3   4
a   =   2   4   5    8   9

Pass1   Pass2
0>1     1>2    2>3   3>4
0>2     1>3    2>4
0>3     1>4
0>4
```

1. Bubble Sort

```
       0   1   2  3   4
a   =   3   4   5  7   9

0>1   0>1   0>1   0>1
1>2   1>2   1>2
2>3   2>3
3>4
```

b. Searching of array

- Searching of array is process of finding an element or data in a stored array
- If the data is found then "Data Found" message can be displayed with its array location.
- If the data is not found in the array till end then "Data Not Found".
- There are 2 methods of searching

1. Linear Search method 2. Binary search method

1. Linear Search method

- It is very simple, easy and common method in which a number is matched wirth each data in array from the begining till the end of array
- If the data is matched is searching is stopped

```
        0  1  2  3  4
a  =  9  1  5  7  2
```

x = 5

2. Binary Search an Array

- In this type of searching the array must be sorted before the searching starts
- There are 3 variables

lb - lower bound - starting address of array 0

ub - upper bound - largest index in the array n-1

mid - calculated as the (lb+ub) / 2

- Match the array element with data if matched then print array address and stop searching.

b. Multidimensional Arrays

- The data in the MDA are stored on 2 or more dimensions
- The most popular, common and widely used MDA are the 2-D arrays
- They are also called Matrix or Matrices.
- There are large number of matrix applications.
- The matrix are also popular in developing board games.
- The elements in a 2-D are stored on rows and columns.
- Each element or data of MDA has "row" and "column" address.
- Both of them begin with 0

```
        0   1   2
------------------------
   0    1   2   3
   1    4   5   6
   2    7   8   9
```

- the matrix can be processed using 2 nested for loop
- outer loop represents rows and inner loop represents columns.
- The 2-D is declared using data and two [] both empty.

ex. 1
```
int a[][],i,j;
```

ex. 2
```
int[][] a,b,c;
```

ex.
```
int a[][];                    ----- Declare a matrix
a = new int[3][3];      ----- Initialize a matrix

int a[][] = new int[4][4];
```

- There are 2 ways to declare a matrix

1. Predefined matrix -- The data is pre-declared.

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

Programs of Matrix

1. Transpose a matrix
2. Addition of rows in Matrix
3. Diagonal addition
4. Reverse diagonal addition
5. Identity matrix
6. Multiplication of Matrix

$$\begin{array}{ccccccc}
 & 0 & 1 & 2 & 3 & i & j \\
0 & 0 & 0 & 0 & 1 & 0 & 3 \\
1 & 0 & 0 & 1 & 0 & 1 & 2 \\
2 & 0 & 1 & 0 & 0 & 2 & 1 \\
3 & 1 & 0 & 0 & 0 & 3 & 0
\end{array}$$

1. Transpose a matrix

- The rows are transposed to columns.

1 2 3
4 5 6

1 4
2 5
3 6

5. Identity matrix

- the diagonal elements are 1 and others are 0

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

Multiplication of matrix

A

$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 7 & 8 & 9 \end{array}$$

B

$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 7 & 8 & 9 \end{array}$$

C

-- -- --

Object-Oriented Programming Concepts

If you've never used an object-oriented programming language before, you'll need to learn a few basic concepts before you can begin writing any code. This lesson will introduce you to objects, classes, inheritance, interfaces, and packages. Each discussion focuses on how these concepts relate to the real world, while simultaneously providing an introduction to the syntax of the Java programming language.

What Is an Object?
An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life. This lesson explains how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner.

What Is a Class?
A class is a blueprint or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object. It intentionally focuses on the basics, showing how even a simple class can cleanly model state and behavior.

What Is Inheritance?
Inheritance provides a powerful and natural mechanism for organizing and structuring your software. This section explains how classes inherit state and behavior from their superclasses, and explains how to derive one class from another using the simple syntax provided by the Java programming language.

What Is an Interface?
An interface is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface. This section defines a simple interface and explains the necessary changes for any class that implements it.

What Is a Package?

A package is a namespace for organizing classes and interfaces in a logical manner. Placing your code into packages makes large software projects easier to manage. This section explains why this is useful, and introduces you to the Application Programming Interface (API) provided by the Java platform.

Introduction to OOPS (Object Oriented Programming System)

- Java is fully Object Oriented Languages OOPS.

- Generally there are 2 types of programming languages
  a. Procedure Oriented Languages - C (*), COBOL, FORTRAN, Pascal, BASIC
  b. Object Oriented Languages - C++, Java, C#, php, python, perl, Eifell

- The old method of programming was procedure oriented and the languages were known as procedure oriented languages.
- The procedure oriented program executes in linear fashion from top to bottom.
- The procedure oriented programming became less popular because of some limitations.
- In 1982 a concept was developed called Object Oriented Technology by Grady Booch, he developed concepts of Object Oriented Technology.
- He did not develop any programming language, he published a book 'Methodology of Object Oriented Technology'
- He developed a language called Unified Modelling Langugage. It is used for Object Oriented Analysis and Design.
- The limitations of the Procedure Oriented Programming are mainly 2

1. Less importance and security to data
2. Lack of reusability in program

- Procedure oriented programming provides less security and importance to data and more emphasis is on program.
- The programs developed using Procedure Oriented programming provides less reusability.

- Both above features are not applicable to modern day programming and software development.
- The OOPS has overcome both the limitations of Procedure Oriented Programming and it perfectly suited for modern programming languages.
- All modern computer programming languages are Object Oriented except one that is C.
- The OOPS provides high degree of security to data and great reusability of program.
- There are 4 features of Object Oriented Technology as decribed by Grady Booch in his book.

```
                         OOT
                          |
-----------------------------------------------------------------
|        |          |            |               |
OOPS    OODBMS     OOAD        OOCAD          OOOS
```

1. Encapsulation   2. Inheritance   3. Polymorphism   4. Abstraction

1. Encapsulation

- The encapsulation is the fundamental and basic feature of OOPS and all other features are developed on Encapsulation.
- The encapsulation is the foundation of OOPS.
- Encapsulation refers to creating classes and objects.
- A class is collection or "data and functions" or in general it is collection of "attributes and behavior".
(Attribute is the properties and bevaiour is the activities)
- The object is an instance or example of a class.
- A class a template or a skeleton and the object is the real life enity or real life model.

2. Inheritance
- This features refers to creating new classes called Derived or sub class from existing class called as Base class or super class.

- The inheritance is extension to the existing class.
- All the features of the base must be available and accessible in the derived class.
- In addition the derived class may have some new properties. hence the derived class is the extension of the base.

3. Polymorphism

- The polymorphism is another imp. feature of OOPS.
- It refers to defining many functions in a class with same name but different passing parameters or returning value.
- Many activities of one behaviour that change based on runtime situation.
- The word polymorphism is made of word poly and morph. Poly is many and morph is forms.

4. Abstraction

- The abstraction refers to hiding all important details form the world and exposing a very things.
- The data abstraction is an important concept in OOPS.
- Only a few things are exposed and all important details are hidden.

Java as Object Oriented Programming Language.

1. Encapsulation

- Encapsulation refers to declaring class and creating object from the class.
- A class can be created using class keyword and the name of class which is user defined or created by the user.
- A class contains data and functions or also called as "Data Members" and "Member Functions".
- The data in the class should be declared by "private" keyword
(Access Modifier or Access Specifier)
- The functions of a class are declared by using "public" keyword.
- The private keyword makes data members of class not accessible outside the class.
- The public functions can be accessed outside the class.

- In this way data becomes hidden from outside the class.

ex.

