DATE

POP
- pop stands for procedure oriented programing.
- pop uses top-to-bottom approach.
- due to pop, there can be code complexity
- due to pop, there is no security
- due to pop, we can not execute code as per requirement
- by using pop, we can create an appln, but we can not achieve requirement of end-user

OOP
- oop stands for object oriented programing structure.
- uses bottom-to-top approach
- due to oop, we can remove code complexity
- due to oop we can provide security
- due to oop we can execute code as per requirement
- by using oop, we can create an appln & we can achieve any requirement of end-user

POP
eg.

```
class mainclass
{
    public static void main (String args[])
    {
        int n=5;
        for (int i=1; i<=n; i++)
        {
            for (int j=1; j<n; j++)
                s.o.p (" ")
            for (int j=1; j<=(i*2)-1; j++)
                s.o.p ("*");
            s.o.println ();
        }
    }
}
```

OOP

eg:

```
class demo
 {
    void m1()
     {
       int n=5
       for(int j=1; i<=n; i++)
        {
         for(int j=i; j<n; j++)
           s.o.p(" ");
         for(int j=1; j<=(i*2)-1; j++)
           s.o.p("*");
         s.o.println();
        }
     }
 }

class mainclass
 {
   public static void main(String args[])
    {
      demo d1 = new demo();
       d1.m1();
       d1.m1();
       d1.m1();
    }
 }
```

JDK vs JRE vs JVM

JDK

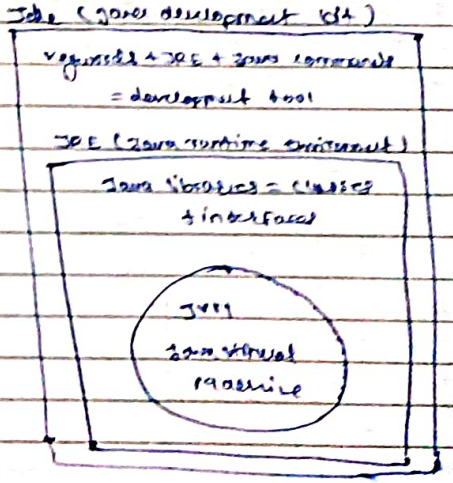
- Jdk stands for java development kit.
- Jdk contains development tools
- development tool contains keyword + JRE + java commands
- Jdk is responsible to build java appln

JRE

- JRE stands for java runtime environment
- JRE is a set of library, which can be used to execute java appln
- contains classes & interfaces
- JRE responsible to execute java appln

JVM

- JVM stands for java virtual machine
- JVM is a virtual machine physically not available
- JVM is a set of classes & interfaces, which is responsible to execute class file.
- JVM is actual responsible to execute java appln



- classloader subsystem
- Classloader subsystem is a super class in JVM
- — is a parent class of BootstrapClassLoader subsystem
- It is responsible to load & execute class file

- Bootstrap class loader subsystem
- Bootstrap classloader subsystem is a child class of class loader subsystem
- BootstrapClassLoader subsystem is parent class of Extension classloader subsystem
- — is responsible to deploy running environment is require rt.jar file

- Extension classloader subsystem
- — is a child class of BootstrapClassLoader subsystem
- — is a parent class of Appclassloader subsystem.
- — is responsible to check given class file.

- Appclassloader subsystem
- — it is a child class of extension classloader subsystem.
- — is responsible to load a class file from source folder to JVM
- to load a classfile from source folder to JVM it uses 3 notions
- we can initialized system variable implicitely & explicitely too.
- to set explicitely, we can use set path command

Types of memory

1) class Area memory
- class area memory also known as a native memory
- -v- can be used to store structure of a class
- here non-static element of a class will be stored during class loading process by using class name
- structure of a class will be stored one in a lifecycle.

2) Heap area memory
- heap area memory also known as main memory in java
- heap area memory consumes special memory like class area memory, stack, area, PC, JNI, constant pool, thread pool, metaspace etc.
- heap area memory can be used to store objects of a class
- heap area memory consumes approx 1/4 of RAM
- non-static element of class can be considered in heap area memory

3) stack area memory.
- stack area memory can be used to execute blocks in LIFO
- here blocks means method, loop, cond^n, constructor initializers, synchronized block, try, catch, finally etc.
- after exist from each block, stack area memory destroy their block
- hence local variable can-not accessible from outside their block.
- hence local variable in stack area memory.
- stack area memory considered as a faster than heap area memory.

4) PC Register
- PC Register ~~can~~ stands for program counter
- It is a set of instruction regarding memory mgmt
- by using PC register we can manage main & nursery memory.
- to increase main memory
  java -xms1g className
  java -xmx1g classname
- to increase nursery memory
  java -xns1gm classname.
  java -xnx10m className.

5) JNI
- JNI stands for java native environment
- JNI is responsible to execute external java appl^n

Class
- to create a class we can use class keyword
- In java class name should be capitalized & unique
- In java class can be used to store data or process the data.
- In java class contains variable methods, constructors & initializers.

