JAVA HANDWRITTEN NOTES

HAPPY LEARNING!

Java Evolution and basics of java

1. Java evaluation 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 makin 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 beam of researches to create language that would allow consume electronic devices to communicate with each othe

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 plateforms (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.

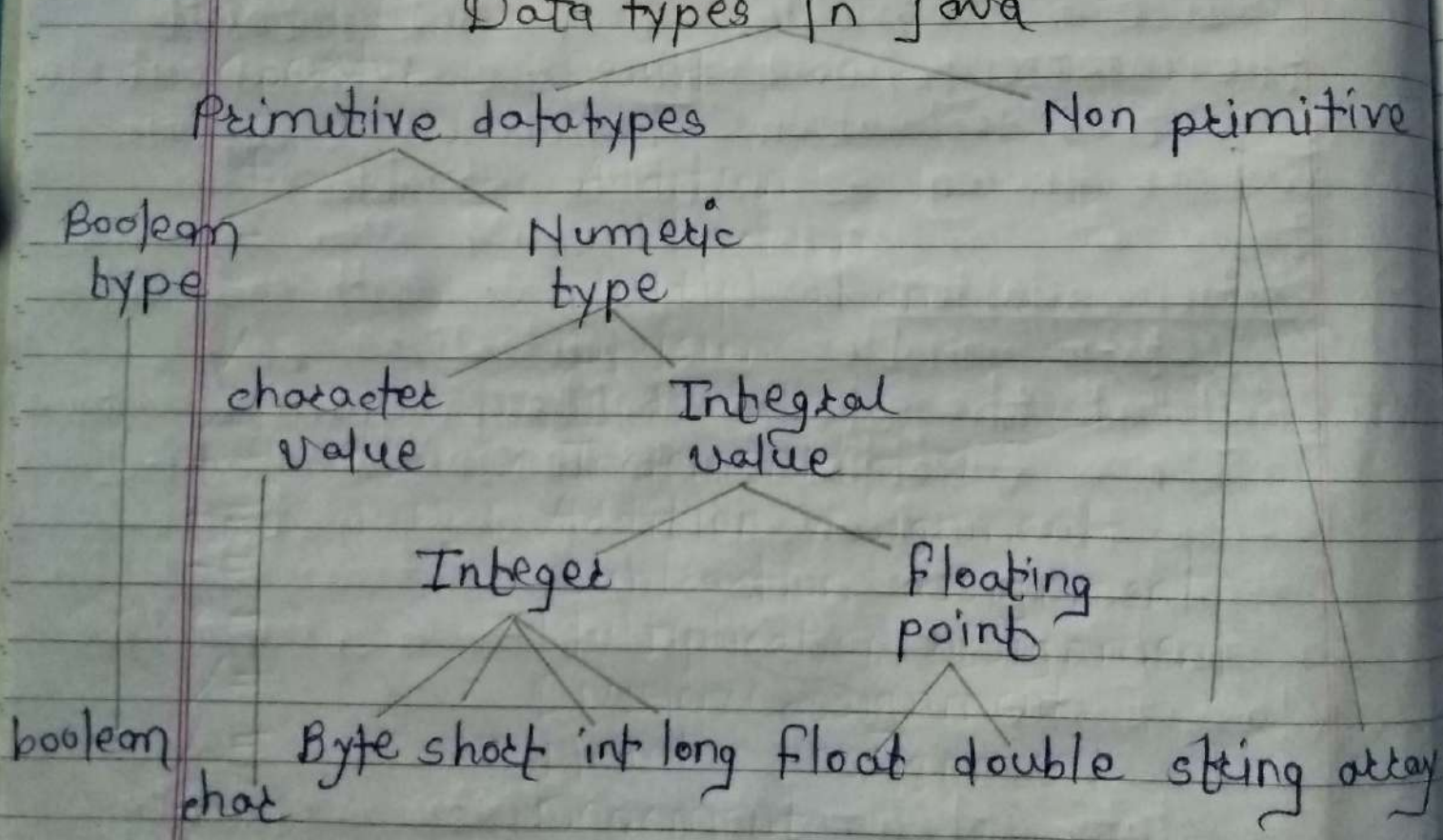
3. Static variables -
   Static variables are initialized only once at the start of the program execution. These variables should be initialized first, before the initialization of any instance variables.

```
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



- **Constant** -

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

Java doesn't have builtin 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 anytime 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 input of an operator.

Types of operators -

1. **Arithmetic operators**

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

- Operands of ari. operators must numeric type. Char 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
~	Bitwise unary not
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	shift right
>>>	shift right zero fill
<<	shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>> =	shift right assignment
>>> =	shift right zero fill assignment
<< =	shift left assignment

3. Relational Operator -

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

Operator	Result
==	Equal to
!=	not equal to
>	Greater than