```
class Student
{
   private int rno;
   private String name;
   private float per; // data members

   public void setData()
   {
       rno = 10; name = "Amey; per = 89.78f;
    }
   public void showData()
   {
      System.out.println("Rno = " + rno + "  Name = " + name + " Per = " + per);
    } // member functions
}

Student  s1;   // s1 is the object of Student class
s1 = new Student();  // instantiate or initialize
s1.setData();
s1.showData();

System.out.println("Rno = " + s1.rno + "  Name = " + s1.name + " Per = " + s1.per);
---- Error
```

- The object can be created from a class and instantiated or initialized by new operator.
- Lastly the functions are accessed using object and dot operator.

Topics of Encapsulation

1. Classes and objects ***
2. Access modifiers or Access specifiers ***
   - private
   - public
   - protected
3. Constructors in Java
   - Defualt
   - Parameterised
4. Destructor in java
   - Finalizer
5. passing parameters to member function
6. Returning value from a member function
7. Static data and functions
8. Overloading in Java
   a. Function overloading
   b. Construcor overloading
9. Objects and arrays

Access modifiers or specifiers

- The access modifiers are the keywords used to specify and declare the data or functions in a class. They declare the accessibility or visibility or the members in the class (dada and functions)
- There are types access modifiers in Java

1. public 2. private 3. protected

1. public

- The public AM declares that the data member or member function can be accessed anywhere i.e. inside class and outside class.
- Generally the functions are declared using public keyword and hence can be accessed outside the class.

2. private

- The private AM declares that the data member or member function can be accessed only inside class and not outside class.
- Generally the data of a class are declared using private keyword and hence can not be accessed outside the class.

3. protected (***)

- This access modifier is used in inheritance to declare the data members of a base class.
- The protected specifies that the data are available in the base class as well as the derived class.

Overloading in Java

- The overloading is also important feature of Object Oriented Programming and supported by Java
- The overloading refers to defining or writing two or more functions with same name but different passing parameters, in a class.
- Depending upon the parameters passed, during runtime one of the appropriate functions get executed.
- The use overloading makes program faster in executing, effecient and easy to handle and easy to remember the function names.

- There are 3 types of overloading in "Object Oriented Programming" like..
  a. Function overloading
  b. Constructor overloading
  c. Operator overloading (Not supported in java but it is available in C++)
- But java supports only two overlaoding i.e. a and b

a. Function overloading

- A class may contain many functions written with same name but different passing parameters.

```
public void swap(int a, int b)
{
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
   System.out.println("A = " + a + "  B = " + b);
}
public void swap(float a, float b)
{
   float tmp;
   tmp = a;
   a = b;
   b = tmp;
   System.out.println("A = " + a + "  B = " + b);
}
public void swap(String a, String b)
{
   String tmp;
   tmp = a;
   a = b;
   b = tmp;
   System.out.println("A = " + a + "  B = " + b);
}

----
swap(10,20);
swap(1.7f,3.5f);
swap("aaa","bbb");
```

- An overloaded function may have same number of parameters but different data types
- overloaded functions may also have many parameters with same data types.

2. Constructor overloading

- The constructor overloading is exactly same is function overloading
- Only parameterized constructors can be overloaded.

- Functions can be overloaded till there are NO parameters passed.
- Constructor can overloaded till there is at least one parameter.

Objects and arrays

- The array of object is a collection of objects of same class declared as an array.
- The array of object is required when there are many objects created or declared from a class.
- The array of objects are convenient for processing using a for loop and also using foreach loop.
- The array begins with 0 index till the last element in the array.
- There are 2 initilizations in array of object
  1. Initialize the array with some.
  2. Instantiate each object in the array using a for loop.
- The arrays and objects or array of objects is very important particularly on collection classes.

ex.

```
Student s[];

s = new Student[5]; // initialize the array ---- 1

for(i=0;i<5;i++)
    s[i] = new Student();  // initialize each object in the array  ---- 2
```

2. Inheritance

Inheritance is one of the feature of Object-Oriented Programming (OOPs).

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.
It allows the creation of hierarchical classifications.
The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).
Inheritance is a <u>compile-time</u> mechanism. A super-class can have any number of subclasses. But a subclass can have only one superclass. This is because Java does not support multiple inheritance.
Advantage of inheritance is that once you have created a superclass that defines the attributes common to a set of objects, it can be used to create any number of more specific subclasses.
The **extends keyword** indicates that you are making a new class that derives from an existing class.

Syntax

```
class Super {
  .....
  .....
}
class Sub extends Super {
  .....
  .....
}
```

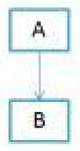
Access Modifiers for inheritance:

The derived class inherits all the members and methods that are declared as public or protected. If declared as private it cannot be inherited by the derived classes. The private members can be accessed only in its own class. The private members can be accessed through assessor methods as shown in the example below. The derived class cannot inherit a member of the base class if the derived class declares another member with the same name.

Types of inheritance in java
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

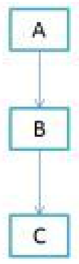
1. Single Inheritance

Single inheritance is Very easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.



(a) Single Inheritance

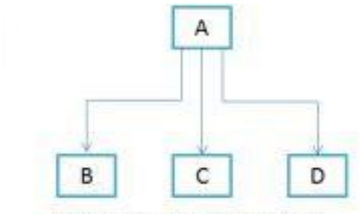
2. **Multilevel Inheritance**
Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



(d) Multilevel Inheritance

3. Hierarchical Inheritance

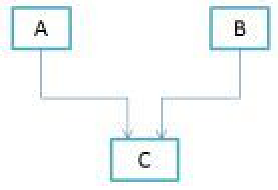
In such kind of inheritance one class is inherited by many **sub classes**. In below example class B,C and D **inherits** the same class A. A is **parent class (or base class)** of B,C & D.



(c) Hierarchical Inheritance

4. Multiple Inheritance

"**Multiple Inheritance**" refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes. Using multiple inheritance often leads to problems in the hierarchy. This results in unwanted complexity when further extending the class.

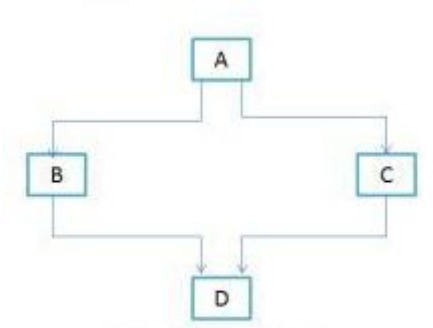


(b) Multiple Inheritance

Note : Most of the new OO languages like **Small Talk, Java, C# do not support Multiple inheritance**. Multiple Inheritance is supported in C++.

5. Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of **Single, Hierarchical** and **Multiple inheritance.** A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance ! Using interfaces. By using **interfaces** you can have multiple as well as **hybrid inheritance** in Java.



(e) Hybrid Inheritance

This keyword

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of java this keyword
Here is given the 6 usage of java this keyword.

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.

1) The this keyword can be used to refer current class instance variable.

If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

```
//example of this keyword
class Student11{
    int id;
    String name;
    Student11(int id,String name){
    this.id = id;
    this.name = name;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    Student11 s1 = new Student11(111,"Karan");
    Student11 s2 = new Student11(222,"Aryan");
    s1.display();
    s2.display();
}
}
//example of this keyword
class Student11{
    int id;
    String name;

    Student11(int id,String name){
    this.id = id;
    this.name = name;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    Student11 s1 = new Student11(111,"Karan");
    Student11 s2 = new Student11(222,"Aryan");
    s1.display();
    s2.display();
}
}
```

```
Output111 Karan
      222 Aryan
```

this() can be used to invoked current class constructor.

The this() constructor call can be used to invoke the current class constructor (constructor chaining). This approach is better if you have many constructors in the class and want to reuse that constructor.

```
//Program of this() constructor call (constructor chaining)

class Student13{
   int id;
   String name;
   Student13(){System.out.println("default constructor is invoked");}

   Student13(int id,String name){
   this ();//it is used to invoked current class constructor.
   this.id = id;
   this.name = name;
   }
   void display(){System.out.println(id+" "+name);}

   public static void main(String args[]){
   Student13 e1 = new Student13(111,"karan");
   Student13 e2 = new Student13(222,"Aryan");
   e1.display();
   e2.display();
  }
}
```

Output:

```
default constructor is invoked
default constructor is invoked
111 Karan
222 Aryan
```

The this keyword can be used to invoke current class method (implicitly).

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method.

```
class S{
 void m(){
```

```
 System.out.println("method is invoked");
 }
 void n(){
 this.m();//no need because compiler does it for you.
 }
 void p(){
 n();//complier will add this to invoke n() method as this.n()
 }
 public static void main(String args[]){
 S s1 = new S();
 s1.p();
 }
}
class S{
 void m(){
 System.out.println("method is invoked");
 }
 void n(){
 this.m();//no need because compiler does it for you.
 }
 void p(){
 n();//complier will add this to invoke n() method as this.n()
 }
 public static void main(String args[]){
 S s1 = new S();
 s1.p();
 }
}
```
Output:method is invoked

Super keyword in java

The **super** keyword in java is a reference variable that is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.
Usage of java super Keyword

1. super is used to refer immediate parent class instance variable.
2. super() is used to invoke immediate parent class constructor.
3. super is used to invoke immediate parent class method.

1. super is used to refer immediate parent class instance variable.

