UNIT-2

Inheritance : Inheritence is nothing but creating a new class from already existing class.

Syntax

```
class subclass-name extends superclass-name.
{
  // methods and fields
}
```

Types of Inheritance

1. Single Inheritance :- It is a process of creating only one subclass from only one superclass

Syntax:

```
public class A{
  // code
}
public class B extends A{
  // code
}
```

2. Multilevel Inheritance :- It is a process of creating one subclass from already derived class.

Syntax :-

```
public class A {
  //code
}
public class B extends A {
  //code
}
public class C extends A {
  //code
}
```

3. Multiple inheritance :- It is a process of creating only one subclass from more than one superclass.

Syntax :- like Multilevel // In java it does not support Multiple inheritance

4. Hierarchical inheritance :- It is creating more than one subclass from one superclass

Syntax :- public class A { .. }
public class B extends A { .. }
public class C extends B { .. }

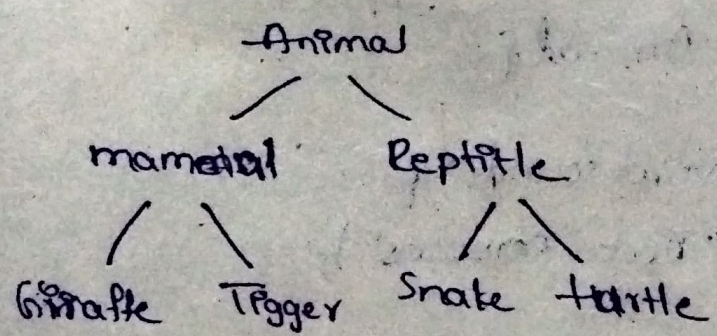
5. Hybrid inheritance :- Combination of any two inheritance.

Program

```
class Animal{
    void eat(){
        S.out.println("This animal eats food");
    }
}
class Dog extends Animal{
    void bark(){
        S.o.p("This dog barks");
    }
}
public class Main{
    P S V m(String[] args){
        Dog dog = new Dog();
        dog.eat();
        dog.bark();
    }
}
```

Output: This animal eat foods
This dog barks

Example of Inheritance



Polymorphism

Polymorphism is a combination of poly + morphs which means many forms.

Ex a person who at the same time with diff chas.

Types of polymorphism

-> Static or Early polymorphism when the type of the object is determined by the compiler is.

-> Dynamic or late polymorphism run-time is

Program

```
class Animal{
public void sound() {
        S.O.p("Animals makes a sound");
} }
class Dog extends Animal{
        void soun
        @Override
        public void sound(){
            S.O.p ("Dog barks");
    } }
class cat extends Animal{
        @override
        public void sound(){
            S.O.p("Cat meows");
        }
}
public class MaPn{
    public static void main(String[] args){
        Animal myDog = new Dog();
        Animal myCat = new Cat();
```

```
        myDog.Sound();
        myCat.Sound();
    }
}
```

output :- Animal makes a sound.
Dog barks
Cat meows.

Constructor overloading

It is a technique of defining more than one constructor with same name but with diff signatures.

Signatures
- Number of arguments
- Type of arguments
- Sequence of arguments

~~Do this even different~~ (this any) ~~on argument list of a constructor overloading~~

Program

```
class Box{
    double width, height, depth;
    Box(double w, double h, double d){
        width=w;
        height=h;
        depth=d;
    }
    Box(double len){
        width=height=depth=0;
    }
    Box(double len){
        width=height=depth=len;
    }
}
```

```
Public class Test{
    P S V m(String[] args) {
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mybox3 = new Box();
        double vol;{ return width* height * depth; }
        vol = mybox1.volume();
        S.o.p ("volume of mybox1 is" + vol);
        vol = mybox2.volume();
        S.o.p ("volume of mybox2 is" + vol);
        vol = mybox3.volume());
        S.o.p("volume of mybox3 is"+vol);
    }
}
```

Method overloading :- having two or more methods in a class with same name with diff arguments is known as method overloading

Program

```
public class Sum{
    public int sum(int x, int y, int z){
        return(x+y+z);
    }
    public int sum(int x, int y, int z){
        return (x+y+z);
    }
public static void main(String[] args){
    Sum s = new Sum();
    Sy.o.p(s.sum(10,20));
    S.o.p(s.sum(10,15));
    s.o.p(s.sum(15,20));
}
}
```

Method overriding :- Using two or more subclass methods in a subclass. Such that the methods have same name & same signature is called method overriding.