<	Less than
>=	Greater than or equal to

<=	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 -

expression 1 ? expression 2 : expression 3

If expression 1 evalutes 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.

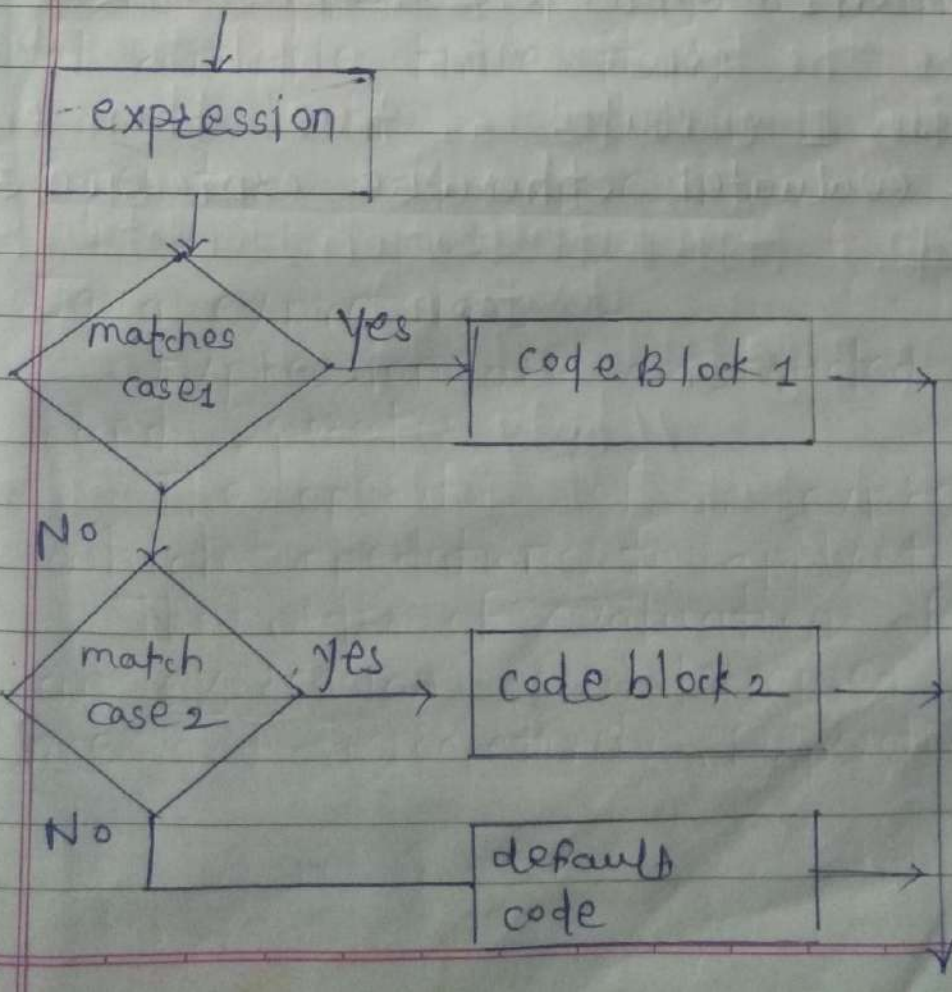
General form: if (condition)
statement;
else if (condition)
statement;
.
.
.
else
statement;

- **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 homogenous 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.

eg. 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];

eg. 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 -

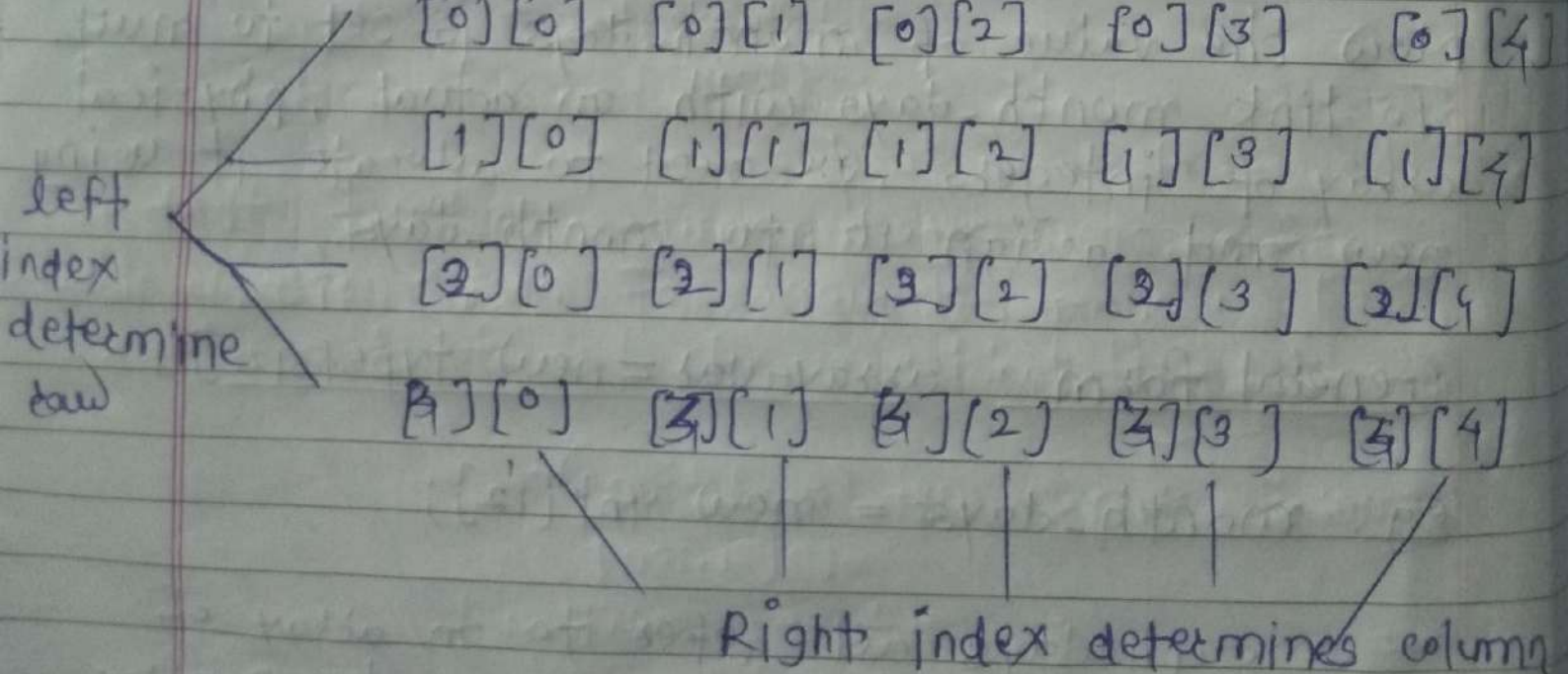
month_days[1] = 28;

The statement assigns the value 28 to 2nd 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.



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 defination -

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 defination, 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 without object
void: Main method doesn't return anything
main(): Name configuared in JVM.
string[]: The main method accepts a single argument: an array of element of type string.

main method in 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 serching 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 encapsu-

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 direct of college is present in CLASSPATH.

- Subpackage -
  packages that are inside another pack are the subpackages.
  These are not imported by default, they have to imported explicitly.

eg. 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("GeeksforGeeks");
    }
}
```