```
//example of super keyword

class Vehicle{
  int speed=50;
}

class Bike4 extends Vehicle{
  int speed=100;

  void display(){
   System.out.println(super.speed);//will print speed of Vehicle now
  }
  public static void main(String args[]){
   Bike4 b=new Bike4();
   b.display();

}
}
Output:50
```

2) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor as given below:

```
class Vehicle{
  Vehicle(){System.out.println("Vehicle is created");}
}

class Bike5 extends Vehicle{
  Bike5(){
   super();//will invoke parent class constructor
   System.out.println("Bike is created");
  }
  public static void main(String args[]){
   Bike5 b=new Bike5();

}
}
```

```
Output:Vehicle is created
       Bike is created
```

3) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used in case subclass contains the same method as parent class as in the example given below:

```
class Person{
void message(){System.out.println("welcome");}
}

class Student16 extends Person{
void message(){System.out.println("welcome to java");}

void display(){
message();//will invoke current class message() method
```

```
     super.message();//will invoke parent class message() method
     }

     public static void main(String args[]){
     Student16 s=new Student16();
     s.display();
     }
     }
Output:welcome to java
       Welcome
```

In the above example Student and Person both classes have message() method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority is given to local.

Abstract, final and public classes

1. abstract class

- The abstract classes play very important role in creating inheritance
- The abtract class is generally the super declared using abstract keyword
- The abstract class must be inherited into at least one sub class
- The object of an abstract can be never be instantiated.

ex.

```
abstract class Student
{

}
public final class Result extends Student
{

}
```

2. final class

- The final class is the last class in the inehritance declared using final keyword
- The objects of the final class can be created and instantiated
- The final class can not be further inherited or it can not create sub classes.

3. public class

- A class may be declared as public using public keyword.
- The public class can be used in any other java program stored in a separate file.
- This concept brings modularity to the java applications.
- The public keyword makes a class public and accessible in any other java program.

Interfaces

An **interface in java** is a blueprint of a class. It has static, constants and abstract methods only.
The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.
Java Interface also **represents IS-A relationship**.
It cannot be instantiated just like abstract class.
abstract class is used to achieve partial abstraction(hiding irrelevant details from user). But the interfaces are used for achieving full abstraction.
The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

An interface is similar to a class in the following ways −

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including −

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

In this post we will discuss **difference between Abstract Class and Interface in Java with examples.**

	abstract Classes	**Interfaces**
1	abstract class can extend only one class or one abstract class at a time	interface can extend any number of interfaces at a time
2	abstract class can extend from a class or from an abstract class	interface can extend only from an interface
3	abstract class can have both abstract and concrete methods	interface can have only abstract methods
4	A class can extend only one abstract class	A class can implement any number of interfaces
5	In abstract class keyword ‘abstract’ is mandatory to declare a method as an abstract	In an interface keyword ‘abstract’ is optional to declare a method as an abstract
6	abstract class can have protected , public and public abstract methods	Interface can have only public abstract methods i.e. by default
7	abstract class can have static, final or static final variable with any access specifier	interface can have only static final (constant) variable i.e. by default

Declaration
Interfaces are declared by specifying a keyword “interface”. E.g.:

```
interface MyInterface
{
  /* All the methods are public abstract by default
   * Note down that these methods are not having body
   */
  public void method1();
  public void method2();
}
```

Creating a package in java

- A package is a collection of classes (without main function)
- There are 2 types of packages

1. Predefined or built-in java packages or API packages. (application programming interface or libraries)
   java.util, java.io, java.net, java.lang, java.applet, java.awt.
These are commonly used java packages.

2. User created package
   These packages are created by the user. Once created they can be used in any java program by using import statement.

- A package is created using package keyword and the name of package, the package name is user defined.
- The package statement must be written in the begining of program.
- After compiling the program having package, a new folder is created by the name of package and the class file is generated into the folder.
- A new java program can be created using import statement and the name of package to access the classes in the package.

Introduction to JAR

- JAR (Java ARchive) is a package file used to collect many Java class files, associated metadata and resources (text, images) into one file
- A JAR file is a compressed file used to distribute java applications or java library
- JAR files are also used to distribute libraries for the Java platform.
- They are built on the ZIP file format and have the .jar file extension.
- A JAR file can be created using jar utility that comes with any JDK version.
- JAR is a basic build and deployment tool.
- A JAR file may optionally contain a manifest file with .mf extension

Creating Jar file without manifest file

```
m:\new adv java\>jar -cvf HelloWorld.jar HelloWorld.class
```

- Create new jar
- Verbose list of file
- File name of the jar file

Creating Jar file with manifest (ideal)

- A manifest file can be created using any editor like notepad
- The manifest decribes the jar file and contains the mainclass or the entry point of application.
- Main properties of manifest file are

```
Main-Class: HelloWorld
```

```
m:\new adv java\>jar -cvfm HelloWord.jar HelloWorld.mf HelloWorld.class
```

Creating excutable jar file

```
jar -cvfe HelloWorld.jar HelloWorld HelloWorld.class
```

run the jar file using command

```
java -jar HelloWorld.jar
```

Viewing the contents of jar file

```
jar -tvf HelloWorld.jar
```

- Table of contents

extracting contents in a jar file
- eXtract the contents of jar

```
jar -xvf HelloWorld.jar
```

Creating jar file from Eclipse

- Select Project for which you want to create jar file.
- Go to File Menu and select Export
- Expand Java folder and select JAR file

3. Polymorphism

- The 4th important concept of OOPS is polymorphism.
- The polym. gives a very dynamic aspect to a java application.
- The polym word is made of two words "poly" and "morph" means many forms
- The polym. refers to defining many functions with same name but different behaviour during runtime.
- In other words when a function behaves differenltly in different runtime situation it is said to be a polymorphic function.
- This kind of approach provides a dynamic appearance to a program.
- There are 2 types ploym.

1. Static polym 2. Dynamic polym

1. Static polym (*****)

- The static polym is also called as Compile time polym or also called Early binding, compile time binding and static binding. Fixed binding.
- The example of of static polym is function overloading and constructor overloading.
- The binding is a process of associating a function with an object using a dot operator. ex. s1.getData().
- The static binding happens during program compilation.
- The example of static polym. are all types of "Overloading"
1. function and operator overloading

Compilation and execution/runtime

s1.getData()

2. Dynamic polym

- The static polym is also called as Run time polym or also called Late binding, run time binding and dynamic binding.
- The example of of dynamic polym is "function overriding".
- The binding is a process of associating a function with an object using a dot operator.
- The dynamic binding happens during program runtime or execution.
- In dynamic polymorphism the functions have same name in the base class as well as the derived.
- The object is created from the base class and control of function flows from the base class to the derived.
- Although the object is created from the base class it is never instantiated on the base but the object is instantiated on the derived class or any one of the derived class.
- Finally the functions of the derived class are executed.

ex.

class student

```
{
   public void setdata()
   {
   }
   public void putdata()
   {
   }

}

class engg extends student
{

   public void setdata()
   {
   }
   public void putdata()
   {
   }

}
class medical extends student
{
   public void setdata()
   {
   }
   public void putdata()
   {
   }
}
----
----
student s1;

s1 = new engg();
```

```
s1.setdata();
s1.putdata();

s1 = new medical();
s1.setdata();
s1.putdata();

Student s1;
s1 = new Medical();
s1.setData();
s1.showData();

s1 = new Engineering();
s1.setData();
s1.showData();

s1 = new Arts();
s1.setData();
s1.showData();
```

- The overridden function in the derived extends the functionality of the base. That means the statements of the function in the base class are repeated in the derived.
- This feature called Extending functionality.

4. Abstraction

- In the case of polymorphism, the object is created from the base class but the object never instantiates the base class instead it can instantiate any derived class and this brings dynamic polymorphism.
- The function control starts from the base but it executes the functions of the derived class which is instantiated.
- For the sake presence or the names base class must have all the functions which are having same name as in derived class.
- The base class function never execute and they can be declared without function body or statements using abstract keyword.

ex.

