

# **JAVA HANDWRITTEN**

## **NOTES**

**HAPPY LEARNING!**

# Java Evolution and basics of java

## 1. Java evolution and basics of java -

1.1 Creation of java , java features , The java Buzzwords , simple java program

1.2 Java virtual machine , constant , variable , Data types , operators and expressions

1.3 Decision making and branching , Decision making and looping

1.4 Arrays , One dimensional array , creating an array , two dimensional array

## • Creation of java -

Java is a modern object oriented computer programming language.

James Gosling led a team of researches to create language that would allow consumer electronic devices to communicate with each other

Work on language began in 1991  
Java was first release in 1995

Code in other languages is first translated by a compiler into instructions for a specific type of computer

The java compiler instead turns code into something called Bytecode which is then interpreted by software called java runtime environment (JRE) or java virtual machine.

JRE act as virtual computer that interprets bytecode and translates it for the host

computer because of this java code can be written the same way for many platforms (Write once run everywhere)

- Features of Java or Java Buzzwords

1. Simple
2. Object oriented
3. Distributed
4. Interpreted
5. Robust
6. Secure
7. Architecture neutral
8. Portable
9. High performance
10. Multithreaded
11. Dynamic

- A simple java program.

```
class HelloWorld  
{
```

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

- Variable - Data container that stores the data value during java program execution

int a;

a = 10;

## 1 Local variables -

Local variables are declared inside the body of a method.

## 2 Instance variables -

Instance variables are defined without the STATIC keyword. They are defined outside a method declaration. They are object specific and are known as instance variables.

## 3 Static variables -

Static variables are initialized only once at the start of the program execution.

These variables should be initialized first, before the initialization of my instance variables.

class Demo

```
{  
    static int a = 1;      // static variable  
    int data = 99;        // instance variable
```

void method()

```
{  
    int b = 90;           // Local variable  
}
```

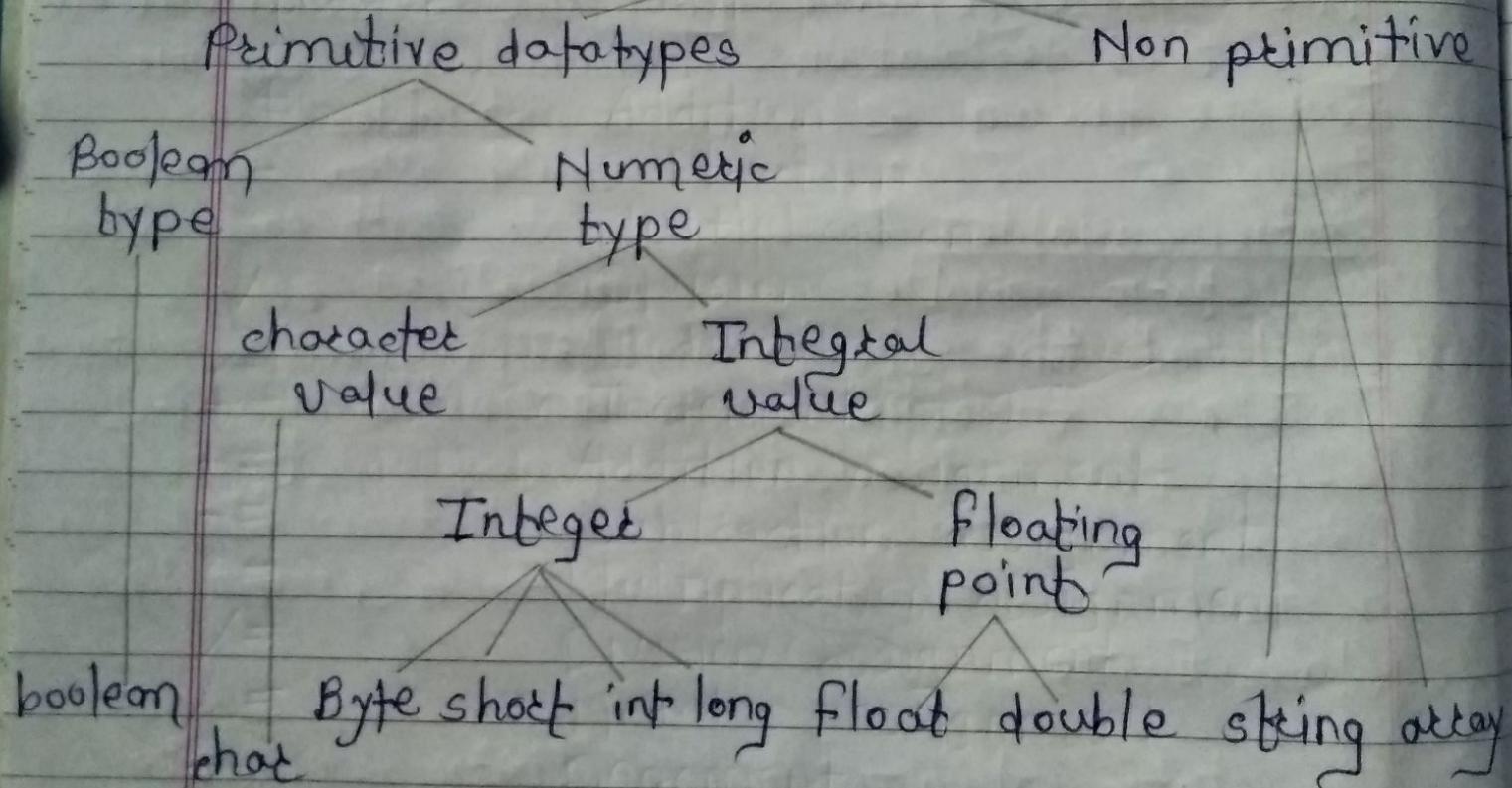
- Data Types -

Data types in java are defined as specifiers that allocate different sizes and types of values that can be stored in a variable or an identifier.

- Java has two categories of data :

1. Primitive Data type - Boolean, char, int, short, byte, long, float and double.
2. Non-Primitive data type - string, array

### Data types in java



## Constant -

A constant is a variable whose value can't change once it has been assigned.

Java doesn't have built-in support for constants.

To define a variable as a constant, we just need to add the keyword "final" in front of variable declaration.

```
final float pi = 3.14f;
```

above statement declares the float variable "pi" as a constant with a value of 3.14f. We cannot change value of pi at any time in program.

Later if we try to do that using statement like "pi = 5.25f", java will throw errors at compile time itself.

## Operators and expressions -

### Java expressions -

Expressions are constructed from operands and operators.

The operators of an expression indicate which operations to apply to the operands.

The order of evaluation of operators in an expression is determined by the precedence and associativity of operators.

## Operator -

An operator is a special symbol which indicates a certain process is carried out.

Operators in programming languages are taken from mathematics. Programmers work with data. The operators are used to process data. An operand is one of the inputs of an operator.

## Types of operators -

### Arithmetic operators

Operator	Result
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement

- Operands of arithmetic operators must be numeric type. That type in java is a subset of int.

## 2. Bitwise Operators

These operators can be applied to the integer types, long, int, short, char and byte. These operators act upon individual bits of their operands.

Operator	Result
<code>~</code>	Bitwise unary not
<code>&amp;</code>	Bitwise AND
<code>^</code>	Bitwise OR
<code>&gt;&gt;</code>	Bitwise exclusive OR
<code>&gt;&gt;&gt;</code>	Shift right
<code>&lt;&lt;</code>	Shift right zero fill
<code>&lt;&lt;=</code>	Shift left
<code>&amp;=</code>	Bitwise AND assignment
<code> =</code>	Bitwise OR assignment
<code>^=</code>	Bitwise exclusive OR assignment
<code>&gt;&gt;=</code>	Shift right assignment
<code>&gt;&gt;&gt;=</code>	Shift right zero fill assignment
<code>&lt;&lt;=</code>	Shift left assignment

## 3. Relational Operator -

The relational operator determine the relationship that one operand has to the other.

Operator	Result
<code>==</code>	Equal to
<code>!=</code>	not equal to
<code>&gt;</code>	Greater than
<code>&lt;</code>	Less than
<code>&gt;=</code>	Greater than or equal to

&lt;=

Less than or equal to

#### 4. Assignment operator -

Format : var = expression

eg. `int x,y,z;`

`x=y=z=100`

The fragment sets variable x,y,z to 100 using assignment operator.

#### 5. Conditional operator -

It's a ternary operator.

General form -

`expression1 ? expression2 : expression3`

If expression 1 evaluates as true then expression 2 is evaluated otherwise expression 3 is evaluated.

## • Decision making and branching.

- 1) if - IF can be used to route program execution

General form : if (condition)  
 statement 1;  
 else  
 statement 2;

- 2) Nested if - A nested if statement that is the target of another if or else.