Program

```
class parent{
    void show(){
        S.o.p("Parent's show()");
    }
}
class child extends Parent {
    @Override void show(){
        S.o.p("child's show()");
    }
}
class Main{
    P S V m(String[] args){
        Parent obj1 = new Parent();
        obj1.show();
        Parent obj2 = new Parent();
        obj2.show();
    }
}
```

Out.

Parent's show().
child's show().

Interface :- Interface is like a superclass. It contains final variables & abstract methods only. [multiple inheritance is used both interface bcz it doesn't implement on multiple inheritance]

*In this we use implement keyword to extend the class

Syn

```
interface {
    // variable declaration;
    // method
}
```

```
Public Interface Animal {
    void eat();
    void sleep();
}
Public class Dog implements Animal{
    @override
```

(like polymorphism)

Abstract class :- A class which is declared as abstract is known as Abstract class.

- Abstract class must be declared with abstract keyword.
- It compiles with constructor & static method.

Syntax

```
abstract class Shape
{
    int color;
    // code
    abstract void draw();
}
```

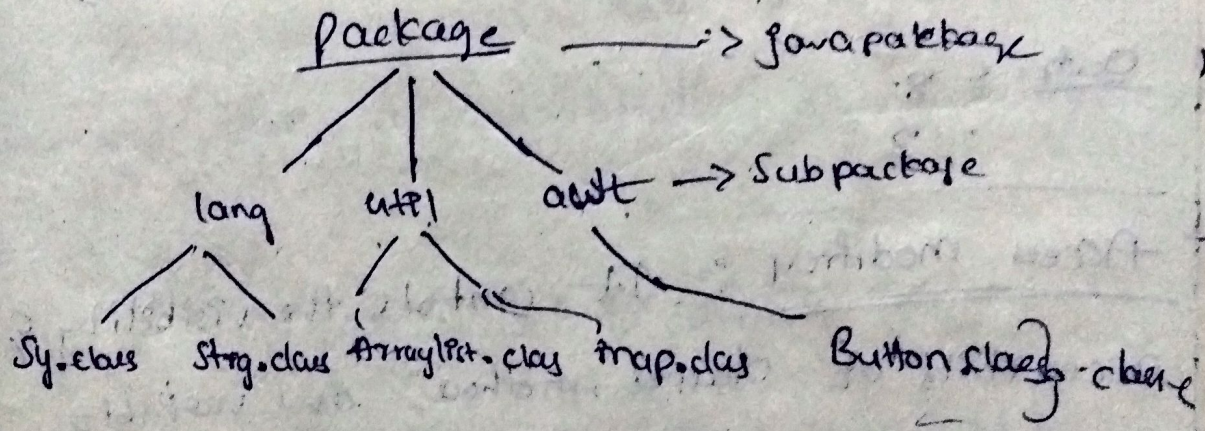
Program

```
public class abstract class Animal{
    public abstract void eat();
    public void sleep(){
        Sy.o.p("The Animal is Sleeping");
    }
}
public class Dog extends Animal {
    @override public void eat(){
        S.o.p("The dog is eating");
    }
}
public class Main{
    Public static void m(String[] args){
```

```
    Animal myDog = new Dog();
    myDog.eat();
    myDog.sleep();
  }
}
```

Packages : Package is a collection of similar type of classes, interfaces and subpackages.

Ex: vegtable markets.



Program

```
import java.util.Animal.*;
class Animals {
  pu s v m(String[] args) {
    one obj = new obj();
    obj.Dog();
  }
}
```

Output:

```
package import java.util.calculator.*;
public class CalculatorApp{
    psvm(String[] args){
        Calculator calc = new Calculator();
        int sum = calc.add(5,3);
        int difference = calc.substract(5,3);
        Sy.o.p("Sum:" + sum);
    }   Sy.o.p("Difference:" + difference);
}
```

Out :- 8
       2

Access modifiers :- It controls the visibility & accessibility of classes, methods and variables

4 types, public, private, protected, default.

- Same file package -

```
int product = calc.multiply(5,3);
S.o.p("Product:" + product);
```

import java.util.*;

3-UNIT

Exception handling :- It is a meahanism which can handle run time errors. with a normal flow condition. will be maintained.

try & catch, throw, throws finally.