```
public  abstract void getdata();
public  abstract void setdata();
public  abstract void putdata();
```

- A class that contains abstract functions must also be declared as abstract with abstract keyword.

========================Part 2 Ends========================

GUI Programming with Java

Introduction to windows programming or GUI programming

- GUI based application is a program that runs on windows operating system
- The GUI program has all the features of windows application like mouse activity, cursor, bitmaps, images, various fonts, colors or graphics and large number user interface components or controls.
- The use of all the above makes a windows application.
- The Windows applications runs on events there fore it is also called Event driven programming.
- Java has powerful facility to develop the windows application or windows programming.
- There are 2 methods of developing a windows application in java

1. AWT   -- Abstract Window Toolkit (Older method)
2. Swing -- Modern Method for windows application developmen
3. GWT (*)

- AWT is an old and very popular method for development of windows applications in java
- The swing is more modern and "graphically more rich" GUI application development methods.

1. Abstract Window Toolkit

- The AWT provides a very large collection of classes and interfaces which come from java.awt package.
- These classes and interfaces can be used in the java program to develop windows based application.
- The most commonly used classes are the control classes or UI classes.
- These classes are used to develop user interface screens or data entry froms.
- The list of AWT classes for UI controls is as follows...

1. Frame -- this class is used to create the main frame of the application, It has maximize, minimize and close button. Every windows application has a Frame. The frame holds the container which can be a Panel or canvas. Every windows application has a single frame .

2. Panel -- The working area inside frame is called as Panel. The panel is used to add the user inerface controls. The panel fits into the frame.

3. Label -- The label is a control that displays static text. The label is only for display purpose.

4. TextField -- The control to add data or type data in a single line. it can be used to enter any type of data but considered as string data.

6. Button -- This class creates a button for the data entry form. It is also called as command button which can clicked to complete the activity like of a form ok, cancel button.

5. TextArea -- The control which has lines and columns and data can be entered in many lines. It is useful to enter text that goes into more than one lines.

7. List -- It Displays a list of items from which one item can be selected or many items can be selected. It is a selection control.

8. Choice -- It is a selection control which has a dropdown button.

9. Checkbox -- It is a square shaped selection control which can be checked or unchecked i.e. selected or unselected.

10. Checkbox Group -- used to create radio button which is circular in shape and only one can be selected at a time.

- The above control or classes are commonly used in a GUI program
- There are some advanced control or classes.

1. MenuBar -- It is the horizontal bar on which menus are created
2. Menu -- Each text on the menubar is called Menu
3. MenuItem -- The submenu items under each menu is called MenuItem
4. ToolBar -- It is below the menubar and it contains command buttons or toobar buttons.
5. StatusBar -- it is at the bottom of the screen and it shows the different information about the application like line no, column number etc.
6. ProgressBar -- it show the progress of an activity like copiying files from one drive to another drive or installation process.
7. ScrollBar -- The scroll is very common in many application. It is used scroll or move the contents of an application. There are two scroll bar vertical and horizontal.
8. Slider -- It is movement control which moves by the mouse. There can be a vertical slider or horizontal slider. Example of a slider can be sound volume control of windows OS.
9. Dialog -- It is a small window that may contains some UI controls.

1. Creating a windows program using AWT.

2. Swing components

- The swing is more modern and graphically rich system for developing GUI application
- It is more popular than AWT now a days
- All swing classes and interfaces come from javax.swing package.

- Almost all classes of AWT are available in swing in addition there are some new classes and controls in swing.
- All swing classes begin with letter J

1. JFrame -- this class is used to create the main frame of the application, It has maximize, minimize and close button. every windows application has a single frame .
2. JPanel -- The wodking area inside frame is called as Panel
3. JLabel -- The label is a control taht displays static text. The label is only for display purpose
4. JTextField -- The control to add data or type data in a single line
5. JTextArea -- The control which lines and columns and data can be entered in many lines.
6. JButton -- The command button which can clicked to complete the activity like ok, cancel button.
7. JListBox (List in awt) -- Displays a list of items from which one item can be selected.
8. JComboBox (Choice in awt) -- Like a list box but it has a dropdown button.
9. JCheckBox -- It is a square shaped selection control which can be checked or unchecked.
10. JRadioButton (CheckboxGroup)
11. JPasswordField -- To enter password.

- The above control or classes are commonly used in a program
- There are some advanced control or classes.

1. JMenuBar -- It is the horizontal bar on which menus are created
2. JMenu -- Each text on the menubar is called Menu
3. JMenuItem -- The submenu items under each menu is called MenuItem
4. JToolBar
5. JStatusBar
6. JProgressBar
7. JScrollBar
8. JSlider
9. JDialog

10. JOptionPane (***)
11. JTabbedPane (***)
12. JTable
13. JTree

Name	- TextField
Addr	- TextField
Age	- TextField
city	- List
State	- Choice
Gender	- CheckboxGroup
Qual	- Checkbox
Ok, Cancel	- Button

Layout Managers in Java

- A layout is a systematic arrangement of controls on the panel or a container.
- The layout decides the locations and positions of controls or components on the panel.
- The layout is managed by a special class called Layout manager
- There are 6 layout manager classes in awt package.
- The default layout manager is called FlowLayout which is already present on a panel.
- This layout puts the control one after another, from left to right and on the next row when the right margin comes.
- All layout classes come from java.awt package.

1. FlowLayout (Default layout)
2. BorderLayout
3. GridLayout
4. CardLayout
5. BoxLayout
6. GridBagLayout

2. BorderLayout

- The BorderLayout is a simple layout and it used to add controls on the panel on 5 directions
- The direction on which the controls are added are North, South, East, West and Center
- This layout manager is useful for developing applications like games in Java. or adding some components on 5 directions.
- The object of the BorderLayout is instantaitated with an empty constructor.
- A setLayout() function is used to set the new layout on the panel.
- All layout manager classes come from java.awt package.

ex.

```
BorderLayout bl;
bl = new BorderLayout();
Panel pan = new Panel();
pan.setLayout(bl);
```

- The controls can be added on 5 direction using add() function.
- The borderlayout is useful in windows application for adding menubars, tool bars (North), scoll bars(East and south) etc.
- All or some of the directions may be used in a program depending upon need.

3. GridLayout

- This layout is also very popular in developing board games and matrix applications.
- In the layout the controls are added on a grid or rows and columns like a matrix.
- There are fixed number of rows and columns on which the controls are added.
- The object of GridLayout class passes 2 contructors parameters i.e. and number of rows and columns.

ex.

```
GridLayout gl = new GridLayout(3,3);
pan.setLayout(gl);
```

- The setLayout() function is used to set the new gridlayout and remove defauly layout.

4. CardLayout

- The CardLayout is an old layout manager which has now become outdated and very rarely used.
- The CardLayout is now replaced by a special swing class called JTabbedPane
- There are some limitations of this layout and hence became outdated.
- In a card layout there are multiple panels or many panels each having a set of different controls.
- These panels are placed on the cardlayout like horizotnal stack.
- Only one panel is visible at a time and other panels are hidden behind the visible panel.
- The JTabbedPane has implementation of CardLayout each card having a named tab or tab with a name by which a user can access the panel.

JTabbedPane class

- It is new class of swing package.
- It is a replacement of CardLayout
- The object of JTabbedPane is created with a default constructor.
- The addTab() function is used to add the panels on the control.
- This function passes 2 parameter
1. Title of the Tab
2. the object of panel.
- Finally add the JTabbedPane object on the frame.

ex.

```
CardLayout cl = new CardLayout();
pan.setLayout(cl);
```

5. BoxLayout (layout of layouts)

- The BoxLayout is used to combine two or more layout in one application
- It is a combination of two or more layouts
- It is a mixed layout of many layout for ex. A FlowLayout and GridLayout or GridLayout and BorderLayout.
- It is a layout of layouts.
- It is required in some specific application.

6. GridBagLayout

- This was a popular layout to add controls on the panel and create a neat and systematic "data entry forms".
- It is used to add controls at given location on the panel.
- The GridBagLayout always works with another important class called GridBagConstraints.
- The GridBagConstraints is more important than the layout class.
- This class sets the rules and locations to add the controls on the panel.

ex.
```
GridBagLayout gb = new GridBagLayout();
GridBagConstraints gc = new GridBagConstraints();

pan.setLayout(gb);
```

- The GridBagConstraints has three important properties

1. anchor -- it is used to set direction for adding control on the panel.
   There are 8 directions on which the controls can be added on the panel.