Variable
- variable in java is a container, which can be used to store data temporary
- In java variable should be identical & decapitalize
- variable also considered as member or property of a class

Types of variable
① Non-static (instance) variable
- to declare non-static variable we don't need to used static keyword.
- Non-static variable can be declared within a class & outside the method
- non-static variable initialized during object creation.
- hence it can be access by object only
- non-static variable separate for all all object.
- non-static variable store in heap area memory.
- non-static variable also known as instance variable

② static variable.
- to create static variable we can used static keyword
- static variable can be declare within a class & outside the method.
- static variable generate during class loading process
- Hence it can be access by using class name directly
- static variable common for all objects.
- static variable store in meta-space during class loading by using class name.

③ local variable
- local variable can be declare within method or any block.
- In Java all blocks execute in stack area memory
- ~~local variable can~~ exist from each block stack area memory destroy that block
hence local variable can not be accessible from outside that block.
- local variable can be considered in stack area memory
- Note: local variable has to setup initialized before used.

- Note: There is NO global variable concept in java.

eg.
```
class Demo
{
    int a; // non-static variable
    static int b; // static variable
    void m1()
    {
        int c = 100 // local variable
```

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

class mainclass
{
    public static void main(String args[])
    {
        Demo d1 = new Demo();
        Demo d2 = new demo();
        d1.a = 10;
        d1.b = 20;
        System.out.println("d1.a = "+d1.a);
        ~ ("d1.b = "+d1.b);
        ~ ("d2.a = "+d2.a);
        ~ ("d2.b = "+d2.b);
    }
}
```

Methods

- In java method also known as flow of behaviour
- Method can be used to execute repeated statements
- due to method, we don't have to write repeated statement
- method executes in stack area memory in LIFO manner
- In java method name should be identical & descriptive

Syntax

```
return_type methodName (datatype para1, datatype para2, datatype para3)
    {
        // logic
    }
```

eg.

```
class Demo
{
    void m1() // method signature.
    {
        System.out.println("m1 method called");
    } //method body
    void m2()
    {
        System.out.println("m2 method called");
    }
}

class mainclass
{
    public static void main(String args[])
    {
        Demo d1 = new Demo();
```

```
        d1.m2();
        d1.m2();
        d1.m1();
    }
}
```

Types of methods

1) no argument no return value

- if method does not contain any argument, we shouldn't pass any argument to invoked method
- if method doesnot return any value, return type should be declared with void keyword.

e.g.

```
class Demo
{
    int a,b;
    void add()
    {
        System.out.println(a+b);
    }
}

class maininclass
{
    public static void main(String args[])
    {
        Demo d1 = new Demo();
        d1.a = 10;
        d1.b = 20;
        d1.add(); // 30
        d1.a = 40;
        d1.b = 20;  d1.add();
    }
}
```

2) with argument no return value.

- if method contains any argument, we can pass matching or supportive no. of parameter
- if method doesn't return any value, return type should be declared with void keyword.

e.g.

```
class Demo
{
    void add(int a, int b)
    {
        System.out.println(a+b);
    }
}

class maininclass
{
    public static void main(String args[])
    {
        Demo d1 = new Demo();
        d1.add(10,20);
        d1.add(40,70);
        d1.add('A','B');
        d.add(12,'A');
    }
}
```

3) no argument with return value.

- if method does not any argument we shouldn't pass any argument to invoked method.
- if method returns any value, value can be returned only using return keyword.

- return value & return type should be matching of supportive.
- method can return one value at a time

e.g.

```
class Demo
 {
   int a, b;
   int add()
    {
      return a+b;
    }
}

class mainclass
 {
   public static void main (String args[])
    {
      Demo d1 = new Demo();
      d1.a = 10
      d1.b = 20
      System.out.println (d1.add());
    }
}
```

4) With argument with return value.
- if method contains any argument, we can pass matching or supportive no. / parameter
- if method returning any value, value can be return by using return keyword.

- return value & return type should be matching of supportive
- method can return only value at a time

e.g.

```
class Demo
 {
    int add(int a, int b)
     {
        return a+b;
     }
}

class mainClass
 {
    public static void main (String args[])
     {
        Demo d1 = new Demo();
        S.O.p(d1.add(10,20));
     }
}
```

WAP to add sum of digit at the end of number

21

```
class Demo {
  int addSumAtEnd (int n) // 23575
      int sum=0, temp=n;
       while(n!=0)
        {
         sum+=n%10;
         n/=10;
        }
       [redacted];
       if (sum<10)
```

```
        temp *= 10;
        else
            temp *= 100;
        temp *= sum;
        return temp;
    }
}
```

WAP

```
class Demo
{
    int addSumFirst(int n)
    {
        int sum = 0, temp = n;
        while(n!=0)
        {
            sum += n%10;
            n /= 10;
            count ++;
        }
        sum *= Math.pow(10, count);
        sum += temp;
        return sum;
    }
}
```

```
# Wap to add highest digit at the end.
class Demo
{
    int addDigitAtLast(int n)
    {
        int max=0, temp=n;
        while(temp!=0)
        {
            int num = temp%10;
            if(num>max)
                max = num;
            temp /= 10;
        }
        n*=10;
        n += max;
        return n;
    }
}
```

```
* Shift last to first
class Demo
{
    // shift last to first
    int shiftLastToFirst(int n)
    {
        int last = n%10;
        n /= 10;
        int temp = n;
        while(temp!=0)
        {
            last *= 10;
            temp /= 10
        }
    return last+n;
```