~~Catch~~

try:- try is reveseved key word it contains block of statements which are doubt full

try: It is a keyword. used to create block statment which are doubtfull of generating exep

Syntax:-
```
try
{
  // code (doubtful code)
}
```

Catch :- It is a keyword which is used to handle the exception.

Syntax catch (Type of exception excepn obj)
```
{
  = handling code
}
```

Throw:- It is a keyword which is used to create an exception and throw it explicity

Syntax:- throw new exception type("content");

Throws: It is a keyword ~~which~~ It contains list of the type of exception that a method may throw

Syntax:- return type methodname (para) throws Exception list
```
{ //body
}
```

Program

```
import java.lang.*;
class ArithmeExceptionExample{
    ps vm(String[] args){
        int x=10, y=2, z;
        try{
            z=x/y;
            Sy.d.p(z);
        }catch(Arithmeticexception e){
            S.O.p("Division by two exception");
        }
        S.O.p("program over");
    }
```

out: 5
program over.

---

final Block :- It is a keyword used to create a block that will be executed after a try/catch block whether exception is handle or not

* If catch block is excute after that finally block will be executed.
* If no exception, the finally block is executed.

-> It follows a try block or catch bolck.

Syntax :-

① try
{
// doubtfull code
}
finally
{
exception code.
}

② try
{ // code
}
catch(ex)
{ code
}
finally
{ // code
}

③

Program

```
Import java.lang.*;
Class ExceptionExample{
  P S V m(String[] ar){
    try
    { int x, y, res;
      x=10
      y=2
      res=x/y;
    }
      S.O.P("res");
    }
  catch(ArithmeticException e){
    S.O.P("Division by zero");
  }
  finally{
    S.O.P("hello this is finally block");
  }
  S.O.P(program is over);
}
```

Throw vs Throws

Throw	Throws
→ It is a keyword used to explicitly throw an exception.	It is a keyword used to declare an exception.
→ void main() { throw new ArithmeticEx.(); }	→ void main() throws // code }
→ Checked Exception cannot be propagated using throw	→ Checked Exc can be propagated with throws.
→ Throw is followed by instance	→ Throws is followed by class
→ Throw is used within a method	→ Throws is used with method signature.
→ we cannot throw multiple exception	→ we can declare multiple Exc.

Exc: two types

- → Uses defined exception
- → Builtin exception:
  - → Checked Exc
    - → ClassNotFoundEx
    - → FileNotFoundEx
    - → IO Ex
    - → InterruptedEx
    - → SQLEx
  - → Unchecked Ex
    - → ArrayIndexOutOfBoundEx
    - → Arith Ex
    - → NullPointer Ex
    - → ArrayStore

Use dd

~~Class MyEx ext Exception {~~

~~Pu s vm (String[] args){~~

~~try{~~

~~Dat a~~

```
Class MyEx extends Ex{
    public MyException(String{
        Super(s)
    }
}
Class Main{
    p s v m (String[] args){
        try{
            throw new MyException(a);
        } catch(MyExe e){
            S.O.p("caught");
        }
    S.O.p (e.getMessage());
```

Multithreading is proces of executing multi threads Simultaneously.

Program

```
p class MultiThread extends Thread{
    private String thread
```

Prg

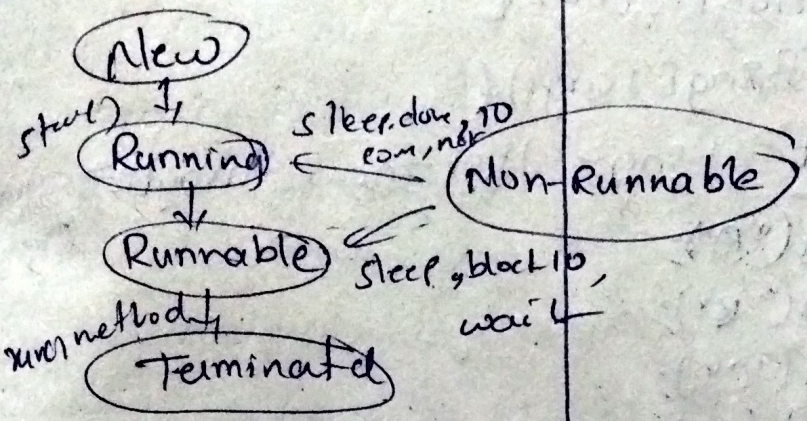
```
public class MultiThreadExample1
    p s vm(String[] args){
        MyThread thread1 = new MyThread(Thread1
        MThread thread2 = new MyThread(Thred2)
        thread1.start();
        thread2.start();
        try{
            thread1.join();
            thread2.join();
        } catch(InterruptedException e){
                s.o.p("Main thread interrupted.");
            }
        s.o.p("Main thread is finished);
    }
}
```