General form : if (condition)  
 if (condition)  
 statement 1;  
 else  
 statement 2  
 else  
 statement n;

- 3) The if-else-if ladder -

A common programming construct that is based upon a sequence of nested ifs is the if-else-if ladder.

General form : if (condition)  
 statement;  
 else if (condition)  
 statement;  
 :  
 else  
 statement;

## Java switch statement -

Switch statement tests the equality of a variable against multiple values.

General form -

```
switch (variable / expression)
{
```

```
    case value1 :
```

```
        // statement of case1  
        break;
```

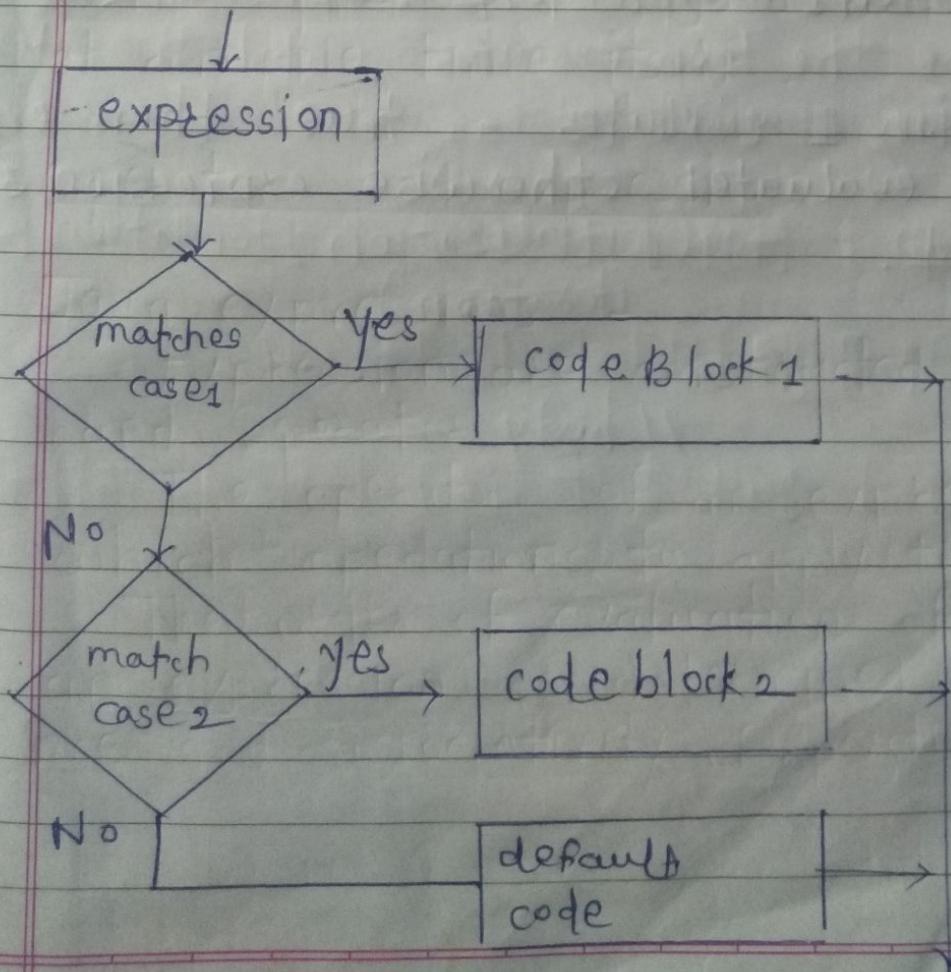
```
    case value2 :
```

```
        // statement of case2  
        break;
```

```
    default :
```

```
        // default statement
```

```
}
```



- Decision making and looping .

1. while - It repeats a statement or block while its controlling expression is true.

General form: `while (condition)`

```
{  
    // body of loop  
}
```

2. do - while -

It is same as while loop but it executes the body of loop at least once.

General form: `do {`

```
    // body of loop  
} while (condition)
```

3. for - here the loop executes until the iteration condition becomes false.

General form: `for (initialization; condition;  
 iteration)`

```
{  
    // body  
}
```

## • Arrays

Array is a group of homogeneous data type. A specific element in array is accessed by its index.

### 1. One dimensional array

General form - type var-name[];

type declares base type of the array. base type determines the data type of each element that comprises the array.

e.g. int month-days[];

The statement declares an array named month-days with type "array of int".

Now the value of month-days is set to null. To link month-days with an actual, physical array of integers, we need to allocate it using new and assign it to month-days.

General form : array-var = new type [size];

e.g. month-days = new int[12];

month-days will refer to an array of 12 integers. Further all elements in array will be initialized to zero.

- accessing element by using index -

`month_days[1] = 28;`

The statement assigns the value 28 to 2<sup>nd</sup> element of month\_days.

2. Multidimensional array -

Multidimensional arrays are arrays of arrays

General form:

`int twoD[][] = new int[4][5];`

This allocates a 4 by 5 array and assigns it to twoD.

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]

left  
index  
determines  
row

Right index determines column

## • Programming

Java programming can be simplified in 3 steps :

1. Create the program by typing it into a text editor and saving it to a file - `HelloWorld.java`
2. Compile it by typing "`javac HelloWorld.java`" in terminal window.
3. Execute (or run) it by typing "`java HelloWorld`" in terminal window.

Program -

```
/* This is a simple Java program.  
FileName : "HelloWorld.java". */  
class HelloWorld  
{  
    // program begins with a call to main()  
    // prints "Hello, World" to terminal window.  
  
    public static void main( String args[] )  
    {  
        System.out.println( "Hello, World" );  
    }  
}
```

Output

Hello, World

- 3 primary components -

- 1 Class definition -

This line uses the keyword `class` to declare that a new class is being defined.

```
class HelloWorld
```

`HelloWorld` is an identifier that is the name of the class. The entire class definition, including all of its members, will be between the opening curly brace `{` and closing curly brace `}`.

- 2 main method -

Signature of main method -

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

`public`: So that JVM can execute the method from anywhere.

`static`: Main method is to be called with no object.

`void`: Main method doesn't return anything.

`main()`: Name configured in JVM.

`string []`: The main method accepts a single argument: an array of elements of type `string`.

Main method is entry point for your application and will subsequently invoke all the other methods required by your program.

3. `System.out.println("Hello, World");`

This line outputs the string "Hello, World" followed by a new line on screen. Output is actually accomplished by the built-in `println()` method. `System` is a pre-defined class that provides access to system, and `out` is the variable of type `output stream` that is connected to the console.

4. Comments -

They can either be multiline or single line comments.

```
/* This is simple java program.  
Call this file "HelloWorld.java". */
```

Multiline comment: `/* ..... */`

Singleline comment: `//.....`

• Packages in java -

Package in java is a mechanism to encapsulate a group of classes, sub packages and interfaces.

Packages are used for -

1. Preventing naming conflicts
2. Making searching and usage of classes, interfaces, enumeration and annotations easier.
3. Providing controlled access: protected and default have package level access control.
4. Packages can be considered as data encapsulation.

lation (or data-hiding).

- A package is a container of a group of related classes where some of the classes are accessible are exposed and others are kept for internal purpose.

Packages and directory structure are closely related.

If package name is college.faculty.jp then there are 3 directories college, faculty and jp such that jp is present in faculty and faculty is present in college.

Directory college is accessible through CLASSPATH variable i.e path of parent directory of college is present in CLASSPATH.

- Subpackage -

packages that are inside another package are the subpackages.

These are not imported by default, they have to be imported explicitly.

e.g. import java.util.\*;

util is subpackage created inside java package.

- Static import -

Static import is a feature that allows members (fields and methods) def

in a class as public static to be used in java code without specifying the class in which the field is defined.

```
import static java.lang.System.*;
class staticImportDemo
```

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

- `import java.util.Vector;`  
 package class      subpackage      class  
 ↑               ↑               ↑
- `import java.util.*;`  
 package subpackage  
 ↑               ↑               ↑  
 All the classes in util subpackage
- `import java.util.System.*;`  
 static      package subpacky class  
 static import      ↑               ↑  
 all the fun in System class

## 2. Classes, objects and Methods

- 2.1 Defining a class , field declaration , Methods declaration , creating object , accessing class members .
- 2.2 Constructors , method overloading , Nesting of methods .
- 2.3 Inheritance : Extending a class (Defining a subclass constructor , Multilevel inheritance , hierarchical inheritance )
- 2.4 Overriding methods , final keyword (variable and methods , final variable and methods , final classes , finalize methods )
- 2.5 Abstract methods and classes , Methods with var args , visibility control , (Public access , friend access , protected access , private access , private protected access )
- 2.6 Vectors , Wrapped classes , Enumerated types , Annotations

- Class -

A class is blueprint for creating objects template for an object & object is instance of

- To create a class we , use the keyword class

Create a class named "Myclass" with a variable x :

```
public class Myclass
```

```
{  
    int x = 5;  
}
```

## • Field Declaration -

A java field is a variable inside a class.  
eg. a class representing an employee, the Employee class might contain following fields:  
name, position, salary, hiredDate

It can be defined as -

```
public class Employee {  
    String name;  
    String position;  
    int salary;  
    Date hiredDate;  
}
```

## • Methods Declaration -

block of code

Java's way of implementing subroutine.