```
// shif first to last
int shiftfirsttoleast (int n)
{
    int ~~fnd=0~~, count = 1, temp = n;
    while(temp > 9)
    {
        temp /= 10;
        count *= 10;
    }
    n%= count
    n*=10;                    or  return ((n % count)*10) + temp;
    n+= temp;
    return n;
}

// shif first & last at last ; digit
int shiffirstandleastAtlast (int n)
{
    if (n > 99)
    {
    int last = n % 10, count = 1;
    n/10;
    int temp = n;
    while (temp > 9)
    {
        temp /= 10;
        count *= 10;
    }
    n %= count;
    n*= 10;
    n+= temp;
    n*= 10;
    n+= last;
}
    return n;
}

// palindrome or not
boolean ~~fff~~ checkpalindrome (int n)
{
    int sum = 0, temp = n;
    while (temp != 0)
    {
        sum *= 10;
        sum += temp % 10;
        temp /= 10;
    }
    return (sum == n) ? true : false;
}

// find 58th palindrome number
int findNthpalindromeNumber (int ~~n~~)
{
    int a = 0, count = 0
    while (true)
    {
        if (checkpalindrome (a))
            count ++;
        if (count == last)
        {
            return a;
        }
        a++;
    }
}
```

WAP to check given number is prime or not

```
boolean checkPrime(int n)
{
    boolean status = false;
    if(n==0 || n==1)
        status = true;
    else
        for(int i=2; i<=n/2; i++)
            if(n%i==0)
            {
                status = true;
                break;
            }
    return !status;
}
```

// 75th prime no

```
int findNthPrimeNumber(int last)
{
    int a=0, count=0;
    while(true)
    {
        if(checkPrime(a))
            count++;
        if(count==last)
        {
            return a;
            a++
        }
    }
}
```

WAP check given no is strong or not

```
class Demo
{
    boolean checkStrongNumber(int n)
    {
        int temp=n, sum=0;
        while(temp!=0)
        {
            int fact=1;
            for(int i=1; i<=temp%10; i++)
                fact*=i;
            sum+=fact;
            temp/=10;
        }
        return (sum==n)?true:false;
    }
}
```

Armstrong Number

```
class Demo
{
    boolean checkArmstrong(int n)
    {
        int length=0, temp=n, sum=0;
        while(temp!=0)
        {
            temp/=10;
            length++;
        }
        temp=n;
        while(temp!=0)
        {
```

```
            sum += Math.pow(temp%10, length);
            temp/=10;
        }
        return (sum==n) ? true : false;
    }
}
```

No. is perfect or not

```
class Demo
{
    boolean checkperfect(int n)
    {
        int sum=0;
        for(int i=1; i<=n/2; i++)
            if(n%i==0)
                sum+=i;
        return (sum==n) ? true : false;
    }
}
```

palindromic prime or not

```
class Demo
{
    boolean checkpalindromicprime(int n)
    {
        if(checkpalindrome(n))
            if(checkprime(n))
                return true;
            return false;
        }
}
```

prime & twist or not

```
boolean checkprimeandTwist(int n)
{
    if(checkprime(n))
        if(checkprime(reversenumber(n)))
            return true;
        return false;
    }
```

Method overloading.

- In java method overloading is a concept, where one class contains 2 or more method with same and different parameter
- In java method overloading is a concept, where method can be execute on the basis of their parameter
- method overloading can be considered as a compiletime polymorphism.

eg.

```
class Demo
{
    void m1()
    {
        S.o.p("no arg m1 method called");
    }
    void m1(int a)
    {
        S.o.p("int arg m1 method called");
    }
    void m1(int a, int b)
    {
        S.o.p("int int arg m1 method called");
    }
```

```
class mainclass
{
    public static void main (String args[])
    {
        Demo d1 = new Demo;
        d1.m1(); // no arg m1 method called
        d1.m1(10); // int arg m1
        d1.m1(10,20); // int int arg m1
    }
}
```

- method overloading is useful for end-user.
- we cannot prevent method overloading.
- NOTE: we cannot perform method overloading on the basis of return type due to method auto-promotion

- what is method auto-promotion?
- In java method auto-promotion is a concept where if method couldn't find matching parameter method auto-promote their parameter to invoked suppositive method.
- due to method auto-promotion, we don't have to overload method unnecessarily
- due to method auto-promotion, we can not overload method on the basis of return type.

e.g.

```
class Demo
{
    void add(double a, double b)
    {
        System.out.println("double double method called");
    }
}

class mainclass
{
    public static void main (String args[])
    {
        Demo d1 = new Demo();
        d1.add(10,20);
        d1.add(10,16.5);
        d1.add('A','B');
        d1.add('A',16.5F);
    }
}
```

CONSTRUCTOR (special method)

- In java CONSTRUCTOR is a special method, which contains same name of a class.
- constructor invocked implicitely during object creation.
- hence constructor cannot be static
- constructor does not have return type, because it returns instance of a class by default.
- constructor can be used to initialized instance & static variable of a class.

eg.