- `import java.util.Vector;`
  package class subpackage class
- `import java.util.*;`
  package subpackge all the classes in util subpackage
- `import static java.util.System.*;`
  static package subpackg class all the fun" in System class
  static import

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, finalizer 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, Wrapper 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 reptesenting 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 seperated 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^n 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-

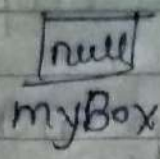
```
class_name obj-name = new constructor of class();
```

eg. `Box myBox = new Box();`

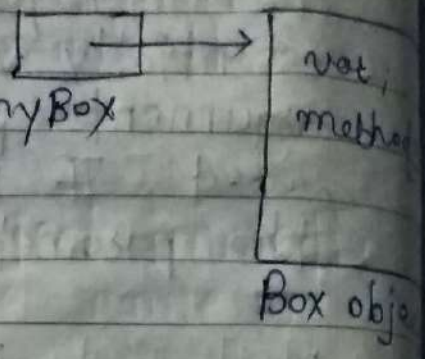
Here now myBox is an instance of Box.
new is a keyword who allocates memory for object and returns a reference to it.

address in memory of object allocated by new.

```
Box myBox;
```




```
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.heigh = 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 immediatly 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 jarq 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();
}
...
}
```

→ javac →

```
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.

Eg:

```
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) } Invalid case of
float add (int, int) } 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 Data type 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', 89);
    obj.disp(89, '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 Greater 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 declare as private in super class i can't be inherited.
- A superclass variable can access a subcla object.
- When a reference to a subclass object is assigned to superclass reference variable, we will have access to those part of objec defined by superclass.
- Using super:

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

super has two general forms —

1. calls superclass constructor super(arg 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)
     {
```

```
        a=x; b=y;
    } void sum(){result = a+b;
                System.out.println("sum = "
}                   +result); }
class B extends A
{
    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 subOb = new B(10,20,30);
        subOb.sum();
        System.out.println("Value of c : "+subOb.c
    }
}
```

2.
```
class A
{
    int i;
}
class B extends A
{
    int j;          //hides the i in A class
    B(int a, int b)
    {
        (this.i = a) super.i = a  //i in A
        j = b
    }
    void show()
    {
        System.out.println("i in superclass: " +super.i);
        System.out.println("i in subclass: " +i);
    }
}
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

```
// Demostrates 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 extending 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 super-class, 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 wit a subclass, it will always refer to the ver of that method defined by subclass. The version defined by superclass will be hidd

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 overridding is an example of runtime polymorphism.

- Polymorphism - Performing single action in different ways.

- Polymorphism is a capability of method to do different things based on the object that it is 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

```
// 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
  }
}
```

```
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 r; // obtain a reference of type A
    r = a; // r refers to a object
    r.callme(); // calls A's version of callme()

    r = b;
    r.callme();

    r = c;
    r.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.

```
r = a; // r refers to an A object
r.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 n-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 method 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 elem
6.	insertElementAt()	It is used to insert the specified object as a comp 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 vect
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 encapsulate 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 (reference 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	Double
boolean	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 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 Wrapper class [Boxing and unboxing].

```
import java.util.*;
public class WrapperDemo
{
    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 isinstance of Byte)
{
    System.out.println("An object of Byte is ");
}

Integer obj2 = Integer.valueOf(b);
if(obj2 isinstance of Integer)
{
    System.out.println("An object of Integer is
                        created");
}

Float obj3 = Float.valueOf(c);
if(obj3 isinstance of Float)
{
    System.out.println("An object of Float is
                        created");
}

Double obj4 = Double.valueOf(d);
{
    System.out.println("An objects of Double is
                        created");
}

Character obj5 = Character.valueOf(e);
if(obj5 isinstance of 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 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 implicitely declare 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 var ;

- 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 var ;

      var = color . pink ;

      //Output an enum value
      System.out.println("Value of var : "+var

      var = color . green ;

      // Compare two enum values
      if (var == color.green)
         System.out.println("var contains green

// Use an enum to control switch state
ment .
```

```
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(), stringBuffer

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 a 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.

eg. 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.

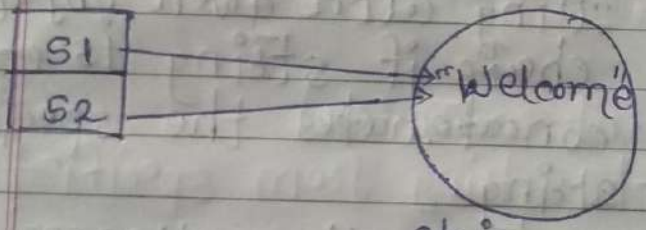
In java string is an object that represents a sequence of characters.

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

- Two ways of creating string object -

1. 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 constant pool

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

2. 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.lang.String class provides many useful methods to perform operations on sequence of char values.

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

- Constructors in StringBuffer class.

1. StringBuffer()
This constructs a single string buffer with no characters in it and an initial capasity of 16 characters.

2. StringBuffer (CharSequence seq)
This constructs a string buffer that contains the same characters as specified CharSequence.

3. StringBuffer(int capacity)
This constructs a string buffer with no characters in it and the specified initial capasity.

4. StringBuffer(String str)
This constructs a string buffer 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 store into the String type args parameter which is main method parameter.

- Syntax to provide commandline arguments -

java Programname arg1 arg2 ....

- 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 Yashshri

OUTPUT:

The command line arguments are:
Yashshri

Multithreaded Programming Managing ~~error~~ exception

- Thread -
small program to be executed

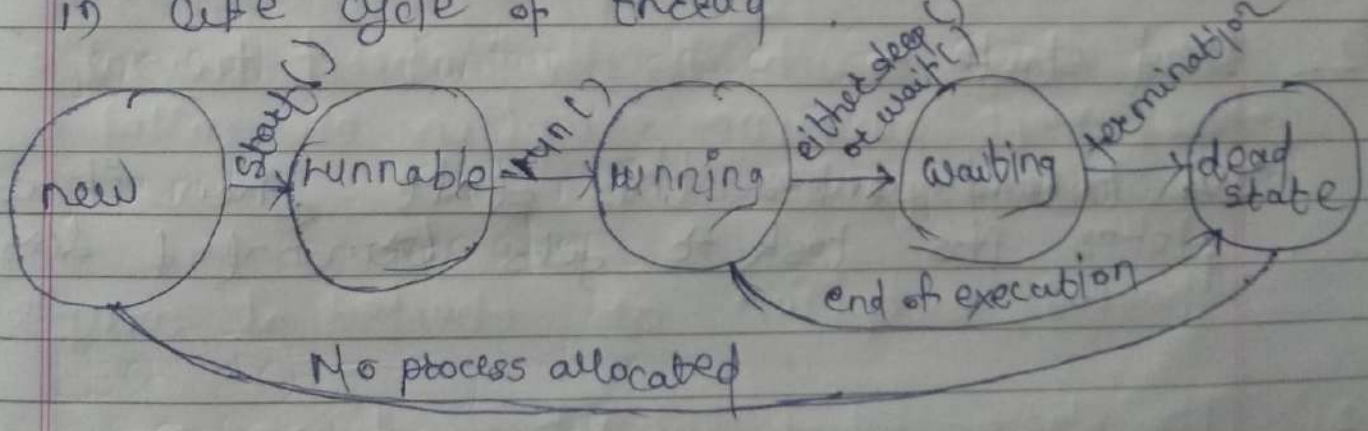
- Two ways for making thread -

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

- Life cycle of a thread -

1. A thread goes through various stages in its life cycle like a thread is born, started, runs and finally dies.

2. All the above stages forms various states in life cycle of thread



new - a new thread begins 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 executed the task whenever threads run method is invoke thread goes from runnable to running state where thread actually executes its task.

Waiting state -

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

Dead state -

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

java.lang.Throwable → (error and exceptions)

abnormal condition arises runtime.

An event that disturbs normal flow of program

Exception handling –
mechanism to handle run time error.

1. Import java.lang.Throwable class
2. Exceptions classified
   1. IOException
   2. SQLException
   3. ClassNotFoundException
   4. RuntimeException

Throwable class

Exception

IO SQL ClassNotfound Runtime

Arithmetic Exception, NullPointer Exception, NumberFormat Exception

ArrayIndexOutOfBound Exception

- Java exceptions are divided into two types -

1. Checked exception
2. unchecked exception.

1. The class which directly inherit throwable class except runtime exception are called as IO, SQL exception

Checked exception are checked at compile ti

2. The classes which inherit run time exception are known as uncheck exceptions

e.g. Arithmatic exception
NullPointer exception
ArrayIndexOutOf Bound etc.

Uncheck exceptions are check at runtime

IPC (Inter thread communication)
Allowing synchronize threds to communicate with each other

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

- 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^n wait & sleep method of thread.

- Thread priority →

Each thread is assign a default priority of thread that is NORM_PRIORITY

Thread priorities can be reset using setPrior method

Some consbants 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 explicitely
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 use 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 seperated 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 (No need of internet)
2. Remote

1. Developed & store on Local system
2. Developed by someone else stored on any server

- Difference bet^n applet & application.

Applet	Application
1. No need of main method to write	main method requires
2. Applets cannot run independly like application	Application is independent
3. applet cannot read from or write into the file place in local comp.	can read from or write into file place in local comp.
4. It can't comunicate with other server on network	can communicate with other server on network

5. can't run any 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.

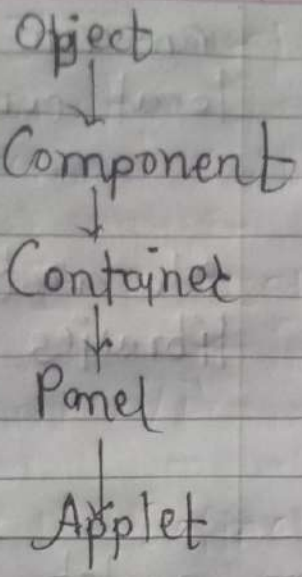
1. Create a .java file.
2. Compile above java file to get .class file
3. Design webpage using HTML tags
4. Incorporate applet tag into webpage
5. Create HTML file
6. Execute an applet (run) We can execute applet by 2 ways.

   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 g as argument. Invoke drawString method.

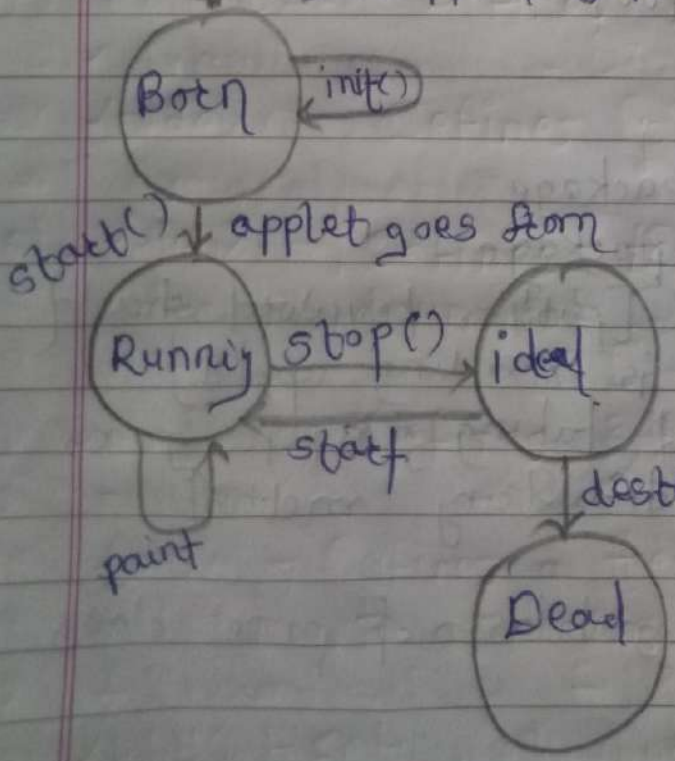
applet class is a subclass of panel class



- **Applets lifecycle-**

Applet state includes -

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



- 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.

 statements...

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

8) Italic tag - It is used to write content in italic form
 statement

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

10) <!DOCTYPE> Defines the document type

11) <abbr> - Defines an abbreviation or any 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) - 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 quatation

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 unarticulated and styled differently from normal text.