General form of a method -

type name (parameter-list)

{

// body of method

}

type specifies type of data returned by method.

If a method does not return a value, its return type must be void.

name of the method is specified by name.  
this can be any legal identifier other than  
those already used by other items within

current scope.

The parameter list is a sequence of type and identifier pairs separated by commas. Parameters are essentially variables that receive the value of arguments passed to the method when it is called. If the method has no parameters, then the parameter list will be empty.

Methods having return type other than void return a value to calling routine using foll' form of return statement.

return value;

Here value is the value returned.

## Creating Objects

Object is an instance of a class.  
class declaration only creates a template,  
it does not create an actual object.

To actually create a object syntax of statement-

class.name obj-name = new constructor\_of\_class();

e.g. Box myBox = new Box();

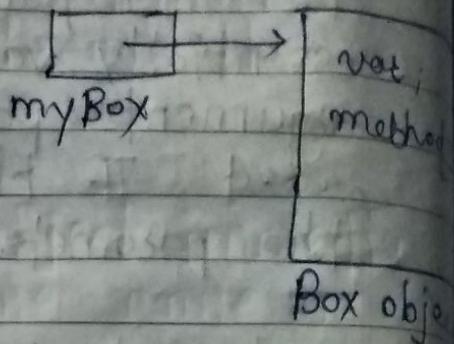
Here now myBox is an instance of Box.  
new is a keyword who allocates memory for  
object and returns a reference to it.

address in memory of object allocated by new.

Box myBox;

Null  
myBox

myBox = new Box();



- Object declaration is a two step process
- Accessing class members

We have a class Box as follow:

class Box

{

    double width;  
    double height;  
    double depth;

}

class BoxDemo2

{

    public static void main(String args[])

    {  
        Box mybox1 = new Box(); double vol;  
        mybox1.width = 10;  
        mybox1.height = 20;  
        mybox1.depth = 15;

        vol = mybox1.width \* mybox1.height \* mybox1.

```
System.out.println("Volume is :" + vol);
```

{ }

Output: Volume is : 3000

- Class members are accessed using dot operator.  
(instance variable)

Dot operator (.) links the name of the object with the instance variable.

For eg. to assign the width variable of myBox the value 10 following statement is used.

```
myBox.width = 100;
```

This statement tells the compiler to assign the copy of width that is contained within myBox object the value 10.

We use (.) dot operator to access both instance variable and methods.

## • Constructors -

Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of constructor.

A constructor initializes an object immediately upon creation. It has same name as class in which it resides.

## • Types of constructors -

### 1. Default constructor -

If we don't apply any constructor in our class the java compiler inserts default constructor into your code on your behalf.

You will not see default constructor in your source code (.java file) as it is inserted during compilation and present in bytecode (.class)

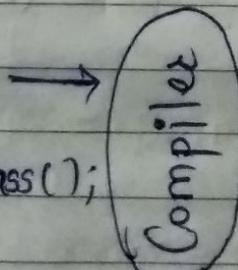
```
public class MyClass
```

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

```
{  
    MyClass obj = new MyClass();
```

```
---
```

```
}
```



```
public class MyClass
```

```
{  
    MyClass()
```

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

```
{  
    MyClass obj = new
```

```
MyClass()
```

```
}
```

```
---
```

If we implement any constructor then we no longer receive a default constructor from Java compiler.

## 2 No argument constructor -

Constructor with no argument is known as no-arg constructor. The signature is same as default constructor, however body can have any code unlike default constructor where the body of constructor is empty.

ex- class Demo

```
{ public Demo()
```

```
{
```

```
    System.out.println("This is a no argument constructor");
```

```
}
```

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

```
        new Demo();
```

```
}
```

```
}
```

OP -

This is a no argument constructor

## 3. Parameterized Constructor -

Constructor with argument or parameter is known as Parameterized constructor.

Q:

```
public class Employee {  
    int empId;  
    String empName;  
  
    // Parameterized constructor  
    Employee(int id, String name) {  
        this.empId = id;  
        this.empName = name;  
    }  
  
    void info() {  
        System.out.println("Id: " + empId + " Name:  
                           " + empName);  
    }  
  
    public static void main(String args[]) {  
        Employee obj1 = new Employee(10245, "Ram");  
        Employee obj2 = new Employee(10246, "Riya");  
        obj1.info();  
        obj2.info();  
    }  
}
```

O/P:

```
Id : 10245 Name : Ram  
Id : 10246 Name : Riya
```

- Method overloading - (Compile time poly)  
Feature that allows a class to have more than one method having same name, if their argument list are different.

Cases -	1) add (int, int)	[No of parameters]
	add (int, int, int)	
2)	add (int, int)	[Data type of parameters]
	add (int, float)	
3)	add (int, float)	[Sequence of data types]
	add (float, int)	

int add (int, int)  
float add (int, int) } Invalid case of  
method overloading

eg. - 1) Diff. no of parameters in arg. list

class DisplayOverloading

```
{ public void disp(char c)
{
```

```
    System.out.println(c); }
```

```
{ public void disp(char c, int num)
```

```
    System.out.println(c + " " + num); }
```

```
}
```

class Sample

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

```
    DisplayOverloading obj = new DisplayOverloading(); }
```

obj.disp('y');  
obj.disp('y', 69);

2) Diff. in datatype of parameter

```
class Display
{
    public void disp(char c)
    {
        System.out.println(c);
    }

    public void disp(int c)
    {
        System.out.println(c);
    }
}

class Sample2
{
    public static void main(String args[])
    {
        Display obj = new Display();
        obj.disp('y');
        obj.disp(69);
    }
}
```

③ Sequence of Datatype of arguments.

class Display

{ public void disp(char c, int num)

{ System.out.println ("First");

} public void disp(int num, char c)

{ System.out.println ("Second");

}

class Sample3

{

public static void main(String args[])

{ Display obj = new Display();

obj.disp('y', 69);

obj.disp(69, 'y');

}

}

Type promotion -

When data type of smaller size is promoted to data type of bigger size, it's a type promotion.

Type promotion table -

byte → short → int → long → double

short → int → long

int → long → float → double

float → double

long → float → double

## • Nesting of method in java

Program -

```
class Nest_Method
```

```
{
```

```
    int a,b;
```

```
    Nest_Method (int m,int n)
```

```
{
```

```
    a = m;
```

```
    b = n;
```

```
}
```

```
    int greater()
```

```
{
```

```
        if(a>b)
```

```
        { return a; }
```

```
        else
```

```
            return b;
```

```
}
```

```
    void Display()
```

```
{
```

```
        int great = greater();
```

```
        System.out.println("The Greatest no = "  
                           +great);
```

```
}
```

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

```
{
```

```
    Nest_Method obj = new Nest_Method  
                    (80,90);
```

```
    obj.Display();
```

```
}
```

A method of a class can be called only by an object of that class using the dot operator. So, there is an exception to this. A method can be called by using only its name by another method of same class that is called Nesting of methods.

### Inheritance -

The process by which one class acquires the properties of another class is called inheritance.

The aim of inheritance is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from another class.

#### Child class -

The class that extends the features of another class is known as child class, sub class or derived class.

#### Parent class -

The class whose properties and functionalities are used by another class is known as parent class, super class or Base class.

class xyz extends ABC

{  
}

Here xyz is child class and ABC is parent

Class xyz is inheriting the properties of ABC class.

- Member declared as private in super class can't be inherited.
- A superclass variable can access a subclass object.
- When a reference to a subclass object is assigned to superclass reference variable, we will have access to those part of object defined by superclass.
- Using super:

Whenever a subclass needs to refer its immediate superclass it can do so by keyword super.

super has two general forms -

1. calls superclass constructor Super(at a list)
2. used to access a member of superclass that has been hidden by a member of a subclass super.member.

Example.