```
class Demo
{
    int a, b;
    void display()
    {
        System.out.println("a="+a+" b="+b);
    }
}
```

```
Demo()
{
    a=20;
    b=40;
    System.out.println("constructor called");
}

class mainclass
{
    public static void main(String args[])
    {
        Demo d1 = new Demo();
        Demo d2 = new Demo();
        Demo d3 = new Demo();
        d1.display();
        d2.display();
        d3.display();
    }
}
```

NOTE:- To prevent object creation constructor declare private.

Types of constructor

1) Default constructor

- default constructor does not contain argument hence it also known as a no argument constructor.
- Java compiler provides default constructor by default if there is no constructor.
- Default constructor is invoked implicitly during object creation.
- default constructor can be used to initialized instance & static variable with default values.

2) User defined constructor

- user defined constructor contains argument. hence it is called as parameterized constructor.
- Java compiler does not provide user define constructor, we have to create our own user-defined constructor as per our requirement.
- user-defined constructor can be invoked explicitly by passing matching or supportive number / parameter.
- user define constructor can be used to initialized instance & static variable with user-defined values.

eg.

```
class Demo
{
    int a,b;
    void display()
    {
        S.O.P("a="+a+" b="+b);
    }
    Demo()
    {
        a=10;
        b=20;
        System.out.println("default const called");
    }
    Demo(int a, int b)
    {
        this.a=a;
        this.b=b;
        S.O.P("user-defined const called");
    }
}
```

```
class newclass
{
  public static void main (String args[])
  {
    Demo d1 = new Demo();
    Demo d2 = new Demo (20, 30);
    Demo d3 = new Demo (30, 40);
    d1.display();
    d2.display();
    d3.display();
  }
}
```

#Initializers.

- In Java initializers can be used to initialized properties of a class
- initializers can be used if there is no use of constructor

Types of initializers

1) instance block.

- instance block also known as a non-static block

Syntax

```
{
}//instance block
```

- instance block invoked implicitly during object creation before constructor.
- instance block can be used to initialized instance and static variable with default values only

2) Static block

- To create static block we can use static keyword.

Syntax:

```
static
{
}
```

- static block invoked implicitely during class loading process before constructor & instance block.
- static block can be used to initialized static variable with default values only.

e.g.

```
class Demo
{
  int a, b;
  void display()
  {
    S.o.p("a=" +a+ "b=" +b);
  }
  {
    a = 10;
    b = 20;
    S.o.p("instance block called");
  }
  static
  {
    S.o.p("static block called");
  }
  Demo()
  {
    System.out.println("default const called");
  }
}
```

```
class Mainclass
{
  public static void main (String args[])
  {
    Demo d1 = new Demo();
    Demo d2 = new Demo();
    Demo d3 = new Demo();
    d1.display();
    d2.display();
    d3.display();
  }
}
```

Object
- In java object is a class located in java.lang package.
- Object class is a parent class of all class hence it is considered as a super class.
- methods of Object class

1) hashCode()
   - hashcode() method can be used to represent hashcode of object.

2) toString()
   - toString() method can be used to object in string format.

3) equals()
   - equals() method can be used to compare objects of the basis of values

4) getClass()
   - getClass() method can be used to represent source of object

5) clone()
   - clone() method can be used to copy object.

6) finalize()
   - finalize() method used by garbage collector implicitly to destroy unreferenced object from app

7) wait()
   - wait() method can be used to put thread object into non-runnable state until notify() or notifyAll() method invocked.

8) notify()
   - notify() method can be used to free thread object from non-runnable state.

9) notifyAll()
   - notifyAll() method can be used to free all thread object from non-runnable state.

Object
- in java object is an instance of a class
- Object also considers as a reference, memory, instance resource of a class.
- by using object we can access elements of class.

Types of object creation Java

① By using new keyword

- In Java we can create object by using new keyword
- new keyword returns new memory of objects
- we can compare 2 objects by using == and equals() method.

NOTE:- == is a relational operator, which can be used to compare 2 objects on the basis of memory.

Equals() is a method of object class, which can be used to compare 2 objects on the basis of values

default implementation of equals() method will not work hence we have to override equals() method into child class.

eg.

```
class Demo
{
  int a,b;
  void display()
  {
    System.out.println("a="+a+"b="+b);
  }
public boolean equals(Demo d1)
{
  if(a==d1.a)
    if(b==d1.b)
      return true;
  return false;
}
}
```

```
class mainclass
{
  public static void main(String args[])
  {
    Demo d1=new Demo();
    S.O.p(d1.hashCode());
    Demo d2=new Demo();
    S.O.p(d2.hashCode());
    d1.display();
    d2.display();
    S.O.p(d1==d2);
    S.O.p(d1.equals(d2));
  }
}
```

② By using reference

- In Java we can create object by using reference old object to new one.
- this approach can be used to save memory.

eg.

```
class mainclass
{
  public static void main(String args[])
  {
    Demo d1=new Demo();
    Demo d2=d1;
    d1.a=10;
    d1.b=20;
    d1.display();
    d2.display();
    S.O.p(d1==d2);
    S.O.p(d1.equals(d2)); // true
  }
}
```

3) by using anonymous way
- in java anonymous object does not reference to any one
- anonymous object also considered as an unreferenced object.
- anonymous object can be used if we want to access single element of a class
NOTE: In java garbage collector destroy unreferenced object first.

```
class democlass
{
  public static void main (String args[])
  {
    Demo d1 = new Demo(); // reference
    d1.a = 10;
    d1.b = 20;
    d1.display();
    new Demo().a = 20;
    new Demo().b = 40;
    new Demo().display();  // unreference
  }
}
```

4) by using method.
- In java we can create object by using method also
- we can use forName() & newInstance() method of class class
- forName() static method of class class, can be used to load given class file.
- FORNAME() the method throws with ClassNotFound Exception
- newInstance() method of class class, can be used to create object of loaded class file.
- newInstance() method throws with InstantiationException & IllegalAccessException
- NOTE :- the approach can be used to create an object of external class file.