Output

```
Thread1 - count 1
Thread2 - count 1
    :
    :
Thread1 - count 5
Thread 2 = count 5
Thread 1 - finish
Thr 2 finish
main thread is finished
```

Multiple processes and multiple threads

Life cycle of Thread



Thread :- Thread class provides constructor and methods to create and perform operations on thread. It extends the object class and implements Runnable interface

Program

```
pu class Multi extends Thread{
    public void run{
    S.o.p ("thread is running...");
    }
    public s v m(String[] args){
        Multi t1 = new Multi();
        t1.start();
    }
}
```

o/p thread is running....

4 UNIT

Array list :- class uses a dynamic array for storing the elements

```
import java.util.*;
public class ArrayListEx1{
    pub s v m(String[] args){
        ArrayList<String> list=new ArrayList<String>();
        list.add("A");
        list.add("B");
        list.add("C");
        list.add("d");
        S.o.p(list);
    }
}
```

Linked List:- class uses a doubly linked list to store elements

```
LinkedList<String> list= new LinkedList<Str>();
list.add("May");
```

Hashset:- class is used to create a collection that uses a hash table for storage.

```
Hashset<String> list = new HashSet<String>();
```

Treeset :- class implements the set interface that uses a tree for storage.

- treeset class has unique elements
- treeset class doesnot allow null element
- It is non synchronized
- maintain acending order

TreeSet<String> set = new TreeSet<String>();

HashMap:

- Hash Map contains values based on keys
- contains unique keys
- non synchronized
- maintains no order
- It may have one null key and multiple null values

```
import java.util.*;
public class HashMapExample {
    public static void main(String args[]) {
        HashMap<Integer, String> map = new HashMap
                                    <Integer, String>();
        map.put(1, "Mango");
        map.put(2, "Apple");
        map.put(3, "Banana");
        map.put(4, "Grapes");

        System.out.println("Iterating Hashmap...");
        for(Map.Entry m : map.entrySet()) {
            System.out.println(m.getKey() + " " + m.getValue());
        }
    }
}
```

```
System.out.println("Iterating HashMap");
for(Map.Entry m : map.entrySet())
System.out.println(m.getKey() + " " + m.getValue());
```

```
System.out.println("Iterating HashMap");
for(Map.Entry m : map.entrySet()) {
System.out.println(m.getKey() + " " + m.getValue());
```

TreeMap :- class is a red-black tree based implementation. It provides strong key value

- It contain unique elements
- It is non-synchronized
- Java it maintain ascending order

```
TreeMap <Integer, String> = new TreeMap<Integer, Str>
    for(Map.Entry m = map.entrySet()) {
    Sso.p(m.getkey()+" "+m.getvalue());
} }
```

---

legacy :- classes and interfaces that formed the collections framework in the order version of java are known as legacy classes

- legacy classes are synchronized.
- java.util.package is defined to following legacy classes

legacy classes

- Hashtable
- vector class
- Stack class

Hashtable
vector class
Stack class

Hashtable :- used to create a hashtable.
is initialized with elements of a map.
capacity of the hashtable is twice present in
map

Pr

```
import java.util.*;
class HashtableExample {
  p s v m(String[] args){
    Hashtable<Integer, String> student = new
                    Hashtable<Integer, String>();
    student.put(new Integer(101), "Emma");
    Student.put(10, "Apple");
    Student.put("20, Banana");
    Student.put("20, Grapes");
    Student.put("40, mangos");

    Set dataset = student.entrySet().iterator();
    while(iterate.hasNext());
    {
      Map.Entry = (Map.Entry) iterate.next();
      S.o.p(map.getKey() + " " + map.getValue());
    }
  }
}
```

```
Set dataset = student.entrySet();
while(iterate.hasNext()) {
Map.Entry m=(Map.Entry) iterate.next();
S.o.p(m.getKey() + " " + m.getValue());
```

Stack class :- Stack class extends vector class which follows LIFO principle for the elements.