```
GridBagConstraints.NORTH
GridBagConstraints.SOUTH
GridBagConstraints.WEST
GridBagConstraints.EAST
GridBagConstraints.NORTHWEST ****
```

GridBagConstraints.NORTHEAST
GridBagConstraints.SOUTHWEST
GridBagConstraints.SOUTHEAST

- The most common and popular anchor is NORTHWEST to start the controls on the panel.

2. gridx -- it is used to set the x location of the control on the panel
(x co-ordinate)

3. gridy -- is used to set the y location of the control on the the panel.
(y co-ordinate)

- This layout requires a lengthy java program to implement.
- The controls shift their positions when the frame size is changed.
- This layout is also not very effecient and it is replaced by setBounds() function.

setBounds()

- The setBounds() function is an atrenative to GridBagLayout and much more easier and effecient than the layout.
- It reduces the program length also
- Using setBounds() the controls can be properly placed on the panel and they do not shift even the panel is resized.
- Hence the professional and good looking data forms can be created using setBounds()
- For using the setBounds() function a panel should be "free of layout" i.e. panel must not have any layout. Panel should blank or without layout

ex.
pan.setLayout(null);

- The setBounds() function passes 4 parameters
1. x location
2. y location

3. width of control
4. height of control

ex.

```
                x  y  w  h
lname.setBounds(10,10,50,20); pan.add(lname);
tname.setBounds(70,10,100,20); pan.add(tname);
```

9. Event Handling in Java

- The event handling is a very important concept in windows or GUI programming.
- A windows program cannot be complete without event handling.
- An event is an activity or action that happens during program execution.
- The event can be a Button click, Mouse click, double click, right click, mouse move, mouse drag.
- The event is typical and important feature of windows programming or windows application.
- An windows application run on various events (DOS based or console program does not have event based execution.)
- Differences between console and windows program

Console Program	Windows Program
1. Runs in command prompt (B/W)	Runs on windows
2. Black and white interface (CUI) Character User Interface	Graphical User Interface
3. Runs in linear fashion from top to bottom.	Does not run in linear fashion.
4. Automatically ends when program is complete.	Does not automatically end or exit
5. Non event based programming	Event based programming

- The events are generated by either mouse activity or key board activity
- Based on the event generated, the programs responds by executing a special function called "Event Handler" function
- Java supports different types of events through a collection of classes and interfaces which come from java.awt.event package.
- There are many events depending upon scope and activity
- The most common events that are handled by java are...

1. ActionEvent 2. MouseEvent 3. WindowEvent
4. KeyEvent 5. ComponentEvent 6. AdjustmentEvent

1. ActionEvent

- The ActionEvent is one of most common and popular event handling used in java programs
- This event occurs by pressing the buttons like ok, cancel buttons, also by selecting menu items, by executing toolbar buttons etc.
- The action events are supported by ActionEvent class, ActionListener interface and an actionPerformed() function.
- An additional function addActionListener() is also used to register the event on the buttons.

ActionEvent - class
ActionListener - interface
actionPerformed()
addActionListener() -
getSource() - functions

Steps to use action event in a java program

1. import java.awt.event package
2. The main java class should implement ActionListener interface which makes the program listen to event.

3. The button should register the event by using addActionListener() function for current application. This function makes the buuton active and capable of responding to events.
4. implement the actionPerformed() function which passes the object of ActionEvent class. The object of ActionEvent class contains the control or button which is clicked by the user. A getSource() function is used to recongnize the name of the control or the button.

Biodata Form (AWT)

1. Name -- textfield
2. Addr -- textfield
3. city -- list
4. state -- choice
5. Qual -- checkbox
6. Gender -- checkboxgroup
7. ok, cancel -- button

Create the Biodate using setBounds() function and event handling.

2. MouseEvent

- The second popular and commonly used event is MouseEvent.
- It is used to handle the mouse activity in applications like paint brush and other drawing programs, games etc.
- The mouse events take place on any mouse activity like mouse move, mouse drag, mouse click, mouse pressed, mouse released etc.
- The mouse events can be used in a program by "one class and two interfaces".
- The name of class is MouseEvent and the interfaces are MouseListener and MouseMotionListener
- The MouseListener contains 5 functions and the MouseMotionListener has 2 functions.

MouseEvent -- class

MouseListener
MouseMotionListener -- interfaces

Functions of MouseListener (5)

1. mouseEntered()
2. mouseExited()
3. mousePressed()
4. mouseReleased()
5. mouseClicked()

1. mouseEntered() -- this event occurs when the mouse enters the work area of an application or the panel of an application.
2. mouseExited() -- this event occurs when the mouse exits from the work area of an application or the panel of an application.
3. mousePressed() -- this event occurs when the left mouse or right button is pressed.
4. mouseReleased() -- this event occurs when the left mouse or right button is released.
5. mouseClicked() -- this event occurs when the left mouse or right button is pressed and released fast.

Functions of MouseMotionListener

1. mouseMoved() -- this event occurs when the mouse is moved on the panel without pressing a mouse button.
2. mouseDragged() -- this event occurs when the mouse is moved on the panel by pressing a mouse button.
- besides these 7 functions which come from two interface, other inportant functions are getX() and getY()
- The getX() function reads the x location of the mouse pointer on the screen and getY() reads the y location on screen.
- Finally the panel must be active and capable of listening to mouse event for which two functions are used in the program

addMouseListener()

3. Key events

- The key event occurs when user presses any key on the keyboard
- The key events are very useful in many java applications where keys like arrow keys, pg up, pg dn, home, end and function keys need to be used.
- Other special keys can be function keys f1 to f12 and control, alt, shift etc.
- The key event is handled by one class and one interface

1. KeyEvent class
2. KeyListener interface

- There are three important functions in the KeyListener interface which can be implemented in the program

1. KeyPressed --- This event occurs when a key on keyboard is pressed
2. KeyReleased --- This event occurs when key is released
3. KeyTyped --- This event occurs when the key is pressed and released at once.
4. addKeyListener() --- this function is used to register a control for listening key events.
5. getKeyChar() --- returns the actual character or key value pressed by the user.

5. Window Events

- The window event occurs when the main frame or frame class is modified or activated.
- The window events are handled by one class and one interface

1. WindowEvent class
2. WindowListener interface

There are 7 functions used to handle window events are

1. windowOpened() --- when the window is opened
2. windowClosed() --- when the window is closed

3. windowClosing()   --- when the process of window closing begins
4. windowsMaximized() --- when the window is maximized
5. windowIconified()   --- when the window is minimized
6. windowDeiconified() --- when the window is restored
7. windowDeactivated() --- when the window is clicked outside
8. windowActivated()   --- when the window is clicked inside

- Finally the frame object should be registered and made active to listen to window events by addWindowListener() function.

- All window event are active by default except windowClosing event.

Multithreading in Java

Introduction to Multithreading

- There are 2 types of operating system

1. Single tasking operating system   2. Multitasking OS

- A single tasking OS can perform only one task or one application or one job at a time.
- Unless a task or application is finished or closed it can not start another application.

- The example of Single tasking OS is MS DOS (microsoft disk operating system)
- It was used on the PC before windows OS was developed.

- The multitasking OS allows more than one tasks or applications run simulteniously
- All the applications run in their own memory space and have their own activities.

- The OS allocates separate memory space to run each application
- A user can switch between the running applications by using ALT-TAB
- Example of multitasking OS is Windows.

- A thread is a path of execution or a smallest unit of program that is being executed in the memory.
- Every application has at least one thread which is called "main thread" or "primary thread".
- the main thread is created by the main() function in the program.
- Such application that has only one thread or main thread is called single threaded application or program.
- Besides the main thread many other threads can be created in a program and each thread performs some function or activity.
- Such program which has many threads that exceute almost simulteniously is called "Multi Threaded program".
- These threads are called "Secondary threads" or "worker threads" or background thread.
- Each thread has a life cycle including main thread.
- Every thread takes birth, starts executing and performs the activity and when activity is finished the thread automatically gets destroyed or died.
- The applications of multithreading are computer games, animation programs, almost all internet web sites, downloading application, torrents etc.
- The MT is very important feature of Java and it supports full scale multithreading.
- The MT is implemented in a program by using a class and an interface both of them come from java.util package.

1. Thread class
2. Runnable interface

Thread life cycle

- A thread is controlled by several functions in a program all these functions belong to Thread class.

1. init()   2. start()   3. run() **   4. sleep()
5. stop()   6. suspend ()   7. resume()   8. destroy()

1. init()

- This function performs initailization of thread by creating variables and assiging values.
- This function is automatically executed and need not be written in a program.

2. start()

- This function starts executing a thread and it interanlly CALLs run() function.

3. run()

- It is called by start() function and it may contain the statements to be executed in the thraed.
- All thread activity statements can be written in run() function.
- It is the most importnat and impelented in the program.

4. stop()

- it stops the execution of thread for some time which can be restarted by start() function,
- But the activity starts from begining

5. suspend() and resume()

- The suspend() function also stops a running thread and it is continues by resume() function,.

- The thread resumes from the point where it was suspended.

7. sleep()

- This function also stops a running thread for given period of time in miliseconds
- A soon as the time is over it again starts executing.

Thread.sleep(1000); -- sleep for 1000 milisec or 1 sec.

8. destroy()

- It is also automatically CALLed by the system at the end of program or thread acitivity gets completed.
- It destroys the exceuting thread.
- It need not be written in program.
- The most important function used in multithreading program are
- the init() and destroy() function are very rarely used in a program. They are default function.

1. start()   2. run()   3. sleep()   --- commonly used fucntions
4. suspend()   5. resume().

a. Program to control main() thread. (not a multi threaded program).

There are important function in Thread class

1. currentThread() --- to get the currently executing thread.

ex.
Thread mthread = Thread.currentThread(); -- to get the access of main() thread.

2. setName() --- To give a name to thread .
ex.
mthread.setName("Main Thread");

b. Multithreaded Java program
[Running more than one threads each executing a for loop]

- Create and instantiate the object Thread class
- The class should implement Runnable interface.
- Three functions used in the class 1. start() 2. run() 3. sleep().

1. Text or Banner Animation using Multithreading
2. Graphics Animation in Java

Exception Handling in Java

a. Introduction to EH or Structured Exception Handling (SEH)

- There are 3 types of errors that occur during a program development in java.
1. Syntax Error 2. Linking or Library Error 3. Runtime error

1. Syntax Error

- This error is very common in java program and it occurs during compilation of a program using javac compiler.
- The syntax error occurs due imporpoer syntax in the program like misspelled keywords, missing metacharacters like ; , { } , using variables without declarations etc.
- The syntax errors can be easily removed and they in user's control.
- All syntax errors are program errors and the program runs only if all errors are removed.

2. Linking or Library Error

- These errors occur if the java runtime library files are missing or damaged.
- The linking errors can also be removed by reinstalling jdk.

- All java runtime library files are present in the java installation folder in C drive Program Files.
- The library errors are also in control of the user and can be removed by reisntalling java software.

3. Runtime error

- Even if all syntax errors are removed and library files are intact or properly installed, the runtime errors may occur
- The runtime errors occur during the execution or program runtime.
- The runtime errors are also called as Exceptions.
- The runtime errors occur due to many reasons like reading file from disk which is not present, the array is being processed beyond its size, the obejcts are used in the program without instantiation, accepting a string or non numeric value for numeric input etc.
- The runtime errors or exceptions are not in the users' control and a user can not handle the exceptions because the program is in execution.
- The exceptions are sometimes very dangerous and fatal and they cause serious trouble to the system, by putting the system into hang and computer stops working and may also damage important files in the computer or damage operating system.
- The "exception handling" is a mechanism to display some proper error message and close the program, if a run time error occurs. The system should not be put into unstable state or in trouble.
- The exception handling can be used in a program by adding some additional statements in the program to check the exceptions and stop the program if exception occurs.
- There are three statements using to handle the exception properly

1. try 2. catch 3. finally   4. throws

- The try block is used to write statements that may throw an exception.
- The try block is also called as "Guarded Block" and it throws an exception if some statemnent has runtime error.
- The exception is then received by a catch block

- The catch block stores the exception into an Exception object and may display some error message
- The finally block is not compulsory and may not be written.
- The finally statement performs the activities on execption or without an exception.