```
class mainclass
{
  public static void main (String args[])
  {
    System.out.println("program started");
    try
    {
      Class c = Class.forName("Demo");
      Demo d1 = (Demo)c.newInstance();
      d1.display();
    } catch(ClassNotFoundException | InstantiationException
    | IllegalAccessException e){
        System.out.println(e);
    }
    System.out.println("program ended");
  }
}
```

5) by using copy constructor
- in java we can create object by using copy constructor also
- copy constructor is an user-defined constructor which contains class type parameter.
- by using constructor we can copy object on the basis of values

- copy constructor can be used to copy partial or complete object.

eg.

```
class Demo
{
  int a,b;
  void display()
  {
    System.out.println("a=" +a+" b="+b)
  }
  Demo() { }
  Demo(Demo d1)
  {
    a = d1.a;
    b = d1.b;
  }
}
class Mainclass
{
  public static void main(String args[])
  {
    Demo d1 = new Demo()
    d1.a = 10;
    d1.b = 20;
    Demo d2 = new Demo(d1)
    d1.display();
    d2.display();
    S.O.P(d1 == d2);
  }
}
```

6) By using cloning process

- In Java we can create object by using cloning process also
- to perform cloning process, we can use clone() method of object class
- clone method throws with CloneNotSupported Exception
- to perform clonning process, class should be implemented with cloneable interface.

NOTE:- clonning process can be used if complete copy required.

eg.

```
class Demo implements Cloneable
{
  int a,b;
  void display()
  {
    S.O.P("a=" +a+" b="+b);
  }
  public Demo clone()
  {
    Demo d1 = null;
    try {
    d1 = (Demo) Super.clone()
    }
    catch (CloneNotSupportedException e){
      S.O.P(e);
    }
    return d1;
  }
}
```

```
class maincass
{
public static void main (String args[])
 {
    Demo d1 = new Demo();
    d1.a = 10;
    d1.b = 20;
    Demo d2 = d1.clone()
    d1.display();
    d2.display();
    s.o.p (d1 == d2)
  }
}
```

7) By using Singleton pattern
- In java we can create object we can create object Singleton pattern also
- Singleton pattern allows us to create single object only
- to prevent new object creat", we can declare constructor as a private.
- create static method, which will returns static objects and static object is common for all user
Note: Singleton pattern can be used to provide common resource like db connectn

```
=>
class Demo
{
  private static demo d1 = new Demo();
  int a, b;
   void display()
    {
  }  s.o.p("a =" +a+"b =" +b);
  private demo() { }
  static demo getObject()
    {
      return d1;
    }
}

class maincass
{
  public static void main (String args[])
   {
     Demo d1 = Demo.getObject()
     d1.a = 10;
     d1.b = 20;
     Demo d2 = Demo.getObject();
     d1.display();
     d2.display();
     s.o.p(d1 == d2);
   }
}
```

8) By using Shallow copy
- In java Shallow copy means 2 objects having same memory
- if we make changes in one object impact can be seen in other object too.
- Shallow copy is faster
- In java Shallow copy means can be achieved by using reference & Singleton pattern

deep copy
Serialization
deep

9) by using deep copy
- In java deep copy means 2 objects having same values but different memory
- if we make changes in one object impact can not be seen in other object.
- deep copy is slower
- In java deep copy can be acheived by using copy constructor & clonning process.

10) by using Serialization
- In java serialization is a concept, where object convert into bytestream
- reverse operation of serialization considered as a deserialization
- serialization can be used to send data one end to other
- to perform serialization, we can use serializable interface.

Array
- array can be used to store multiple values
- array can store multiple values index bayis.
- by using index, we can set or get element from array
- array index always start from 0 to length - 1

eg. int a[] = new int[5];
2000

2000-2003	2004-2007	2008-2011	2012-2015	2016-2019
0	0	0	0	0
0	1	2	3	4

Locatn = start_loc + (index * datatype);

- if we try to access index of an array, which is not present we will get ArrayIndexOutOfBoundException.
- To determine size of an array, we can use length property.
- if we try to create array in -ve size, NegativeArray one exception will be there

WAP to create array and display all elements

```
class Mainclass
{
    public static void main (String args[])
    {
        int a[] = new int[5];
        a[2] = 10;
        a[4] = 40;
        a[1] = a[2] + a[3];
        S.o.p(a[0]);
        (a[1]);
        (a[2];
        :
        (a[5]);
    }
}
```

```
class mainclass
{
    public static void main (String args[])
    {
        int a[] = {10, 20, 30, 40, 50};
        for (int i=0; i<a.length; i++)
            System.out.println (a[i]);
                ~~~ (" - - - ");
        For (int i = a.length-1; i>=0; i--)
            s.o.p(a[i]);
    }
}
```

① WAP to find sum of array

```
class Mainclass
{
    public static void main (String args[])
    {
        int a[] = {17, 15, 22, 29, 14, 28, 10, 11, 15};
        int temp=0;
        For (int i=0; i<a.length; i++)
            int temp=0;
            temp += a[i];
        s.o.p(" sum is:" +temp);
    }
}
```

② WAP to find avg of array