1 class A

```
{ private private int,  
    ↗ int a; b; int result;  
    A(){}  
    A (int x, int y)
```

q = x; b = y;  
} void sum() { result = a + b;  
System.out.println("sum = "  
class B extends A      + result); }  
{  
int c;  
B(int x, int y, int z)  
{ super(x, y);  
a = x; b = y; c = z;  
}  
}

class Practical - eg

{  
public static void main(String args[])  
{  
B subobj = new B(10, 20, 30);  
subobj.sum();  
System.out.println("Value of c : " + subobj.  
}  
}

2. class A

{  
int i;

class B extends A

{  
int i; // hides the i in A class  
B(int a, int b)

{  
(this.i = a) super.i = a // in A  
i = b

}  
void show()

{  
System.out.println("i in superclass: " + super.i);  
System.out.println("i in subclass: " + i);

3  
class UseSuper

{  
public static void main (String args[])

B subOb = new B(1, 2);

subOb.show();

}

## • Creating a Multilevel hierarchy

It is perfectly acceptable to use a subclass as superclass of another.

class A

{

int a, b;

A (int x, int y)

{  
    a = x; b = y;

}

class B extends A

{

int c;

B (int x, int y, int z)

{  
    super(x, y);  
    c = z;

}

}

class C extends B

{

int d;

C (int x, int y, int z, int w)

{

    super(x, y, z);  
    d = w;

}

} class Multilevel

{

public static void main (String args [] )

```
{   c ob = new c (10, 20, 30, 40);  
    System.out.println ("Value of a : " + ob.a );  
    System.out.println ("Value of b : " + ob.b );  
    System.out.println ("Value of c : " + ob.c );  
    System.out.println ("Value of d : " + ob.d );  
}
```

- Constructors are called in order of derivation

// Demonstrates when constructors are called

// create a super class  
class A

```
{ A () {
```

```
    System.out.println ("Inside A's constructor ");  
}
```

// create subclass by extending A  
class B extends A

```
{ B () {
```

```
    System.out.println ("Inside B's constructor ");  
}
```

// create subclass by extending B  
class C extends B

```
{ C () {
```

```
    System.out.println ("Inside C's constructor ");  
}
```

class CallingCons

{

    public static void main(String args[])

{

        C c = new C();

}

}

Output :

Inside A's constructor

Inside B's constructor

Inside c's constructor.

Method Overriding - (Runtime Polymorphism)

When a method in subclass has same name and type signature as a method in its superclass, then method in subclass is said to override the method in superclass

Method in superclass - Overridden method

Method in subclass - Overriding method

// Method Overriding

class A

{

    int i, j;  
    A (int a, int b)

    { i = a; j = b; }

    void show()

    { // Overridden method

        System.out.println("i and j : " + i + " " + j);

}

class B extends A

```
int k;  
B(int a, int b, int c)  
{ super(a, b);  
k = c;
```

```
} void show() // overriding method
```

```
{ System.out.println("k : " + k);
```

}

```
class override
```

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

```
B subob = new B(1, 2, 3);
```

```
subob.show(); // calls show in B
```

Output -

k : 3

- When an overridden method is called from within a subclass, it will always refer to the version of that method defined by subclass. The version defined by superclass will be hidden.

Method overriding is used when child class wants to give its own implementation to a method defined in parent class.

- If we wish to access a superclass version of overridden method we can do so by using keyword super.

class B extends A

{

    int k;  
    B ( int a, int b, int c )

{

        super(a,b);  
        k = c;

}

    void show()

{

        super.show();

        System.out.println(" k : " + k);

    }

}

Output:

i and j : 1 2

k : 3

- Here super.show() calls the superclass version of show().

Method overloading is an example of compiletime polymorphism where method overriding is an example of runtime polymorphism.

- Polymorphism - Refining single action in different ways

- Polymorphism is a capability of method to do different things based on the object that it's acting upon.
- One interface and multiple implementation.
- Dynamic method Dispatch

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at runtime, rather than compile time.

// Dynamic method dispatch

class A

{

void callme()

{

System.out.println("Inside A's callme method")

}

class B extends A

{

void callme()

{

System.out.println("Inside B's callme method")

}

class C extends A

{

void callme()

{

System.out.println("Inside C's callme method")

}

B

```
class Dispatch
{
    public static void main(String args[])
    {
        A a = new A(); // Object of type A
        B b = new B(); // Object of type B
        C c = new C(); // Object of type C
```

```
A t; // obtain a reference of type A
t = a; // t refers to a object
t.callme(); // calls A's version of callme()
```

```
t = b;
t.callme();
```

```
t = c;
t.callme();
```

{

}

Output -

Inside A's callme method  
Inside B's callme method  
Inside C's callme method

- Version of overridden method to be executed is determined by the type of object being referred to at the time of call.

```
t = a; // t refers to an A object
t.callme(); // calling A's version of callme()
```

- Vector -

Vector is like dynamic array which can grow or shrink its size.

Unlike array we can store m-number of elements in it as there is no size limit.

Vector class is found in java.util package and implements the List interface, so we can use all the methods of List interface here.

Vector is similar to ArrayList but with two differences -

1. Vector is synchronized
2. Java vector contains many legacy methods that are not the part of a collections framework.

- List of vector class methods -

Method	Description
1. add()	It is used to append the specified element in the given vector.
2. addAll()	It is used to append all the elements in specified collection to end of this vector.
3. addElement()	It is used to append the specified component to end of this vector. It increases the vector size by one.

- 4. `Capacity()` It is used to get the current capacity of vector
- 5. `contains()` It returns true if vector contains the specified element.
- 6. `insertElementAt()` It is used to insert the specified object as a component in the given vector at the specified index.
- 7. `isEmpty()` It is used to check if this vector has no components.
- 8. `remove()` It is used to remove the specified element from vector.
- 9. `removeAll()` It is used to delete all elements from vector that are present in specified collection.
- 10. `removeElement()` It is used to remove the first (lowest indexed) occurrence of argument from vector.
- 11. `removeElementAt()` used to delete the component at the specified index.
- 12. `size()` The method returns the number of components in this vector.

- Wrapper class -

Wrapper class is a class that encapsulates a primitive type within an object.

We can wrap a primitive value into a wrapper class object.

- Need of Wrapper class

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method. (Because primitive types are passed by value)
2. The classes in `java.util` package handles only objects and hence wrapper classes help in this case also.
3. Data structure in collection framework such as `ArrayList` and `Vector` store only objects (refer types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

Primitive Data types

Corresponding wrapper class

char

Character

byte

Byte

short

Short

int

Integer

long

Long

float

Float

double  
boolean

Double  
Boolean

## Boxing and Unboxing -

Conversion of primitive types of object of their corresponding wrapper classes is known as autoboxing.  
 eg. conversion of int to Integer, long to Long,  
 double to Double etc.

Unboxing is the reverse process of autoboxing.  
 Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing.

eg. conversion of Integer to int, Long to long,  
 Double to double, etc.

## WAP on Wrapped class [Boxing and unboxing].

```
import java.util.*;
public class WrappedDemo
```

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

// creation of primitive types

byte a = 1;

int b = 5;

float c = 0.4f;

double d = 505.5;

char e = 'a';

// convert into wrapper objects [Boxing]

```
Byte obj1 = Byte.valueOf(a);  
if (obj1 instanceof Byte)  
{
```

```
    System.out.println("An object of Byte is created");  
}
```

```
Integer obj2 = Integer.valueOf(b);  
if (obj2 instanceof Integer)  
{
```

```
    System.out.println("An object of Integer is created");  
}
```

```
Float obj3 = Float.valueOf(c);  
if (obj3 instanceof Float)  
{
```

```
    System.out.println("An object of Float is created");  
}
```

```
Double obj4 = Double.valueOf(d);  
{
```

```
    System.out.println("An object of Double is created");  
}
```

```
Character obj5 = Character.valueOf(e);  
if (obj5 instanceof Character)  
{
```

```
    System.out.println("An object of Character is created");  
}
```

// convert into primitive types [unboxing]

```
byte f = obj1.byteValue();
System.out.println("The value of f: " + f);
```

```
int g = obj2.intValue();
System.out.println("The value of g: " + g);
```

```
float h = obj3.floatValue();
System.out.println("The value of h: " + h);
```

```
double i = obj4.doubleValue();
System.out.println("The value of i: " + i);
```

```
char
double j = obj5.charValue();
System.out.println("The value of j: " + j);
```

{

}

- Enumerated types -

Enumeration is a list of named constants.

```
enum Color
```

{

```
    red, yellow, green, violet, pink, blue
```

The identifiers red, yellow and so on are called enumeration constants. Each is implicitly declared as public static final member of Color.

- Even though enumeration defines a class type we cannot instantiate an enum using new. We can declare an enumeration variable eg. Color vat;

- Following program demonstrates how enum works.