```
try
{
  ----
  ----
}
catch(Exception e)
{
  ----
  ----
}
finally
{
   ----
   ----
}
----
----
----
```

- The finally statement is generally not written in the program
- There are many Exception classes which are specific to a particular type of activity or error.
- All exception classes are derived from the base class called Exception.

1. NullPointerException
2. NumberFormatException
3. ArrayIndexOutOfBoundsException
4. FileNotFoundException
5. InterruptedException
6. ClassNotFoundException

7. ArithmeticException
8. IOException
9. SqlException
10.SocketException

1. NullPointerException
- It occurs when the object is not instantiated but used in the program.

2. NumberFormatException
- Occurs when the numeric input is not in proper form.

3. ArrayIndexOutOfBoundsException
- Array is processed beyond its size

4. FileNotFoundException
- When a file is opened in a program which is not present on the disk

5. InterruptedException
- Used in multithreading to interrupt a thread

6. ClassNotFoundException
- When a class is missing

7. ArithmeticException
- when any number is tried to divide by 0

8. IOException
- when the IO device is not ready

9. SqlException (***)
- Occurs in accessing database

10. SocketException (***)
- Occurs in multi user and network program.

Exception handling programs

1. Number format exception
2. Arrayindex out of bounds exception ***
3. Null pointer exception
4. Artithmatic exception

- the getMessage() is a standard and predefined function to print the error message.
- The function belongs to Exception class.

e.getMessage();

printStackTrace() -- this function prints detailed error or it prints error stack.

e.printStackTrace();

- throws statement can be used at the end of the function to throw and catch exception whenever they are occured.
- This statement if written, there is NO need try and catch.

Additional and advanced Features of AWT / Swing

1. Menus

- A menu system is a very imp. component of a GUI application.
- All windows application have menus.
- Any windows application is incomplete without menu.
- The menu system can be created by 2 methods
  1. AWT 2. Swing
- There are 3 menu classes in both the methods

	AWT	Swing
1.	MenuBar	JMenuBar
2.	Menu	JMenu
3.	MenuItem	JMenuItem

- The horizontal bar is the Menubar
- Each object on menubar is Menu
- Each menu may have several vertical MenuItems.

ex.

```
MenuBar mbar;
Menu file, edit, help;
MenuItem fopen, fnew, fexit;
MenuItem ecut, ecopy, epaste;
MenuItem hhelp, habout;

mbar = new MenuBar();
file = new Menu("File");
edit = new Menu("Edit");
help = new Menu("Help");

mbar.add(file); mbar.add(edit); mbar.add(help);

fopen = new MenuItem("Open");
fnew = new MenuItem("New");
fexit = new MenuItem("Exit");

file.add(fopen); file.add(fnew); file.add(fexit);
```

- Finally a setMenuBar() function is used to add menubar on the frame.

```
fr.setMenuBar(mbar);
```

- There are two important functions in menu system

1. setMnemonic() -- to set the keyboard shortcut for the menuitem
2. setToolTipText() -- to display text about the menuitem.

Java stream classes & Java IO

a. Java Stream classes

- A data stream is a flow of data that flows from a source location to target location.
- The source location can be an input device or a memory stream
- The target location can be the memory stream or the outout device.
- There are large number java stream classes and all of them come from java.io package.
- There are main 2 types of data streams and the generic (general) classes or neutral classes. They are not specific classes.

1. InputStream 2. OutputStream (generic)

1. InputStream is the data stream to read data
2. OutputStream is a data stream that sends data to an output device.

- Depending I/O functionality and data streams there are large number classes which are derived from these 2 classes. These are specific classes.

1. DataInputStream / DataOutputStream

2. FileInputStream / FileOutputStream (**)
3. BufferedInputStream / BufferedOutputStream
4. PushBackInputStream / PushBackOutputStream
5. PipedInputStream / PipedOutputStream
6. FilteredInputStream / FilteredOutputStream
7. ByteArrayInputStream / ByteArrayOutputStream

- And there are many other IO classes like

BufferedReader
PrintStream

1. DataInputStream / DataOutputStream

- The DataInputStream is used to create an object to accept data from standard input device or keyword.
- The DataOutputStream is used to create the object to print data on the standard output device or monitor screen.
- The DataInputStream class is generally used to read data from input device or keyboard.

2. FileInputStream / FileOutputStream / RandomAccessFile

- The FileInputStream reads data from the data file stored on the hard disk.
- The file must be already created and containing some data.
- The FileOutputStream class is used to create an object for writing data into a disk file.
- The RandomAccessFile is more powerful and effecient class which can perform the writing of data to a disk file as well as reading data from the disk.
- The object of RandomAccessFile can be created by CALLing a constructor which passes 2 parameters

1. Filename to create or read as a string data
2. File opening mode.

ex.