```
class Mainclass
{
    public static void main (String args[])
    {
        int a[] = {17, 15, 22, 29, 14, 28, 10, 11, 15};
        int sum=0;
        double avg=0;
        for (int i=0; i<a.length; i++)
        {
            sum += a[i];
            avg = sum/a.length;
            s.o.p ("avg of array:" +avg);
        }
    }
}
```

avg of even no
avg of odd & even element
display greater number than avg of array

#

WAP to shift first to last element from array

```
import java.util.Arrays;
class Mainclass
{
    public static void main (String args[])
    {
        int a[] = {10, 20, 30, 40, 50};
        int temp = a[0];
        for(int i=0; i<a.length-1; i++)
            a[i] = a[i+1];
        a[a.length-1] = temp;
        System.out.println(Arrays.toString(a));
    }
}
```

WAP to shift last to first element from array

```
import java.util.Arrays;
class Mainclass
{
    public static void main (String args[])
    {
        int a[] = {10, 20, 30, 40, 50};
        int temp = a[a.length-1]
```

for(int i=0; i<a.length; i++)
a[i] = a[i+1]

OR

```
For(int i=0; i<a.length ; i++)
    a[i]=a[i+a.length];
    a[0] = temp;
System.out.println(Arrays.toString(a));
}
}
```

WAP to swap first and last element from array

```
import java.util.Arrays;
class Mainclass
{
    public static void main (String args[])
    {
        int a[] = {10,20,30,40,50};
        int temp = a[0];
        a[0] = a[a.length-1];
        a[a.length-1] = temp;
    System.out.println(Arrays.toString(a));
    }
}
```

without using third variable

```
import java.util.Arrays;
class mainclass
{
    public static void main (String args[])
    {
        int a[] = {10,20,30,40,50}
        a[0]+=a[a.length-1];
        a[a.length-1]=a[0]-a[a.length-1];
        a[0]=a[a.length-1];
    }
}
```

WAP to reverse an array without using 3rd variable

```
class Mainclass
import java.util.Arrays;
class Mainclass
{
    public static void main (String args[])
    {
        int a[] = {10,20,30,40,50}
        for (int i=0; i<a.length/2; i++)
        {
            a[i]+=a[a.length-1-i];
            a[a.length-1-i]= a[i]-a[a.length-1-i];
            a[i]-= a[a.length-1-i];
        }
    System.out.println(Arrays.toString(a));
    }
}
```

Second max

```
int max=a[0], smax=0;
for(int i=1; i<a.length; i++)
    if(a[i]>max)
    {
        smax=max
        max=a[i];
    }
    else
        if(a[i]>smax && a[i]!=max)
        smax=a[i];
```

WAP to find sum of 2 element is key = 20

```
import java.util.Arrays
class Mainclass
 {
   public static void main (String args[])
   {
      int a[] = {19, 11, 14, 21, 17, 24, 15, 24, 15, 13};
      int key = 40;
      for(int i=0; i<a.length-1; i++)
        for(int j=i+1; j<a.length; j++)
            if (key == a[i]+a[j])
      System.out.println (a[i]+"\t"+a[j]);
   }
}
```

Sum of 3 elements key = 60

```
import java.util.Arrays;
class Mainclass
 {
    public static void main (String args[])
    {
       int a[] = {19, 11, 14, 21, 17, 24, 18, 24, 15, 13}
        int key = 60;
       for(int i=0; i<a.length-2; i++)
       for(int j=i+1; j<a.length-1; j++)
       for(int k=j+1; k<a.length; k++)
           if (key == a[i]+a[j]+a[k])
      System.out.println (a[i]+"\t"+a[j]"\t"+a[k]);
    }
}
```

WAP to find adjacent values of array

input: {19, 4, 14, 21, 17, 24, 18, 28, 15, 13}
output: {3, 3, 7, 4, 7, 6, 10, 13, 2}

```
import java.util.Arrays;
class Mainclass
 {
   public static void main (String args[])
    {
       int a[] = {19, 11, 14, 21, 17, 24, 18, 28, 15, 13}
       int b[] = new int[a.length-1];
      for(int i=0; i<a.length-1; i++)
        b[i] = Math.abs(a[i]-a[i+1]);
       System.out.println(Arrays.toString(b));
    }
}
```

WAP to merge 2 array

```
import java.util.Arrays
class Mainclass
 {
    public static void main (String args[])
     {
       int a[] = {10, 20, 30, 40, 50}
       int b[] = {11, 22, 33, 44, 55, 66, 77}
       int c[] = new int[a.length + b.length];
      for(int i=0; i<a.length; i++)
          c[i] = a[i];
      for(int i=0; i<b.length; i++)
          c[a.length+i] = b[i];
      System.out.println (Arrays.toString (c));
     }
}
```

Merge one by one