// an enumeration of color names

enum color

{

red, pink, yellow, white, green

class EnumDemo

{

public static void main(String args[])

{

color vat;

vat = color.pink;

// Output an enum value

System.out.println("Value of vat : " + vat)

vat = color.green;

// Compare two enum values

if (opt == color.green)

System.out.println("vat contains green")

// Use an enum to control switch statement.

switch(var)

{

case red :

    System.out.println("It is red");

    break;

case pink :

    System.out.println("It is pink");

    break;

case yellow :

    System.out.println("It is yellow");

    break;

case white :

    System.out.println("It is white");

    break;

case green :

    System.out.println("It is green");

    break;

}

}

}

# Introduction to Strings, Interfaces & Packages

- 3.1 Special string operations, Character Extraction, string comparison, searching strings, Modifying a string, Data conversion using value of(), string Buffer
- 3.2 Commandline arguments, static Members
- 3.3 Interfaces : Defining interfaces, Extending interfaces, Implementing interfaces, accessing interface variable.
- 3.4 Packages : Java API Packages, using system packages, naming conventions, creating packages. Accessing a package, Using a package, adding class to a package, hiding classes, static Import

In java string is basically an object that represents sequence of char values. An array of character is a string.

e.g. char ch[] = {'G', 'P', 'P', 'U', 'N', 'E'};  
String s = new String(ch);

This is same as :

String s = "GPPUNE";

Java string class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

In java string is an object that represents a sequence of characters.

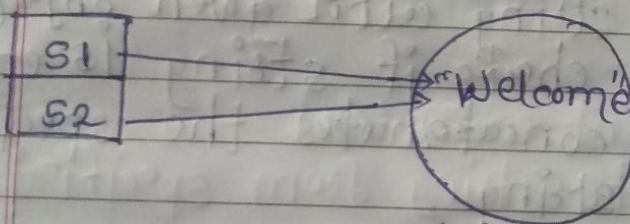
The `java.lang.String` class is used to create a string object.

- Two ways of creating string object -

- By string literal -

`String s1 = "welcome";` // creates new instance s1 in string constant pool.

`String s2 = "welcome";` // Doesn't create a new instance. will return reference to same instance s1.



String objects are stored in special memory called as string constant pool.

- By new keyword -

`String s = new String("Welcome");`

// creates two objects and one reference

JVM will create a new string object in normal (non-pool) heap memory and literal "Welcome" will be placed in the string constant pool. Variable s will refer to object in heap (non-pool).

## • Java String class methods

Java `long.String` class provides many useful methods to perform operations on sequence of char values.

Method	Description
1. <code>int length()</code>	returns string length
2. <code>String substring (int beginIndex)</code>	returns substring for given begin index
3. <code>String substring (int beginIndex, int endIndex)</code>	returns substring for given begin index and end index
4. <code>boolean equals (Object another)</code>	checks the equality of string with given object
5. <code>boolean isEmpty()</code>	checks if string is empty
6. <code>String concat (String str)</code>	concatenates the specified string
7. <code>String replace (char old, char new)</code>	replaces all occurrences of specified char value
8. <code>int indexOf (int ch)</code>	returns the specified char value index
9. <code>int indexOf (int ch, int fromIndex)</code>	returns specified char value index starting with given index
10. <code>String toLowerCase()</code>	returns a string in lowercase
11. <code>String toUpperCase()</code>	returns a string in uppercase

## Java StringBuffer class

The `java.lang.StringBuffer` class is a mutable, thread-safe, mutable sequence of characters.

A `StringBuffer` is like a `String` but can be modified. It contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.

Every `StringBuffer` has a capacity.

### Constructors in `StringBuffer` class.

#### 1. `StringBuffer()`

This constructs a single `StringBuffer` with no characters in it and an initial capacity of 16 characters.

#### 2. `StringBuffer(CharArray seq)`

This constructs a `StringBuffer` that contains the same characters as specified `CharArray`.

#### 3. `StringBuffer(int capacity)`

This constructs a `StringBuffer` with no characters in it and the specified initial capacity.

#### 4. `StringBuffer(String str)`

This constructs a `StringBuffer` initialized to the contents of the specified `String`.

- Methods in StringBuffer class

1. append(string s)

is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double), etc.

2. insert(int offset, string s)

used to insert the specified string with this string at the specified position.

3. replace(int startIndex, int endIndex, string str)

used to replace the string from specified startIndex and endIndex.

4. delete(int startIndex, int endIndex)

used to delete the string from specified startIndex and endIndex.

5. reverse()

used to reverse the string

6. capacity()

used to return the current capacity.

7. substring(int beginIndex)

used to return the substring from specified beginIndex.

## • Commandline Argument

The commandline argument is the argument that passed to a program during runtime. It is the way to pass argument to the main method in java. These arguments stored into the String type args parameter which is main method parameter.

## • Syntax to provide commandline arguments -

java Programname args args

## • Sample program -

```
public class Hello
```

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

// checks if length of args array is greater than 0

```
if (args.length > 0)
```

```
{
```

```
System.out.println("The command line  
arguments are : ");
```

// iterating the args array and printing the command line arguments

```
for (int i = 0; i < args.length; i++)
```

```
{
```

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

```
}
```

```
}
```

else

System.out.println("No command line  
argument found");

}

Executing the program as java Hello Yashshai

OUTPUT : No command line argument of extra

The command line arguments are :  
Yashshai

# Multithreaded Programming Managing error and exception

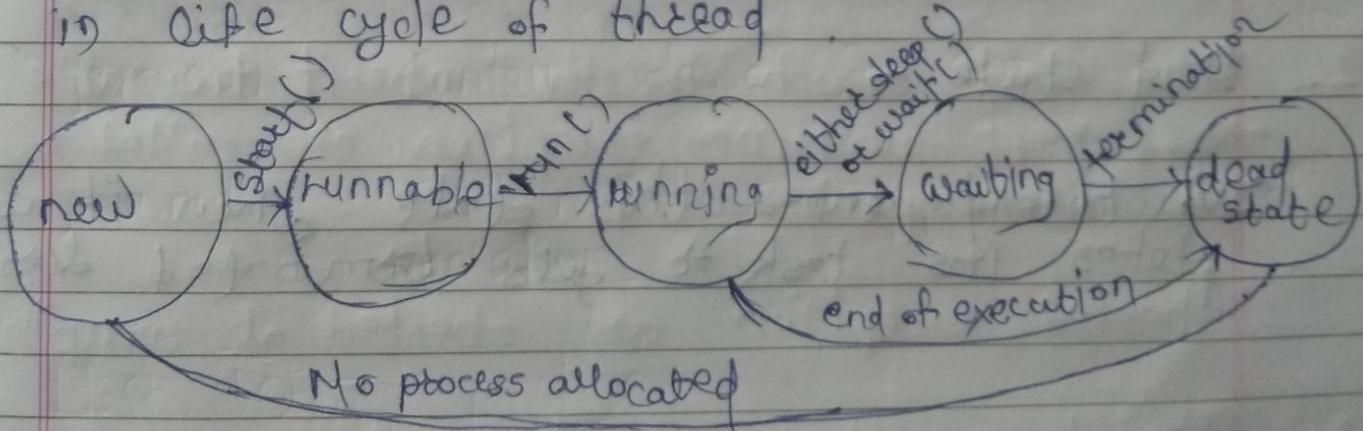
- Thread - small program to be executed

- Two ways for making thread -

1. By extending thread class (from java.lang package)
2. by implementing runnable interface

- Life cycle of a thread -

1. A thread goes through various stages in its life cycle like a thread is born, starts, runs and finally dies.
2. All the above stages forms various states  
i) Life cycle of thread



new - a new thread begin its lifecycle in this state

It remains in this state until program starts thread.  
also called as 'born' thread.

## Runnable state -

After a newly born thread is started the thread becomes runnable or ready to execute its task.

## Running -

It is actually executing the task whenever threads run method is invoke thread goes from runnable to running state where thread actually executes its task.

## Waiting state -

2  
Sometimes a thread waits for another thread to perform a task first. In this state previous thread invokes sleep or wait method.

## Dead state -

A thread enters this state when it completes its task or gets terminated due to

`java.lang.Throwable` → (errors and exceptions)

abnormal condition arises runtime.

An event that disturbs normal flow of program

Exception handling → mechanism to handle runtime error

1. Import `java.lang.Throwable` class

2. Exceptions classified

1. ~~No~~ `IOException`

2. ~~SQL~~ `SQLException`

3. `ClassNotFoundException`

4. `RuntimeError`

## Trowable class

### Exception

To SQL ClassNotFound Runtime

Arithmatic NullPointerException NumberFormatIndex  
Exception Exception Exception Exception Bound

ArrayIndexOutOfBoundsException

- Java exceptions are divided into two types -

- Checked exception
- unchecked exception
- The class which directly inherit throwable class except runtime exception are called as IO, SQL exception

Checked exception are checked at compilation

- The classes which inherit runtime exception are known as uncheck exceptions

e.g. Arithmetic exception  
NullPointerException  
ArrayIndexOutOfBoundsException etc.