27) <wbr> - Defines possible line-break

28) - 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 parameter to applet

<parm> - parm tag has 2 attributes
1) name eg. color
2) value eg. red

getparameter() - for taking parameter
argument passed - name (color)
return value - value of parm tag (red)

- Displaying numeric value

write a valueof method of stringclass
drawString 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 polygoon
5. drawRect() - Draws hollow rectangle
6. drawRoundRect() - Draw hollow rectangle (round)
7. drawString() - Draw string in output
8. fillArc() - fills the hollow arc
9. fillOval() - fills the hollow oval
10. fillPolygon() - fills the hollow polygoon
11. fillRect() - fills hollow rectangle
12. fillRoundRect() - fills hollow rectangle
13. getColor() - Retrives current drawing color

14. getfont() - retrives font
15 setColor() - set drawing color
16. setfont() - set font of text

• Lines and rectangles -

1. drawLine (x_1, y_1, x_2, y_2)
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 parameter
drawRect (70, 100, 30, 30, 50, 50)

4. fillRect (x, y, width, height)
FillRoundRect (x, y, width, height, x1, y2)

5. eg. 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 co-ordinate
  3) integer for total no of points.

- fillPolygoom()
  Use to fill the polygoom.

- Difference between AWT and swing.

Sr.No.	AWT	SWING
1.	AWT components are platform dependent.	Java swing components are plateform independent.
2.	AWT components are heavyweight	Swing components are lightweight.
3.	AWT doesn't support pluggable look and feel.	swing support pluggble 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

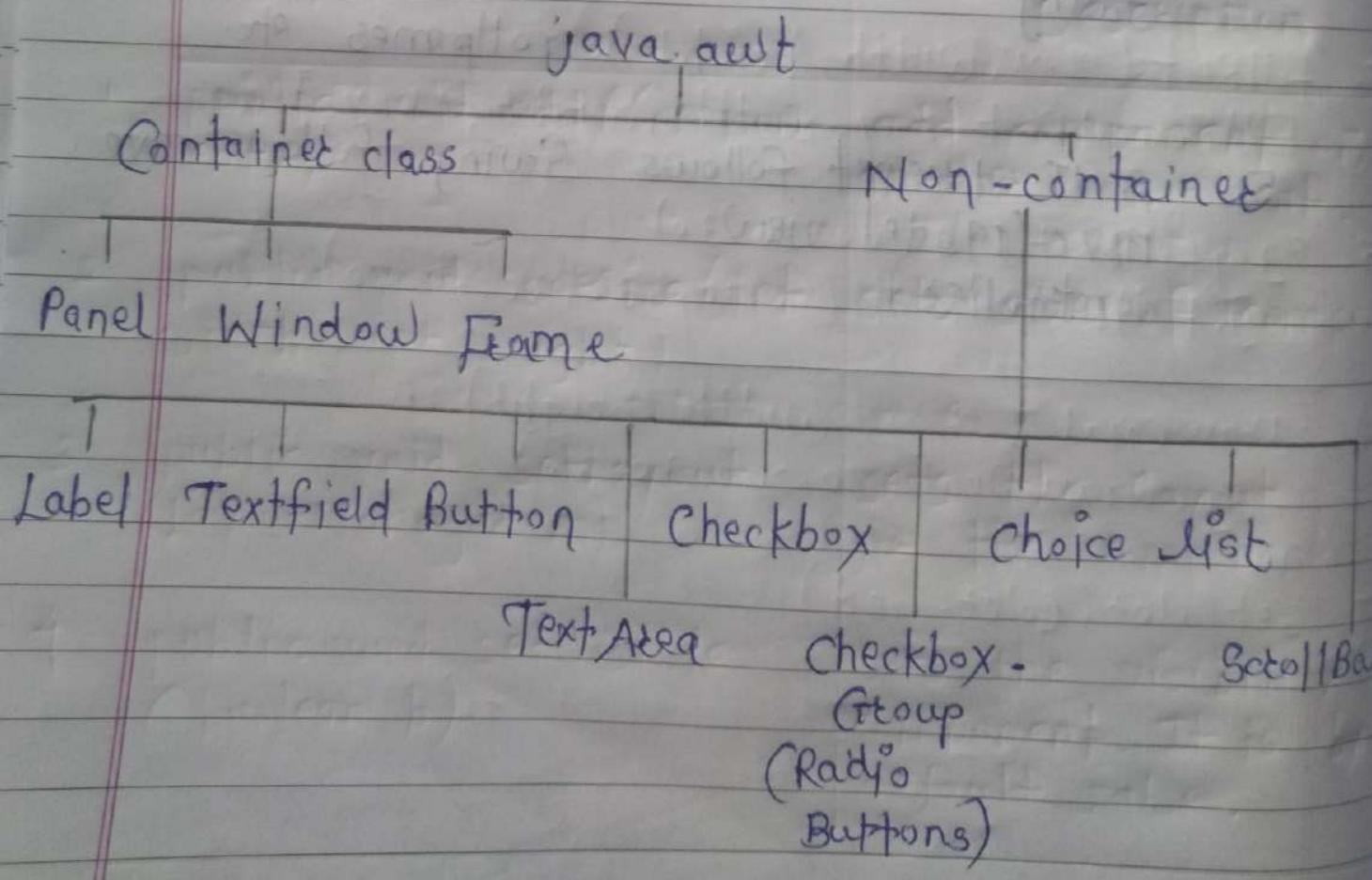
↓ ↓

Active — passive

Action performed — No action performed

(event occured) — (events) Label, Textfield

TextArea



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 describe the state change in source → Ex. Submit button

(Action performed)
(Object that generate event)

1) Needing listner -

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

- Button → Action listner
  The event occure is 'action event'
  * actionPerform();

- method to write for action event / listner
- Takes object of action event class

actionPerform(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, JColotChooser, etc.

* Methods of component class used in java swing

	Method	Description
1	public void add(Component c)	add component on mother component.
2	public void setsize(int width, int height)	sets size of componen
3	public void setLayout (layoutManaged m)	sets the layout manag for the component.
4	public void setVisible (boolean b)	sets the visibility of component. It is by default false.

6. Managing Input/Output files in Java

Persistant data stored in hard disk
It can be retrived.