```
RandonAccessFile rf;
rf = new RandonAccessFile("m:\\mydata\\newfile.txt","rw");
```

- Write data into the file.
- There are many functions to write data into file like

1. writeBytes() --- this function writes the string data into the file.
- A file can be closed using close() function.

```
RandonAccessFile rf;
rf = new RandonAccessFile("m:\\mydata\\newfile.txt","rw");

rf.writeBytes("hello and welcome to Java File Handling\n");
rf.writeBytes("this is second line\n");
rf.writeBytes("End of file");

rf.close();
```

- Read data from the file.
- A readLine() --- this function can be used to read one line at a time from the file.
- This function also reads a string data.

- The RandomAccessFile throws an exception called IOException while performing reading or writing operations.

```
RandonAccessFile rf;
rf = new RandonAccessFile("m:\\mydata\\newfile.txt","rw");

String line;
line = rf.readLine();
while(line!=null)
{

  System.out.println(line);
  line = rf.readLine();
}
```

JFileChooser class -- This class is used to show a file dialog box to select a file to read or write. This class contains two functions

1. getSelectedFile() --- reads the selected file into File object.
2. getName() --- reads the filename from the selected file.
3. showDialog() --- shows the file chooser.

ex.

```
JFileChooser jf = new JFileChooser();
jf.ShowDialog();
String filename = jf.getSelectedFile().getName();
```

Java Class libraries

1. String / StringBuffer / StringTokenizer / Vector / Enumeration / Math
2. Wrapper classes
3. Collection classes

1. String class

- The String class is one of the most popular and commonly used class in Java programs
- String class is an implementation of "non-growable" collection of characters enclosed in double quotes.
- A String object can be created by various methods.

ex.

```
String str = "Hello and welcome";
String str = new String("Hello and welcome");
String str = new String();
```

- The string class contains large number of functions for processing the string.

1. length()   2. equals()   3. equalsIgnoreCase()
3. upperCase()   4. lowerCase() 5. indexOf()   6. lastIndexOf()
7. contains()   8. substring()   9. charAt()
10. concat()   11. trim()   12. split()

1. length() - This function returns the length of a string in int.

prototype : int length();

```
String str = "hello and welcome";
int len = str.length();  [ len = 17 ]
```

2. equals() - this function compares two strings and return true or false. It both string are same then returns true else it returns false.

prototype : boolean equals(String str2);

```
String str = "hello and welcome";
String str2 = "hello and WELCOME";
boolean flag = str.equals(str2);

if(flag==true)
   System.out.println("The strings are same");
else
   System.out.println("The strings are not same");
```

3. equalsIgnoreCase() - it compares two strings by ignoring cases i.e. small or capital letters. It only compares the spellings.

```
String str = "hello and welcome";
String str2 = "hello and WELCOME";
boolean flag = str.equalsIgnoreCase(str2);

if(flag==true)
   System.out.println("The strings are same");
else
   System.out.println("The strings are not same");
```

3. upperCase()   - it converts the string into upper case

```
String str = "hello";
String str2 = str.toUpperCase();
```

4. lowerCase() - converts string into lower case

```
String str = "HELLO";
String str2 = str.toLowerCase();
```

5. indexOf()   - this function searches a character in the string for its first occurance and returns an int value which is the location or index of the character in the string.

```
String city = "Amaravati";
int loc = city.indexOf('a'); [ loc = 2 ]
```

6. lastIndexOf()   - this function searches a character in the string for its last occurance and returns an int value which is the location or index of the character in the string.

```
String city = "Amaravati";
int loc = city.lastIndexOf('a'); [ loc = 6 ]
```

7. contains()   - this function checks if a string is available in another string. If the string is available then returns true and if not available it returns false.

```
String str = "Hello and welcome";
if(str.contains("and"))
    System.out.println("String available");
else
    System.out.println("String not available");
```

8. substring() - this is a very useful function and it is used to extract a part of string into another string, It passes 2 parameters 1. starting index and number of characters from begining

```
String str = "Hello and welcome";
String str2 = str.substring(0,5);
```

9. charAt() -- it returns a single characters at a given index location in a string.

```
String str = "Hello and welcome";
char c = str.charAt(10);  [ c = 'w' ]
```

10. concat() -- it joins two string into a third string

```
String str1 = "Hello";
String str2 = " and welcome";
String str3 = str1.concat(str2);
```

11. trim() -- it removes spaces from the begining or the end of string.
and not from the middle of the string.
 sdmfvsdfs,m

```
String str1 = "  Hello   ";
System.out.println(" Str = " + str.trim());
```

12. split() -- this function converts a sentence into an array of strings. The words in the sentence should be separated by some delimiting character.

```
String str = "This,is,a,sentence,in,java";
String[] sarray = str.split(",");
```

2. StringBuffer / StringBuilder()

- This class is an implementation of "growable string" as against the string class which is non-growable string.
- This class provides facility for maniputlating string by adding data into a stringbuffer object, replacing data and deleting data from the stringbuffer.
- The stringbuffer can grow in size or reduce in size.
- It is very useful class when a string length is not known and should be modified during run time.
- The object of the string buffer class can be created by various methods.
- The StringBuffer class comes from java.util package.

ex.
StringBuffer sbf = new StringBuffer();   -- empty or default constructor
StringBuffer sbf = new StringBuffer("Hello"); -- constructor with a string parameters
StringBuffer sbf = new StringBuffer(200); -- constructor with initial capacity of 20 characters
- The default capacity of StrigBuffer is 16 characters.

- There are many useful functions in this class.

1. capacity()   2. length()   3. append()   4. insert()   5. delete()
6. replace()   7. indexOf()   8. lastIndexOf()   9. reverse()   10. substring()
11. toString()   12. charAt() 13. deleteCharAt()

1. capacity() -- this function returns an int value which is the current capacity of stringbuffer.

int capacity()
ex.
StringBuffer sbf = new StringBuffer("welcome");
int cap = sbf.capacity();

2. length() -- returns the actual length of data in the string buffer.

int length()
ex.
StringBuffer sbf = new StringBuffer("welcome");
int len = sbf.length();

3. append() -- this functions append data at the end of a string buffer. Any data can be appended or added of any data type like string, int, long, float etc. but it gets converted into string.

ex.
StringBuffer sbf = new StringBuffer("welcome");

```
sbf.append("to string buffer ");
sbf.append(35);
sbf.append(60.78f);
sbf.append(" in java ");
sbf.append('a');
```

4. insert() - this function inserts data in the middle of the string buffer. There are 2 parameters passed to this function
1. starting position
2. data to string

```
StringBuffer sbf = new StringBuffer("welcome to string buffer");
sbf.insert(10," java ");
```

5. delete() -- it deletes a part of a string buffer and passes 2 parameters
1. starting location and 2. no of characters from begining of string to delete.

```
StringBuffer sbf = new StringBuffer("welcome to string buffer");
sbf.delete(11,17);
```

6. replace() -- it replaces a part of a string by another string. It also passes 3 parameters
1. staring position
2. ending position from begining
3. string to replace

```
StringBuffer sbf = new StringBuffer("welcome to string buffer");
sbf.replace(8,10,"at");
```

7. indexOf() 8. lastIndexOf()

- These functions are same as the String class function to find the 1st occurance indexOf() and the last occurance lastIndexOf().

8. deleteCharAt()
- It deletes a single character at the given position
sbf.deleteCharAt(10);

3. String Tokenizer

- This class also comes from java.util package.
- This class is used to extract tokens from a string object
- The tokens are the sub-string delimited by some characters

String s = "This is a new string"; delimiter or separator space

- The StringTokenizer object extracts tokens from a string which are delimited by some character.

String s = "This:is:a:new:string";
StringTokenizer stk = new StringTokenizer(s,":");

- The object of the StringTokenizer is created by passing 2 parameters to the constructors 1. String 2. Delimiter

1. countTokens()   - This functions returns total number of tokens in the object. In out case it will return 5

4. Vector

- Vector implements a dynamic array.
- It is similar to ArrayList class (collection class),
- With two differences
1. Vector is synchronized, and
2. Vector contains many legacy (old) and default methods that are not part of the collection classes.
- Vector can be created by 3 methods

- Vector object can be initialized by 4 methods
- A vector object may contain data of any data type.
- The Vector comes from java.util package.

Vector v = new Vector(); -- Default constructor with capacity 10
Vector v = new Vector(5); -- Constructor with initial capacity 5
Vector v = new Vector(5,3); -- Constructor with initial capacity 5 with increment of 3
Vector v = new Vector(collection); (****)

Vector functions

1. add(int index, Object obj) 2. add(Object o)
--- deprecated function outdated and not supported

3. addElement(Object o) 4. capacity() 5. clear() 6. contains()
7. Object elementAt(int index) 8. Enumeration elements() ***
9. equals(Object o) 10. Object firstElement() 11. Object lastElement() 12. indexOf()
13. lastIndexOf() 14. remove(Object o)
15. removeAllElements()

- The Vector class can store data but can not enumerate i.e. visit or acces each element once.

5. Enumeration

- The Enumeration interface defines the methods which can enumerate (access one at a time) the elements in a collection of objects.