```
import java.util.Arrays
class Mainclass
{
  public static void main (String args[])
  {
    int a[] = {10, 20, 30, 40, 50};
    int b[] = {11, 22, 33, 44, 55, 66, 77};
    int c[] = new int [a.length+b.length];
    if (a.length<=b.length)
    {
      for(int i=0; i<a.length ; i++)
      {
        c[i]= a[i];
        c[i*2] = a[i];
        c[(i*2)+1] = b[i];
      }
    for(int i= a.length; i<b.length; i++)
      c[(a.length*2) - (a.length-i)] = b[i];
    }
    else
    {
      for (int i=0; i<b.length; i++)
      {
        c[i*2] = a[i];
        c[(i*2)+1] = b[i];
      }
      for ( int i = b.length; i<a.length; i++)
      c[(b.length*2) - (b.length-i)] = a[i];
    }
    System.out.println (Arrays.toString (c))
  }
}
```

WAP to append sum & element at the end & element from array

input : {15, 171, 288, 23, 175, 97, 54}

output : {15, 1719, 28818, 285, 17513, 9715, 549}

```
import java.util.Arrays;
class Mainclass
{
  public static void main(String args[])
  {
    int a[] = {15, 171, 288, 23, 175, 97, 54}
    for (int i=0; i<a.length; i++)
    {
      int temp = a[i], sum=0;
      while (temp!=0)
      {
        sum += temp%10;
        temp/=10;
      }
      a[i]*= (sum<10)? 10: 100;
      a[i] += sum;
    }
    System.out.println(Arrays.toString(a));
  }
}
```

WAP to display nearest element from.

input : {15, 11, 28, 23, 17, 20, 9, 4, 21}

O/P : 20, 21

```
import java.util.Array
class mainclass
{
  public static void main (String args[])
  {
    int a[] = {15, 11, 28, 23, 17, 20, 9, 4, 21}
    for(int i=0; i<a.length; i++)
```

```
Arrays.sort(a);
int min = a[a.length-1];
for(int i=0; i<a.length-1; i++)
{
    int diff = Math.abs(a[i]-a[i+1]);
    if(min > diff && diff != 0)
        min = diff;
}
for(int i=0; i<a.length-1; i++)
    if(min == Math.abs(a[i]-a[i+1]))
        System.out.println(a[i]+" & "+a[i+1]);
}
}
```

WAP to separate even & odd element from array.

```
import java.util.Arrays;
class Mainclass
{
    public static void main(String args[])
    {
        int a[] = {17, 15, 14, 22, 11, 10, 13, 8, 12, 24, 21};
        int b[] = new int[a.length];
        int count=0;
        for(int i=0; i<a.length; i++)
            if(a[i]%2!=0)
                b[count++] = a[i];
        for(int i=0; i<a.length; i++)
            if(a[i]%2==0)
                b[count++] = a[i];
        System.out.println(Arrays.toString(b));
    }
}
```

WAP to sort each digit from element of array

i/p : {34678, 3456, 8987, 355, 897, 357, 6747, 26967}

o/p : {34678, 3456, 7889, 356, 789, 357, 4677, 26679}

```
import java.util.Arrays;
class Mainclass
{
    public static void main(String args[])
    {
        int a[] = {34678, 3456, 8987, 355, 897, 357, 6747, 26967};
        for(int i=0; i<a.length; i++)
        {
            int temp = a[i];
            int length=0;
            while(temp!=0)
            {
                length++;
                temp/=10;
            }
            int b[] = new int[length];
            temp = a[i];
            length=0;
            while(temp!=0)
            {
                b[length++] = temp%10;
                temp/=10;
            }
            Arrays.sort(b);
            length=0;
            for(int j=0; j<b.length; j++)
            {
                length*=10;
                length+=b[j];
            }
```

```
        a[i] = length;
    }
    System.out.println(Arrays.toString(a));
  }
}
```

WAP to reverse each element from array

input :-

o/p : { 87643, 6543, 7898, 568, 798, 753, 7476, 76962 }

```
import java.util.Arrays;
class MainClass
{
  public static void main(String args[])
  {
    int a[] = { ~ };
    for (int i=0; i<a.length; i++)
    {
      int sum=0, temp=a[i];
      while (temp!=0)
      {
        sum*=10;
        sum+=temp%10;
        temp/=10;
      }
      a[i]=sum;
    }
    System.out.println(Arrays.toString(a));
  }
}
```

WAP to fix array

input : { 6, 3, 1 }

o/p : 3

```
import java.util.Arrays;
class MainClass
{
  public static void main(String args[])
  {
    int a[] = {6, 3, 1, 2}
    double avg=0;
    for (int i=0; i<a.length; i++)
      avg+=a[i];
    avg/=a.length;
    int shift=0;
    if (avg%1==0)
    {
      for (int i=0; i<a.length; i++)
        if (a[i]<avg)
          shift+=avg-a[i];
    }
    else
      shift=-1;
    System.out.println("no. of attempt:"+shift);
  }
}
```

How to create array in java

1) by using new keyword

```
int a[] = new int[5];
a[0] = 10;
a[1] = 20;
a[2] = 30;
for (int i=0; i<a.length; i++)
  System.out.println(a[i]);
```

2) by using static way

```
int a[] = {10,20,30,40}
int b[] = {10,20,30,40};
for (int i=0; i<a.length; i++)
System.out.println(a[i]+"\t"+b[i]);
System.out.println(a==b);
```

3) by using reference

```
int a[] = {10,20,30,40}
int b[] = a;
for (int i=0; i<a.length; i++)
  System.out.println(a[i]+"\t"+b[i]
  System.out.println(a==b);
```

4) by using anonymous way

```
int value1 = new int[] {10,20,30,40}.length;
int value2 = new int[] {10,20,30,40}[2];
System.out.println(value1);
System.out.println(value2);
```

What is 2D Array

eg

```
int a[][] = new int[5][5];
for (int i=0; i<a.length; i++)
{
  for (int j=0; j<a[i].length; j++)
  {
    S.o.p(a[i][j]+"\t");
  }
  S.o.p();
}
```

OR

```
int a[][] = {{10,20,30},{20,30,30},{30,40,30}};
  for (int i=0; i<a.length; i++)
  {
    for (int j=0; j<a[i].length; j++)
    {
      System.out.println(a[i][j]+"\t");
    }
    System.out.println();
  }
```

eg.