→ It has five methods:

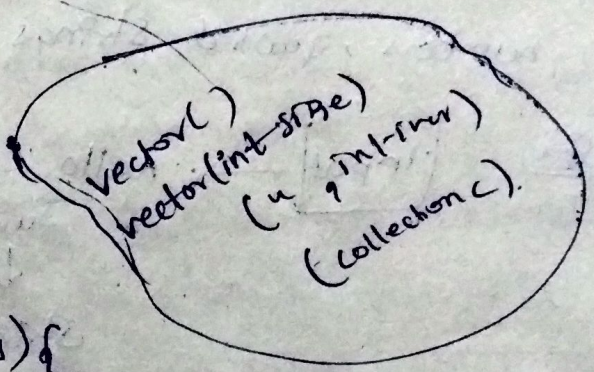
pop()
push()
peek()
search()
empty()

Prgm

```
import java.util.*;
Public class StackExample{
    P s v m (String[] args){
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        Stack.push(2);
        Stack.push(3);
        Stack.push(4);
        Stack.push(5);
        while(!stack.isEmpty()){
            S.o.p(stack.pop());
        }
    }
}
```

vector class :- vector class is a special type of ArrayList that defines a dynamic array. ArrayList is not synchronized while vector is synchronized.

-> vector()
-> vector(int size)
-> vector (int size, int incr)
-> vector (collection c)




```
import java.util.*;
public class VectorExample{
    public static void main(String[] args){
        Vector<String> vec = new Vector<>();
        vec.add("Mango");
        vec.add("Grapes");
        vec.add("Banana");
        vec.add("Pineapple");
        vec.add("Apple");
        Enumeration<String> data = vec.elements();
        while(data.hasMoreElements())
        {
            System.out.println(data.nextElement());
        }
    }
}
```

```
Enumeration<String> data = vec.elements();
while(data.hasMoreElement()).
{
    s.o.p(data.nextElement());
}
```

More utility classes

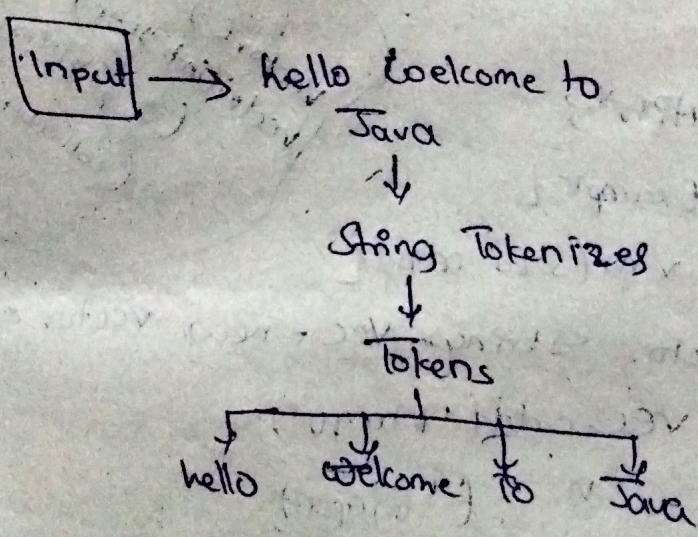
String Tokenizers in Java

-> The java.util.StringTokenizer class allows to break string into a token.

-> It is simple to breake a string.

-> It is legacy class of java.

-> It doesnot provide the facility to differentiate numbers, quoted strings, identifiers etc.

Ex 

There are 3 types of contructors in StringTokenizer

-> StringTokenizer(String str) : it creates ST with the specified string

-> StringTokenizer(String str, String delim : It creates ST with specified string & delimites

-> StringTokenizer(String str, String delim, boolean returnvalue) : It creates StringTokenizer with specified string, delimites and returnvalue. If returnvalue is true, delimites character are considers as tokens. if it is false, delimeters characters will be seperate tokens.

Methods of stringTokenizer

→ boolean hasMoreTokens() : It checks. if there is more tokens available.

→ boolean hasMoreElements() : It is the same as the hasMoreTokens() method.

→ String nextToken() : It returns the next token from the StringTokenizer object.

→ ~~String hasMore~~

→ String nextToken(String delim) : It return the next token based on the delimiter.