Drawbacks :- 1) Wastage of memory
2) Scope of variable
3) Non-permenant 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 -
Storing and managing data using file.
creating
reading
updating

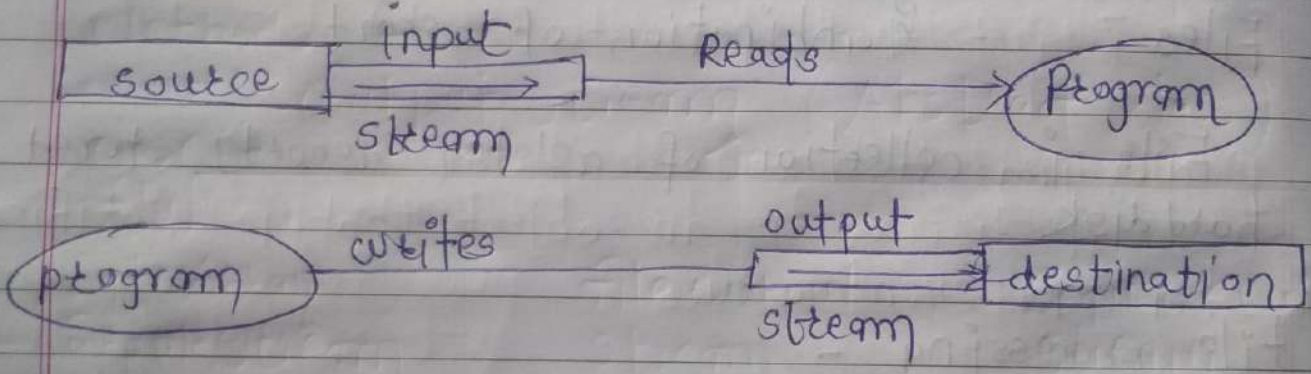
Two keywords needed
input - flow of data into program
output - flow of data out from the program

Stream -
It represents the ordered sequence of data
travel from source to destination

Stream bajs types -
1) Input
2) Output

1) Input stream -
input stream extract data from the source and sends it to program.

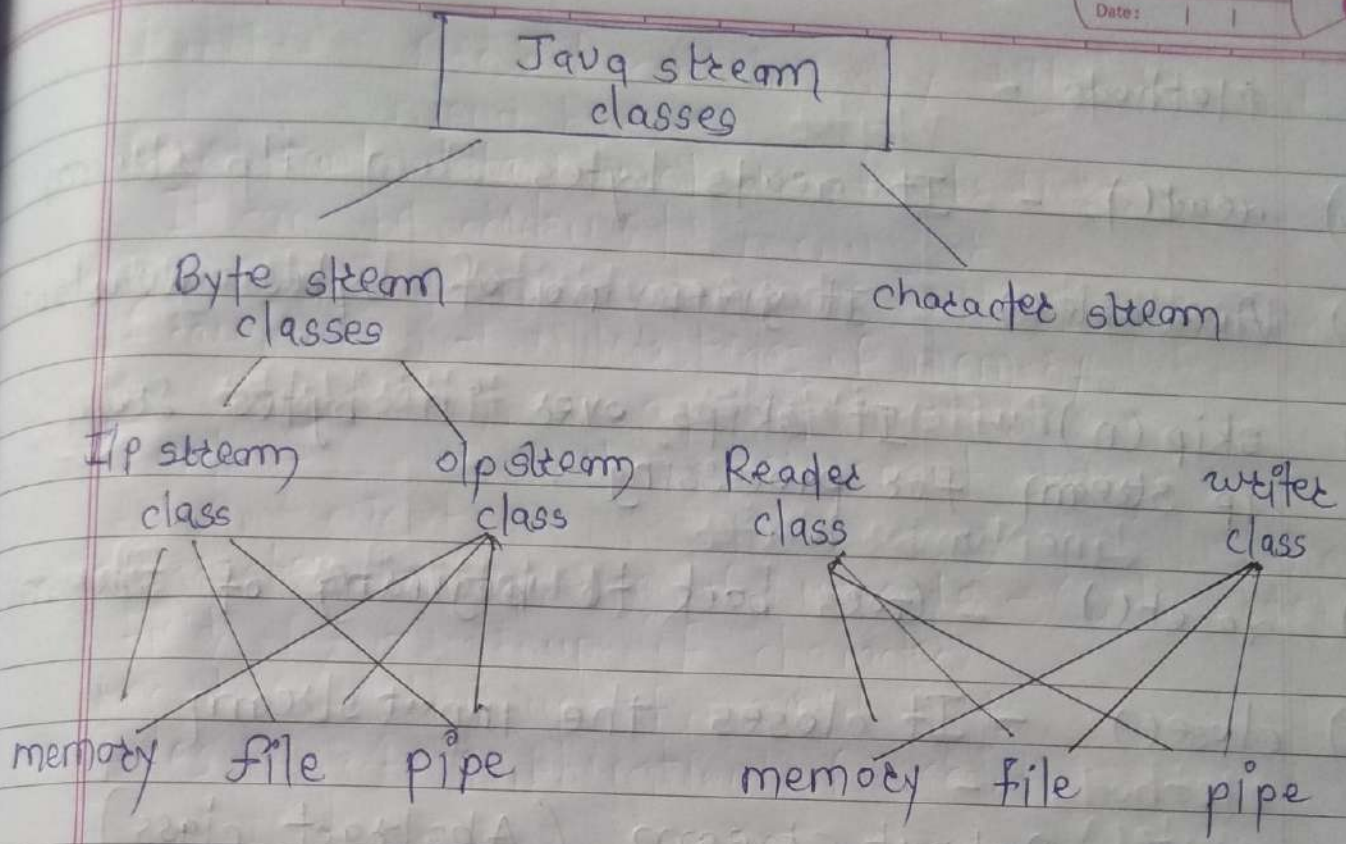
2) Output stream -
Output stream takes data from program & sends it to destination.