Uncheck exceptions are checked at runtime

IPC (Inter Thread Communication)

Allowing synchronized threads to communicate with each other

ITC is a mechanism in which running thread is pause in a critical section and another thread is allowed to enter or lock in some critical section to be executed.

Exception

- ITC is implemented by following methods of object class.

1. wait method      public final void wait()
2. notify method
3. notifyAll

1. It causes current thread to release the lock and wait until other thread either invokes the notify method or notify all method for specified amount of time.

2. wakes up a single thread that is waiting  
syntax - public final void notify()

3. wakes up all threads that are waiting  
syntax - public final void notifyAll()

difference bet' wait & sleep method of thread.

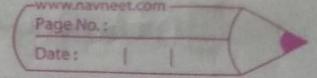
- Thread priority →

Each thread is assigned a default priority of thread that is `new NORM_PRIORITY`

Thread priorities can be reset using `setPriority` method

Some constants of priorities include

1. `Thread.MIN_PRIORITY`
2. `Thread.MAX_PRIORITY`
3. `Thread.NORM_PRIORITY`



#### 4) Throw :

1. It is used to throw an exception explicitly.
2. Throw keyword is used within the method.
3. Check exception can't be handled using throw keyword only.
4. Throw is followed by an instance.
5. Whenever throw keyword is used multiple exception can't be handle at a time.

void add()

{

    throw new ArithmeticException("divide by zero");

}

#### 5) Throws :

1. used to declare exception
2. Checked exception can be handled with throws keyword.
3. throws is followed by a class
4. throws is used with method signature
5. Using throws keyword multiple exceptions can be handled separated by comma.

void add() throws ArithmeticException

{

    // code

}

# Introduction to Applet with graphic programming

- Applet = Small java program used in internet computing.

## Two types of applet

1. Local
2. Remote

(No need of internet)

1. Developed & stored on Local system
  2. Developed by someone else stored on any server ~~site base attribute not having~~
- Difference bet<sup>n</sup> applet & application.

### Applet

### Application

1. No need of main method to write
2. Applets can not run independently like application
3. Applet cannot read from or write into the file place in local comp.

main method requires

Application is independent

can read from & write into file place in local comp.

4. It can't communicate with other server on network

can communicate with other server on network

5. can't run my program from local computer

can run program from local computer

6. Applets are restricted for using libraries from diff. languages

No restriction for using libraries

- Steps to create an applet -

1. Create a .java file
2. Compile above java file to get .class file
3. Design webpage using HTML tags
4. Encapsulate applet tag into webpage
5. Create HTML file
6. Execute an applet (run) We can execute applet by 2 ways.

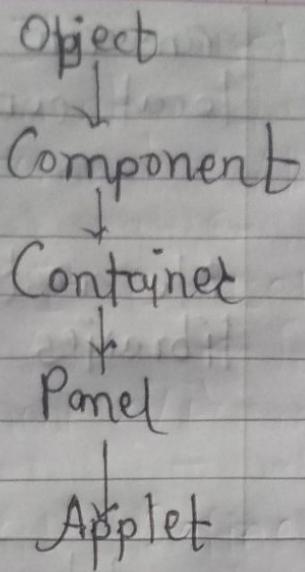
a. By HTML file

b. By applet viewer tool

- While writing applet

1. Import java.awt package
2. Import java.applet package
3. Extend applet class [public keyword should use for class which is extending applet]
4. Write paint method taking graphics as argument. Invoke drawing method.

applet class is a subclass of panel class

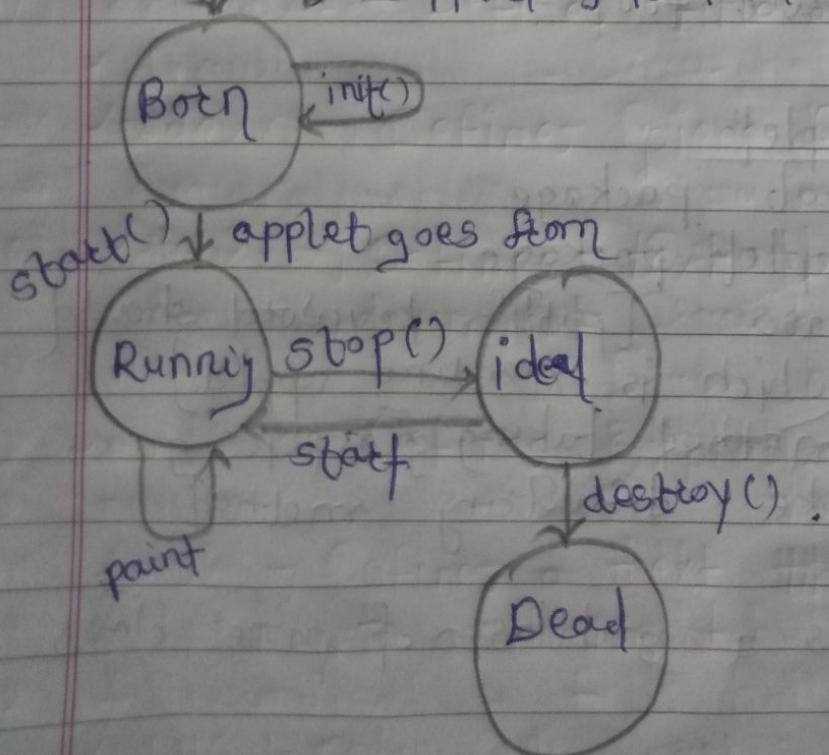


- Applets Lifecycle -

Applet state includes -

- 1) Born state [Initialization state]
- 2) Running state
- 3) Ideal state
- 4) Dead state [Destroy state]

↳ Begin (applet gets loaded)



## • Important tags in HTML

1) HTML tag - Root of html document which is used to specify that document is HTML.

`<html> statement...</html>`

2) Head tag - Head tag is used to contain all head element in HTML file

`<head> statements...</head>`

3) Body tag - Used to define the body of html document

`<body> Content of html page</body>`

4) title tag - It is used to define title of html document.

`<title> statements...</title>`

5) Heading tag - Used to define heading of html document

`<h1> Statement.</h1>`

`<h2> Statement...</h2>`

`<h3> Statement...</h3>`

`<h4> Statement...</h4>`

`<h5> Statement...</h5>`

`<h6> Statement...</h6>`

6) Paragraph tag - It is used to define paragraph content in html document.

`<p> statements...</p>`

7) Bold tag - It is used to specify bold content in html document.  
< b > statement < /b >

8) Italic tag - It is used to write content in italic form  
< i > statement < /i >

9) <!-- --> Defines a comment

10) <!DOCTYPE> Defines the document type

11) <abbr> - Defines an abbreviation or acronym

12) <big> - Defines big text

13) <address> - Defines contact information for author of document

14) <cite> - Defines title of work

15) <code> - Defines piece of computer code

16) <del> - Defines text that has been deleted from document

17) <em> - Defines emphasized text.

18) <ins> - Defines a text that has been inserted into a document

19) <kbd> - Defines keyboard input.

- 20) <mark> - Defines marked text.
- 21) <pre> - Defines preformatted text.
- 22) <q> - Defines short quotation.
- 23) <s> - Defines text that is no longer correct.
- 24) <small> - Defines smaller text.
- 25) <var> - Defines a variable.
- 26) <u> - Defines some text that is underlined and styled differently from normal text.
- 27) <wbr> - Defines possible line-break.
- 28) <strong> - Defines important text.
- 29) <sub> - Defines subscripted text.
- 30) <sup> - Defines superscripted text.
- 31) <strike> - Defines strikethrough text.
- 32) <time> - Defines a specific time.

- passing parameters to applet
- <param> - param tag has 2 attributes
  - 1) name eg. color
  - 2) value eg. red
- getparameter() - for taking parameter argument passed = name (color)  
return value = value of param tag (red)

- Displaying numeric value

write a valueof method of string class  
datatype + accept string value

- Graphics programming

awt package

- Methods under graphics class

1. drawArc()
2. drawLine() - Draw hollow line
3. drawOval() - Draw hollow oval
4. drawPolygon() - Draw hollow polygon
5. drawRect() - Draws hollow rectangle
6. drawRoundRect() - Draw hollow rectangle (round corners)
7. drawString() - Draw string in output
8. fillArc() - fills the hollow arc
9. fillOval() - fills the hollow oval
10. fillPolygon() - fills the hollow polygon
11. fillRect() - fills hollow rectangle
12. fillRoundRect() - fills hollow rectangle
13. getcolor() - Returns current drawing color

14. `getFont()` - retrieves font  
15. `setColor()` - set drawing color  
16. `setFont()` - set font of text

• Lines and rectangles -

1. `drawLine(x1, y1, x2, y2)`  
`drawLine(20, 20, 30, 30)`

2. `drawRect(x, y, width, height)`  
x, y of top left corner

eg.