→ object nextElement() : It is the same as nextToken() but its return type is object

→ int countTokens() : It returns the total number of tokens.

Program

```
import java.util.StringTokenizer;
public class simple{
    p s v m(S[] args){
        StringTokenizer st= new StringTokenizer
                ("my name is bomma",");
        while(st.hasMoreTokens()) {
            S.op(st.nextToken());
        }
    }
}
```

Output

```
my
name
is
bomma.
```

Java random class is used to generate a Stream of pseudorandom numbers.

pseudo random no.

Methods in Random class

doubles(): Returns an unlimited streams of pseudorandom double values.

ints(): Returns an unlimited steams of Pseudorandom int values.

longs(): Returns an unlimited stream of pseudorandom long values.

next(): generates the next pseudorandom numbers.

Program

```
import java.util.Random;
public class JavaRandomExample{
    p s v m(S[] args){
        Random random = new Random();
        S.o.p("Random Integer value:"+random.next());
        long seed=20;
        random.setSeed(seed);
        S.o.p("Seed value:"+random.next());
        Long val=random.nextLong();
        S.o.p("Random long value:"+val);
    }
}
```

double() ints() longs() next()

O/P

Random Integer value: 129402
Seed value = -1150247
Random long value: -7322574

Java Scanner:- In java, Scanner is a class in java.util. package used for obtaining the input of the primitive types like int, double, etc... and strings.

Constructors:-

1. Scanner(File source): It constructs a new Scanner that produces values scanned from the specified file.
2. Scanner(String source): It constructs a new Scanner that produces values scanned from the specified string
3. Scanner(Readable source): It constructs a new Scanner to produces the values scanned from specified source.

Methods:-

1. void → close(): It is used to close this Scanner
2. int → nextInt(): It scans the next input of the int.
3. String → nextLine():
4. double → nextDouble(): It scans the next token of input as a double
5. float → nextFloat(): It scans the next token of the input as a float
6. Short → nextShort(): It scans the next token of the input as a short
7. long → nextLong(): It scans the next token of the input as a long

Prgm

```
import java.util.*;
public class ScannesExample{
    p s v m(S[] args){
        Scanner in = new Scanner(System.in);
        S.o.p("Entee your name:");
        String name = in.nextLine();
        S.o.p("Name is:"+name);
        in.close();
    }
}
```

o/p

Enter your name :- Dharani
Name is : Dharani

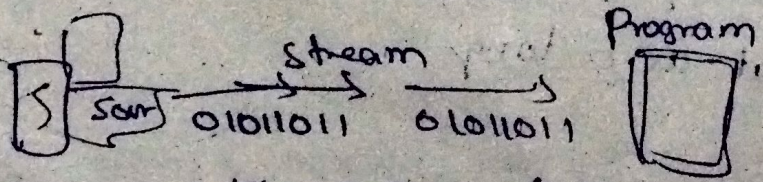
File

Stream class :- Stream is a sequence of data. In java a stream is composed of bytes. Its called Stream.

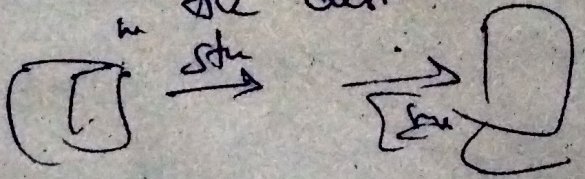
In this there are 3 streams

* System.out : Standard output stream
* System.in : standard input stream
* system.err : Standard error stream.

InPutStream : InputStream is used to read the data from source.



OutPutStream : OutputStream is used to write the data from the dest



It has two types:

* byte ~~Character~~ stream: it provides convenient for handling i/o of bytes.
* Character ~~byte~~ Stream: " " " " " i/o of char. stream uses unicode

By Str i/p.

* int available() → Returns the numb of bytes of i/p currently available for reading
* void close(): closes the i/p source.

Output Stream:

* void close(): closes the output Stream
* void write(): writes a single byte on Output Stream.

Characters stream:

Method's of Reader class

int read(): This method reads the char from i/p Stream

int read(char[] ch): This methods reads the char from the i/p stream & store them in char array

Methods of writer class

void write(int c): This method write the char to o/p stream

void write(char[] arr): This method writes the char arry to the o/p stream.

byte It provides to convenient to handle i/o