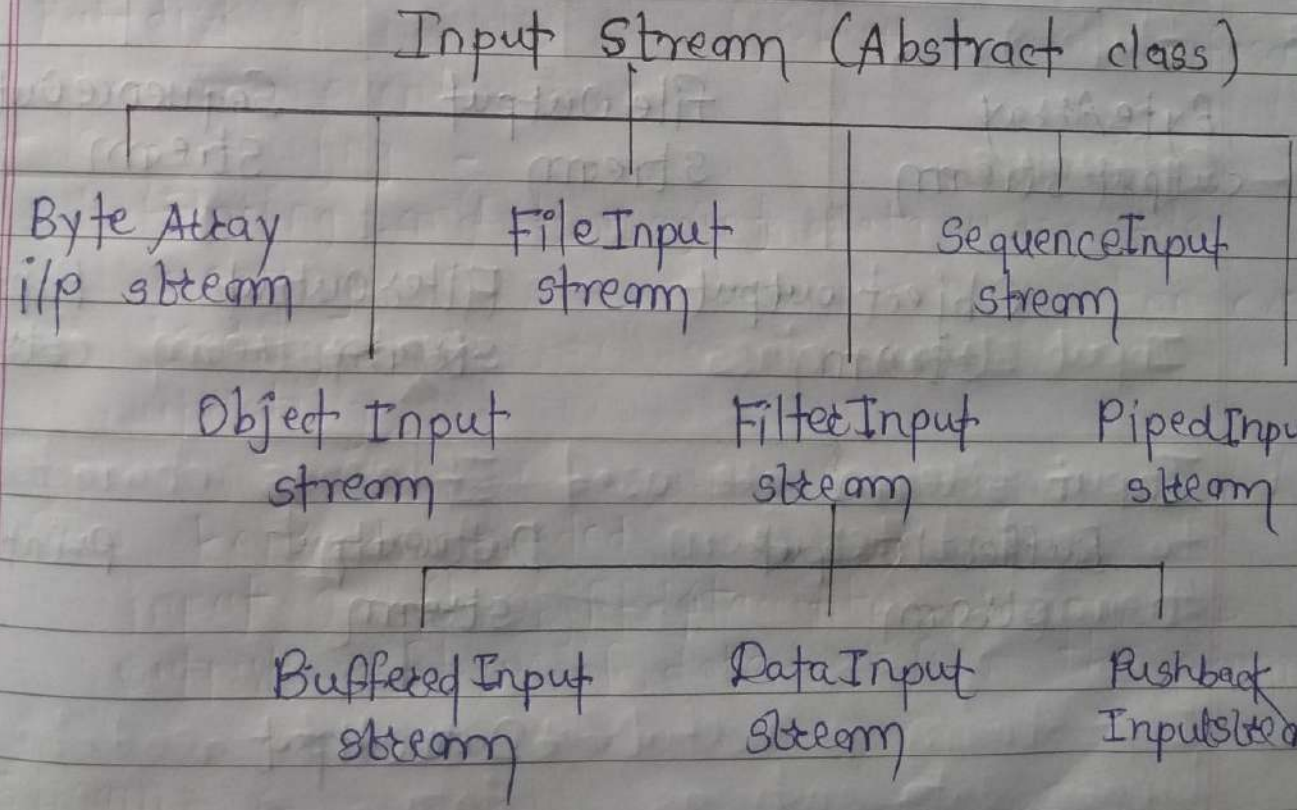


imported java.io.*;