`drawRect(70, 100, 30, 30)`

3. `drawRoundRect(x, y, width, height, x1, y1)`

eg. 6 parameters

`drawRect(70, 100, 30, 30, 50, 50)`

4. `fillRect(x, y, width, height)`

`FillRoundRect(x, y, width, height, x1, y1)`

5. e.g. `g.fillRect(70, 100, 30, 30)`

g. `FillRoundRect(70, 100, 30, 30, 50, 50)`

6. `drawOval(x, y, width, height)`

g. `drawOval(70, 200, 30, 30)`

7. `setColor(color, red)`

- `drawPolygon()`
  - Takes 3 argument
  - 1) Array of integer containing x coordinate
  - 2) Array of integer containing y coordinate
  - 3) integer for total no of points.

- `fillPolygon()`

Use to fill the polygon.

- Difference between AWT and swing.

Sl. No.	AWT	SWING
1.	AWT components are platform dependent.	Java swing components are platform independent.
2.	AWT components are heavyweight	swing components are lightweight.
3.	AWT doesn't support pluggable look and feel.	swing support pluggable look and feel.
4.	AWT provides less components than swing	Swing provides more powerful components such as tables, lists, scrollpanes etc.
5.	AWT doesn't follows mvc (model view controller)	Swing follows mvc.

- awt package -

Active buttons - radio , checkbox

awt package

↑  
Abstract Window Toolkit

↑  
GUI component present eg Textfield  
↓

Active

Action performed  
(event occurred)

passive

No action performed  
(events) Label , Textfield  
TextArea

Container class

Non-container

Panel      Window Frame

Label	Textfield	Button	Checkbox	Choice list
-------	-----------	--------	----------	-------------

TextArea

checkbox -  
Group

(Radio  
Buttons)

Scrollbar

Panel - Doesn't contain any title bar and menu bar

Window - Window is container which has no border and menu bar

Frame - frame can have title bar, menu bar and other GUI components.

Event -

An event is an object that describes the state change in source → Ex. Submit button

(Action performed)

(Object that generates event)

1) Needing listener -

A listener is an object that is notified when an event occurs.

- Button → Action Listener

The event occurs is 'action event'

\* actionPerformed();

- Method to write for action event / Listener

- Takes object of action event class

actionPerformed(ActionEvent e)

## \* swing in java -

Java swing class is a part of java foundation classes (JFC) that is used to create window based application.

It is built on top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, Jcheckbox, JMenu, JComboBox, etc.

## \* Methods of component class used in java swing

Method	Description
1 public void add(Component)	add component on another component.
2 public void setSize(int width, int height)	sets size of component.
3 public void setLayout(LayoutManager m)	sets the layout manager for the component.
4 public void setVisible(boolean b)	sets the visibility of component. It is by default false.

## • Managing Input/Output Files in Java

Persistent data stored in hard disk  
It can be retrieved.

Drawbacks -

- 1) Wastage of memory
- 2) Scope of variable
- 3) Non-permanent storage

Character - Sequence of unicode → (byte code)

Field - group of characters

Record - group of several fields

Files - Combination of record.

File is collection of related records stored on hard disk.

File processing -

Inserting and managing data using file.

Creating

Reading

Updating

Two keywords needed

input - flow of data into program

output - flow of data out from the program

Stream -

It represents the ordered sequence of data

travel from source to destination

Stream has types -

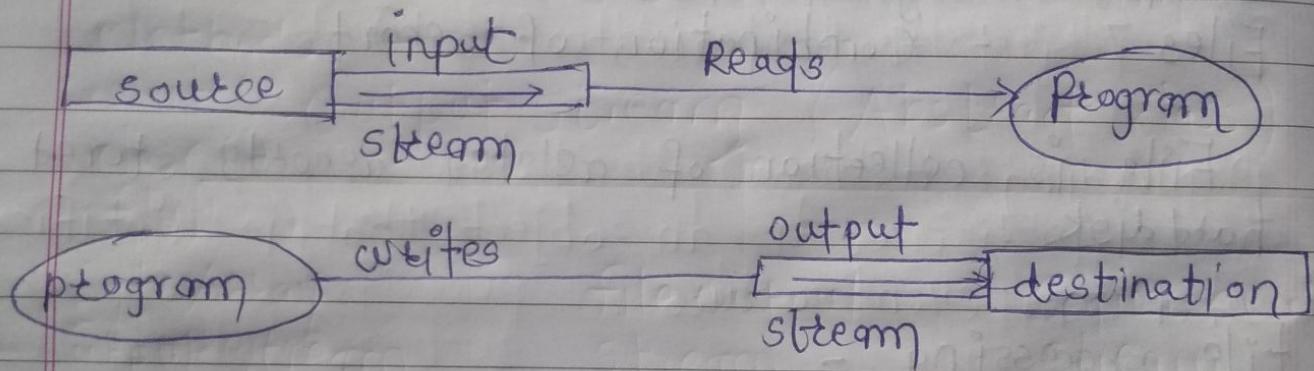
- 1) Input
- 2) Output

1) Input stream -

input stream extract data from the source and sends it to program.

2) Output stream -

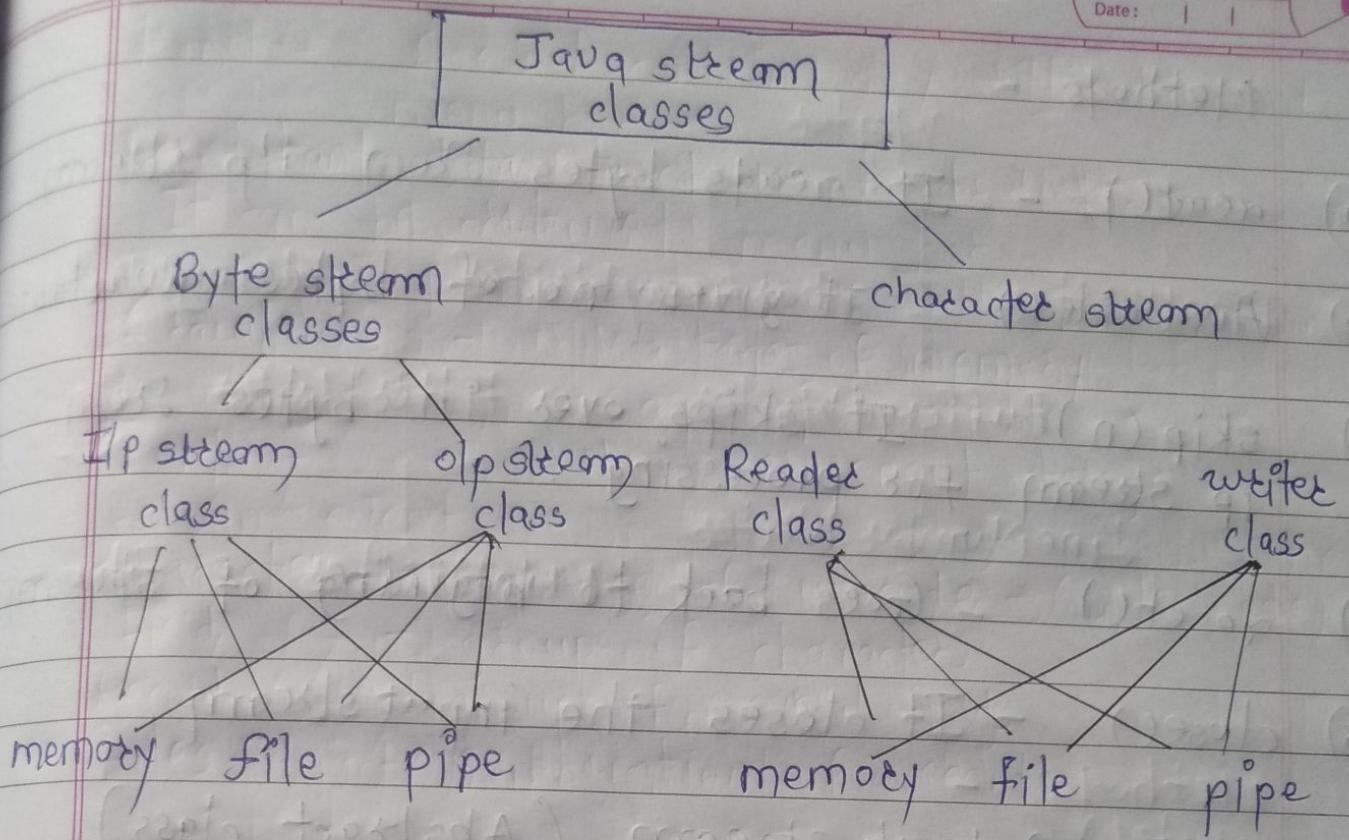
Output stream takes data from program and sends it to destination.