```
Enumeration e = v.elements();
while(e.hasMoreElements())
{
  System.out.println("Element in vector = " + e.nextElement());
}
```

B. Wrapper classes

- A wrapper class wraps (encloses) around a data type and gives it an object appearance.
- Wherever, the data type is required as an object, this object can be used.
- Wrapper classes include methods to unwrap the object and give back the data type.
- It can be compared with a chocolate.
- The manufacturer wraps the chocolate with some foil or paper to prevent from pollution.
- The user takes the chocolate, removes and throws the wrapper and eats it.

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

- The wrapper class object if printed it automatically unwraps the data.
ex.

```
int a = 20;
Integer i = new Integer(a); --- wrap the data

int m = i.intValue();  --- unwrap the data
```

Other functions of wrapper classes
1. intValue() -- unwraps an Integer object

2. floatValue() -- unwraps an Float object
3. byteValue() -- unwraps an Byte object
4. longValue() -- unwraps an Long object
5. shortValue() -- unwraps an Short object

Applet programming using Java

Introduction to applets

- Java can be used to develop two types of programs
  1. Java applications 2. Java Applets

- The Java applications are generally of two types i.e. console application mean non-GUI program that runs on b/w screen.
- The Java application can also be GUI based application windows application developed in AWT or Swing.

- The Java applets are "GUI-Only programs" that are window based developed using either awt or swing.
- The Java applets are small programs than applications that do not have main() function like a java application. The applets are comparitively smaller in size than the Java applications.

- The Java applet has init() function or initializtion function. lmost same as the main() function.
- The java applet is embeded or included into HTML document using a special tag <applet> and executed on the web browser.
- The java application executes on dos prompt but applet executes on web browser.
- A java applet does not have a frame but it can have all other GUI controls like Panel, Label, TextField, Button etc.
- A java applet can be developed by using Applet as the base class this class comes from java.applet package

ex.

Differences between Application and Applet

java application	Java Applet
1. Larger in size	Smaller in size
2. Has main() function	does not have main() function init()
3. Runs on command prompt	Embeded into HTML runs in web browser
4. Requires a frame	Does not require frame
5. class can be public or not	class is always public

- all applet functions come from java.applet package.
- an applect class is created by any name but with super class Applet
- awt applet can be created using Aplet class and swing applet can be created using JApllet class.

```
import java.applet.*;

public class MyApplet extends Applet
{
  ---
  ---
}
```

- A java applet can also be develped by using base class JApplet

```
import javax.swing.*;

public class MyApplet extends JApplet
{
  ---
  ---
}
```

Applet life cycle

- An applet get initailized by init() function which is the first function that executes as soon as applet is run.
- An applet may contain a constructor which is executed before init().
- The applet starts executing by using start() function. This function is internally called by init() and may not be written in a program
- A stop() function can be used to stop a running applet.
- destroy() function is executed when an applet goes out scope or when the web page is closed.

The general functions used in an applet program are...

1. init() ***
2. start()
3. stop()
4. destroy()
5. paint() ***
6. repaint() *** -- used in Graphics Programming.

- The paint() function is used to draw graphics on the applet screen using graphical object like line, circle, oval, rectangle, polygon etc.
- The repaint() function to redraw the applet area. It refreshes the applet screen.

1. An applet example for form or UI form.
2. An applet example for Swing data form or UI form.
3. Applet with drawing capability.

Graphics programming using Java

- Java has a very powerful graphics capability and can be used to draw 2D and 3D graphics.
- Java supports all types graphics features requited for drawing.
- All graphics classes come java.awt package.
- The applets are very appropriate programs to draw graphics

- The paint() function is used to write statement for drawing graphics.
- There are 3 important graphics classes common used in drawing program all come from java.awt package

1. Graphics   2. Color   3. Font

1. Grphics class

- It is an abstract class and the object of this class is passed as parameters to the paint() function.

ex.

```
public void paint(Graphics g)
{

}
```

- There are many drawing functions in Graphics class to draw graphicsl objects on the screen.
1. drawLine()
2. drawRect()
3. drawRoundRect()
4. drawOval()
5. drawPolygon()
6. drawString()
7. drawImage
8. fillRect()
9. fillRoundRect()
10. fillOval()
11. fillPolygon()

1. drawLine()

- This function draws a line using 4 co-ordinates x1,y1 and x2,y2

g.drawLine(10,10,100,100);

2. drawRect()

- This function draws a recatngle on the applet screen and it passes 4 parameters x1,y1 and width, height.
- X1 and Y1 are starting parameters and width and height ending parameters of rectangle

g.drawRect(10,10,100,100);

3. drawRoundRect()

- It also draws a rectangle having roundsed corners in stead of sharp corners of recatngle.
- It passes 6 parameters x1,y1,w,h,xradius,yradius

g.drawRoundRect(10,10,100,100,20,20);

4. drawOval()

- This function can be used to draw an oval or a circle.
- It also passes 4 parameters i.e. x1,y1,x2,y2
- If both the x2 and y2 radius are same then it draws a circle.
- If x2 is larger than y2 then horizontal elipse is drawn and vice versa

g.drawOval(10,10,100,100);

5. drawPolygon()

- The polygon is multiside graphical object drawn using set of polypoints.

- The polypoints can be declared an a single dimension int array, each pair consists of x and y point
- The array should contain the last xy ponint as the starting points.
- The drawPolygon() function passes 3 parameters
- x array, yarray and no of popyloints

ex.

```
int xpts[] = { 100,150, 50,100};
int ypts[] = { 100,150, 150,100};
g.drawPolygon(pts,4);
```

6. drawString()

- The drawString() function draws a text or string on the applet
- It passes 3 parameters

1. string to draw
2. x location
3. y location

```
g.drawString("Hello and welcome",100,100);
```

- Font class
- The Font class comes from java.awt package.
- It is used to create a new Font by passing 3 parameters to the constructor

1. Font Name
2. Font Style
3. Font Size

- A setFont() function can be used to set the newly created font.
- there are 3 styles

Font.BOLD, Font.ITALIC,Font.PLAIN, Font.STRIKEOUT

ex.

```
Font f1 = new Font("Arial",Font.BOLD,25);
setFont(f1);
```

Color class

- The Color class is used to create a color object to set as background color or foreground color.
- The background colotr is the complete applet background color
- The foreground color is the drawing color.
- The setBackgound() function can be used to set the background color of applet
- The setForeground() function can be used to set the drawing color.

ex. 1 to set colors
setBackground(Color.YELLOW);
setForeground(Color.RED);

- Java contains a set of fixed color constants which can be accessed by Color class and the name of color in capital letters using dot operator.

creating custom color
- A custom color can be created by using Color object and and instantiating the object by 3 parameters of RBG values in the range of 0 - 255.
- setForeground() and setBackground() can be used to set the new color.

ex.

Color c1 = new Color(100,150,200);
setForeground(c1);

7. drawImage

- to draw image on the graphics screen.
- The function passes four parameters

1. Object of image class
2. x location
3. y location
4. this keyword whish refers current object in menory.

ex.

```
Image img;
public void init()
{
  img = getImage(getCodeBase(),"desert.jpg");
}
```

- The getCodeBase() function is used to specify the directory of the image.
- By default it is the current directory or directory in which applet and HTML file is stored.

```
public void paint(Graphics g)
{
   g.drawImage(img,10,10,this);
}
```

8. fillRect()
9. fillRoundRect()
10. fillOval()
11. fillPolygon()

- All these function draw the graphical object on the screen like rectangle, rounded rectangle, oval and polygon
- Their parameters are same as the draw functions.
- These functions draw the objects and fill them with color.
- The color can be set by setColor() function.

ex.

```
setBackground(Color.YELLOW);
setForeground(Color.RED);
g.setColor(Color.PINK);
g.drawOval(100,100,300,300);
```