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

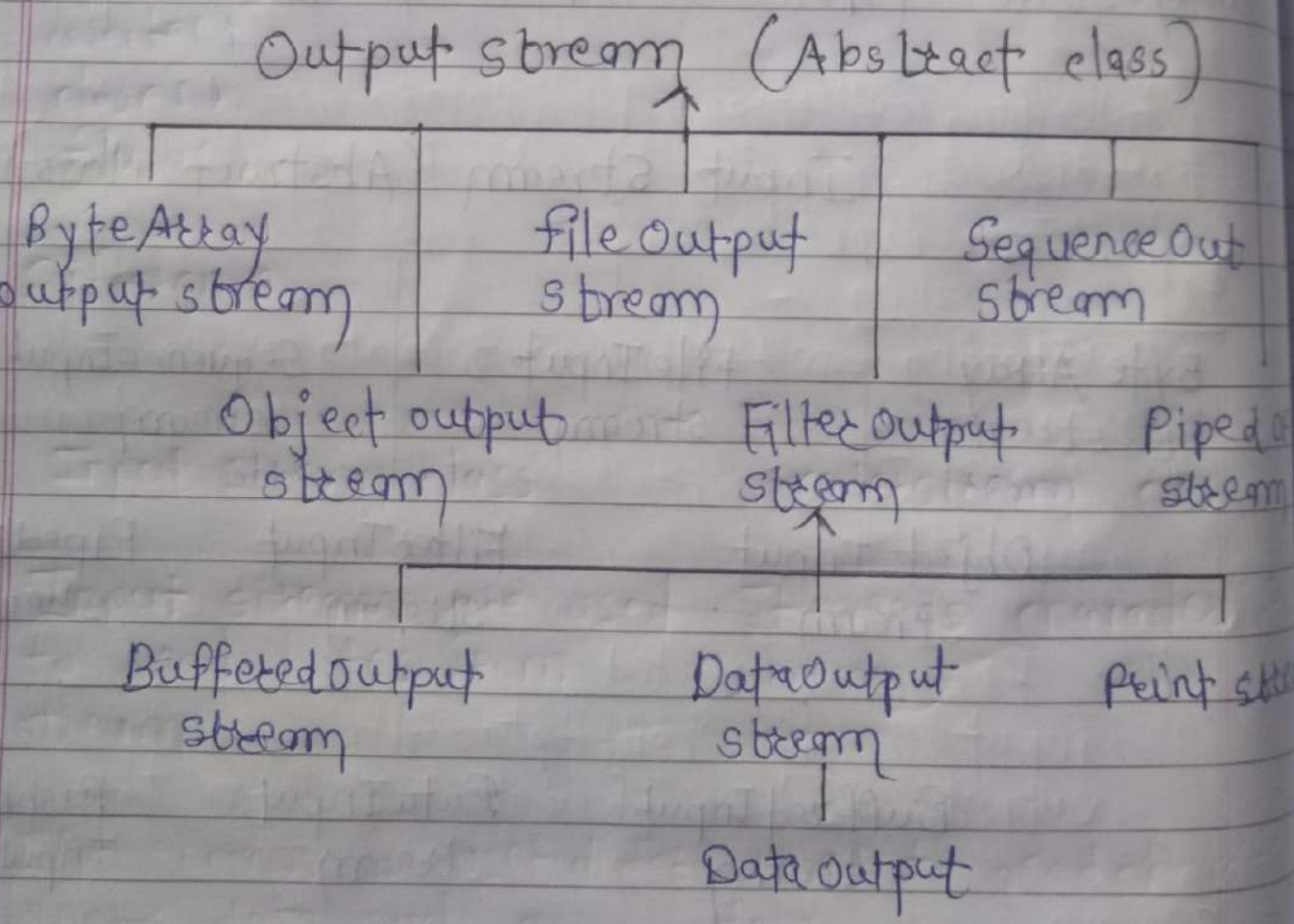
Java stream classes





Methods -

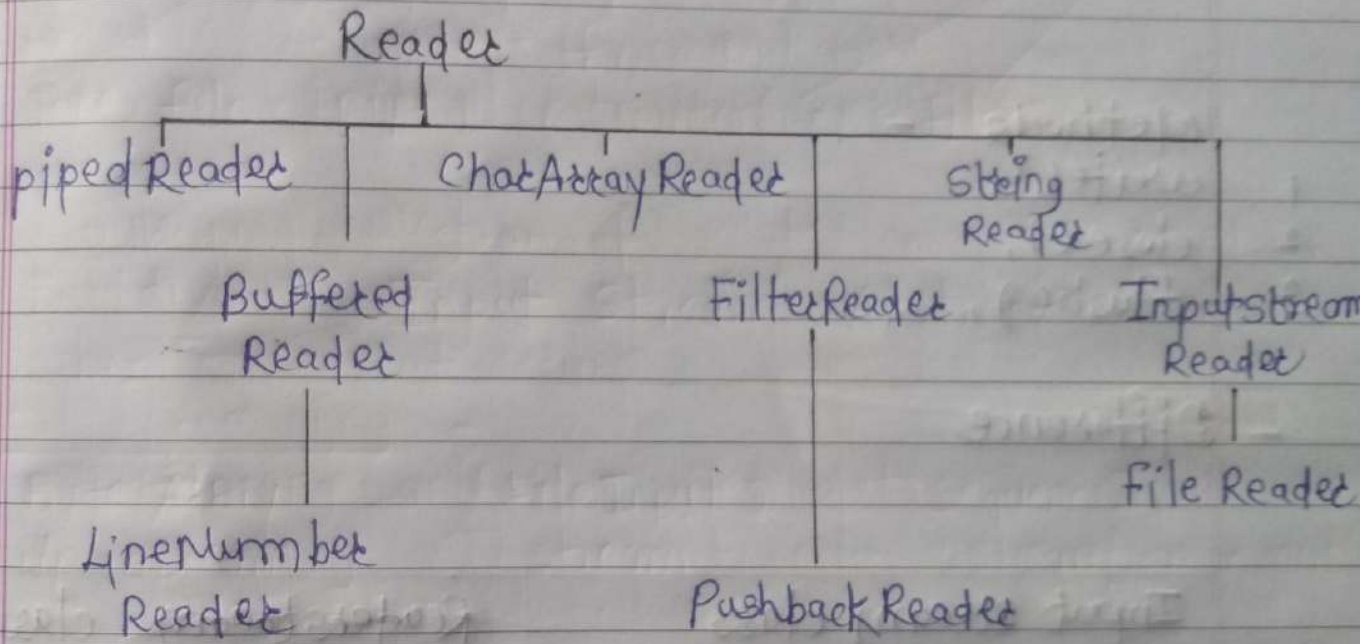
1) read() - It reads bytes from ilp stream
2) Available() - It gives no. of bytes avail
3) skip(n) - It skips over 'n' bytes from ip stream for processing.
4) reset() - Goes back to beginning of stream
5) close() - It closes the input stream.



methods -

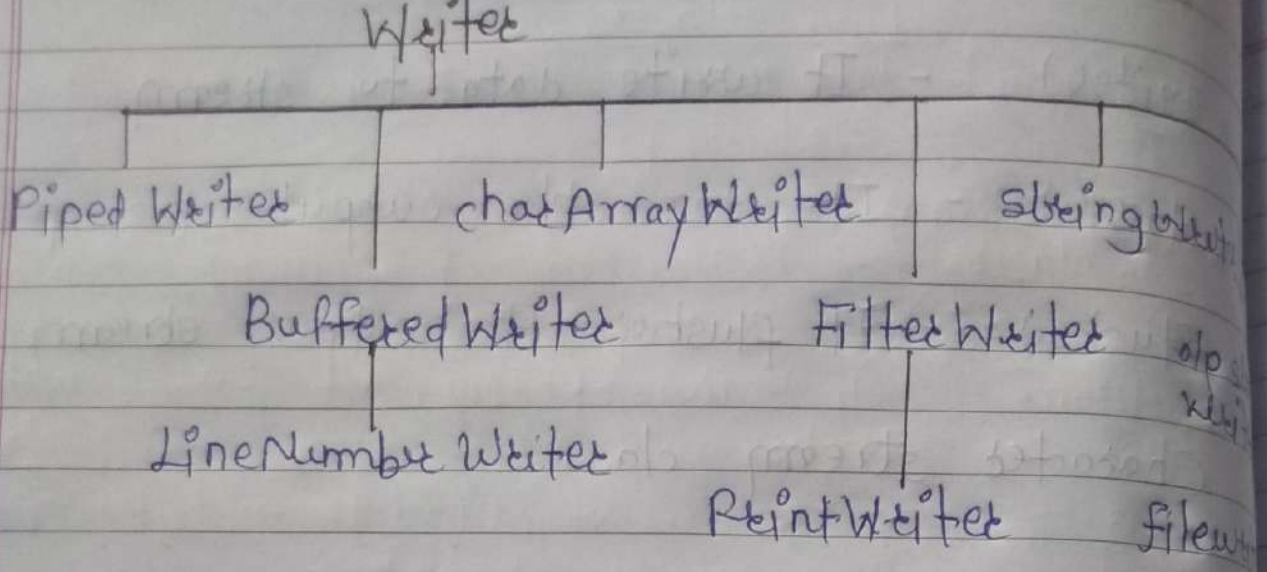
1) write() - It write data to stream.
2) close() - It closes the output stream
3) flush() - It flushes the output stream.

Character stream classes -



All methods can throw IO Exception so it is compulsory to write try block for code and to handle IO exception.

Methods - 1) read();
2) reset();
3) close();
4) skip(π);



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 character 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.