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

Need to import for all I/O operation  
Package contains few classes divided 2 geps

Java stream classes



## Input Stream (Abstract class)

Byte Array  
i/p stream

File Input  
stream

Sequence Input  
stream

Object Input  
stream

Filter Input  
stream

Piped Input  
stream

Buffered Input  
stream

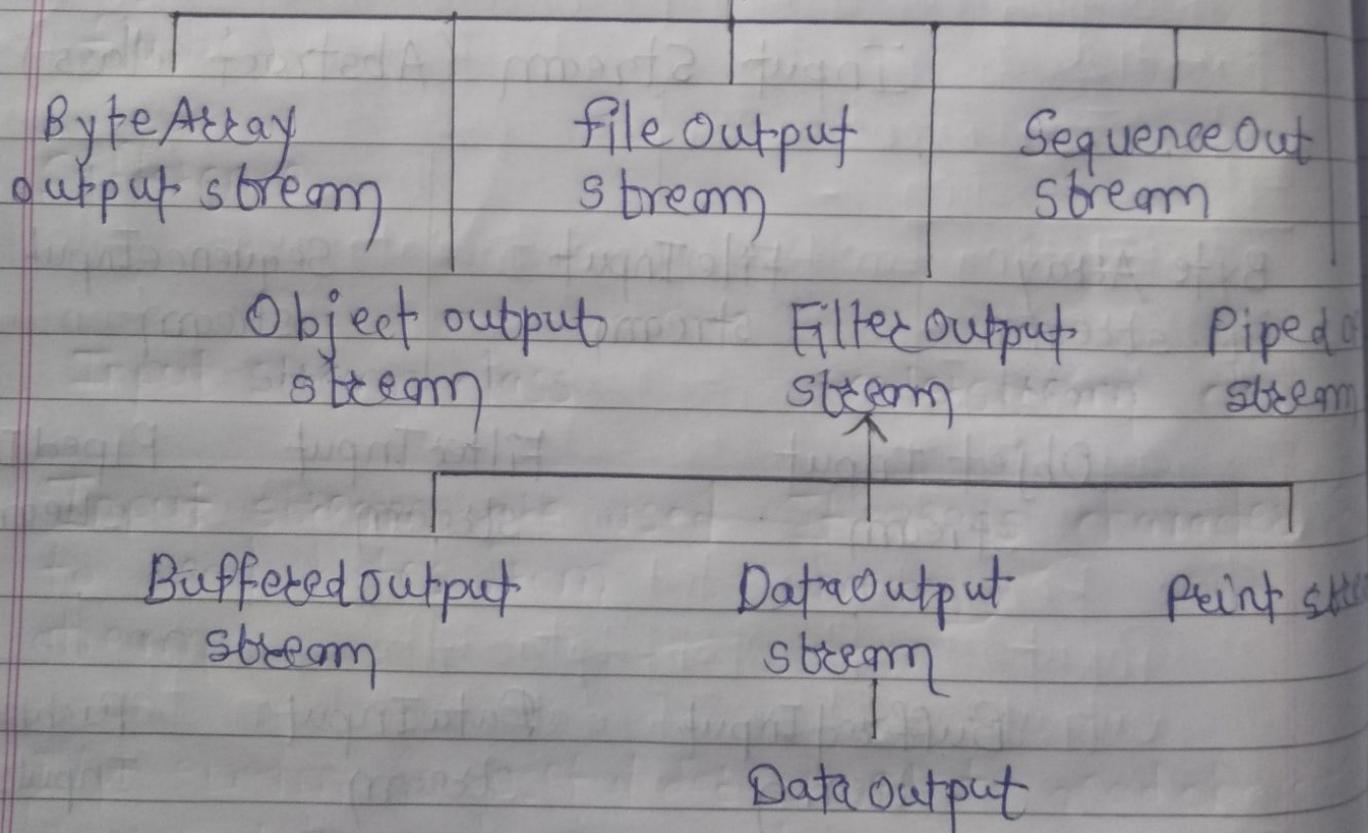
Data Input  
stream

Pushback  
InputStream

## Methods -

- 1) read() - It reads bytes from input stream
- 2) Available() - It gives no. of bytes available.
- 3) skip(n) - It skips over 'n' bytes from input stream for processing.
- 4) reset() - Goes back to beginning of stream.
- 5) close() - It closes the input stream.

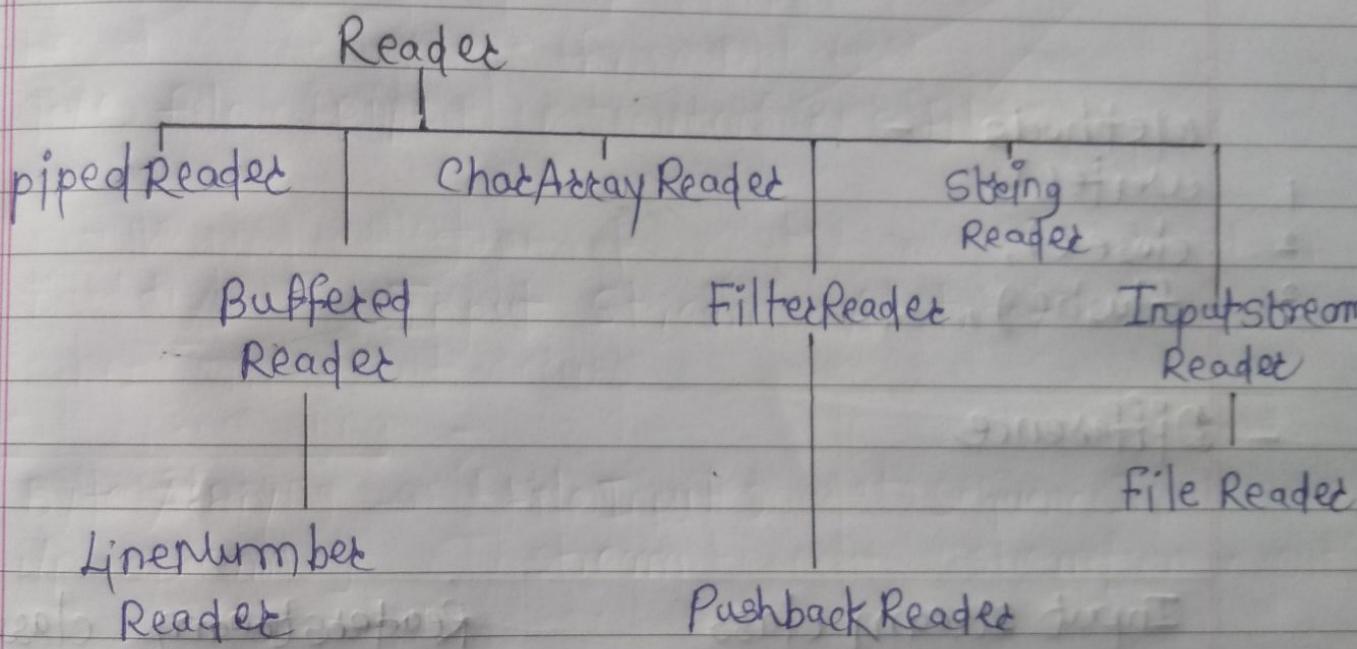
## Output Stream (Abstract class)



## methods -

- 1) write() - It writes data to stream.
- 2) close() - It closes the output stream.
- 3) flush() - It flushes the output stream.

## Character stream classes -



All methods can throw IOException  
 So it is compulsory to write try block  
 for code and to handle IOException.

- Methods -
- 1) read();
  - 2) reset();
  - 3) close();
  - 4) skip( $\pi$ );

## Writer

Piped Writer

char Array Writer

String Writer

Buffered Writer

Filter Writer

Line Number Writer

Print Writer

Methods -

1. write();
2. close();
3. flush();

- Difference

Input Stream class

Reader Stream class

1. Input stream are used to read bytes from stream. It reads characters from stream.
2. I/p stream is useful for binary data such as images, videos and objects. Used to read character data.
3. I/p stream is byte oriented. It is character oriented.

## Output Stream

## Writer class

1. Output stream are used to write bytes to screen

It is used to write character to stream.

2. It is byte oriented

It is character oriented.

3. Used to write eight bit unicode

Used to write 16 bit unicode