```
class mainclass
{
  public static void main (String args[])
  {
    int a[][] = new int[5][5];
    int b=10;
    for (int i=0; i<a.length; i++)
    {
      for (int j=0; j<a[i].length; j++)
      {
        if (i==a.length-1-j)
        {
```

```
a[i][j] = b;
b += 10;
}
System.out.print(a[i][j] + "\t");
}
System.out.println();
}
```

* Jagged array

```
int a[][] = {{20}, {10,20}, {10,20,30}, {10}, {20,30,40,50,60}};
for (int i=0; i<a.length; i++)
{
for (int j=0; j<a[i].length; j++)
{
S.o.print(a[i][j] + "\t");
}
S.o.p();
}
```

how to take input from use

1) command line argument (CLA)

```
int a = Integer.parseInt(args[0]);
int b = Integer.parseInt(args[1]);
int c = a+b;
System.out.println(c);
```

- by using CLA, we can take input from user before execut^{n} of app^{n}

2) By using BufferedReader

- import - BufferedReader is a class which located in java.io package.
- instance of BufferedReader can be create by using InputStreamReader
- we can take input from user, by using readline method of BufferedReader
- readline() method throws IOExcep^{n}
- Note: BufferedReader is faster than Scanner

BufferedReader can be used to take input from user during exception & app^{n}

eg.

```
import java.io.BufferedReader;
        InputStreamReader;
        IOException
class MainClass
{
public static void main(String args[]) throws IOException
{
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
System.out.print("Enter value");
int value1 = Integer.parseInt(br.readline());
System.out.print("Enter value");
int value2 = Integer.parseInt(br.readline());
System.out.println(value1+value2);
}
}
```

3) by using Scanner

- Scanner is a class is located in java.util package
- Instance of Scanner class requires source & input
- Scanner provides methods like nextInt(), nextFloat(), nextDouble(), nextLong() etc.
- Scanner is a slower than BufferedReader, because it takes input from user & convert into required datatype.

eg

```
import java.util.Scanner;
class MainClass
{
    public static void main (String args[])
    {
        Scanner sc = new Scanner (System.in);
        System.out.print("enter value:");
        int value1 = sc.nextInt();
        System.out.print ("enter value");
        int value2 = sc.nextInt();
        System.out.println (value1+value2);
    }
}
```

eg:

```
import java.util.Scanner;
import java.util.Arrays;
class MainClass
{
    public static void main (String args[])
    {
        Scanner sc = new Scanner (System.in);
        System.out.print("enter size of array:");
        int size = sc.nextInt();
        int a[] = new int[size];
        for (int i=0; i<a.length; i++)
        {
            System.out.print("enter element a["+i+"]:");
            a[i] = sc.nextInt();
        }
        System.out.println (Arrays.toString(a));
    }
}
```

String.

- String is a class which is located in java.lang package
- String class can be used to represent a group of characters
- String internally uses character[] to represent a group of characters.
- String index always start from 0 & it ends with length-1
- to determine size of string we can use length() method
- we to get char from index, we can use charAt() method.

How to create String objects

① By using new keyword

```
String s1 = new String ("India");
System.out.println(s1);
```

② By using literal way

```
String s1 = "India";
System.out.println(s1);
```

3) by using reference

```
String s1 = "India";
String s2 = s1;
System.out.println(s1);
System.out.println(s2);
```

4) by using char[] array

```
char c[] = {'I','N','D'};
String s1 = new String(c);
System.out.println(s1);
System.out.println();
```

String is immutable

- in java string is immutable, here immutable means we can not make changes into it.
- String is immutable because string follows string constant pool

what is string constant pool

- string constant pool is a logical storage of object
- string constant pool allows unique references
- due to string constant pool, String saves a lot of memory
- due to string constant pool, string object should be compare by using equals() method not == operator.

Note: == is a relational operator, ~~which can be used~~ which is used to compare 2 objects on the basis of memory

- equals() method of object class, which can be used to compare 2 objects on the basis of value)
- due to string constant pool, string becomes immutable we cannot make changes into string object.

String methods

① toUpperCase

```
str.toUpperCase()
```

② toLowerCase()

```
Str.toLowerCase()
```

③ equals

```
str.equals("hello")
```

④ compareTo("abc")

⑤ contains

```
str.contains("ell")
```

⑥ indexOf()

```
Str.indexOf("i")
```

⑦ lastIndexOf(String str)

```
Str.lastIndexOf("i");
```

⑧ startsWith() & endsWith()

Modification

① replace (char oldchar, char newchar)
str. replace ('l', 'p')

② trim()
s. trim()

③ split
String s = "java, Python"
String [] arr = s. split (", ")

④ is empty()

replace()

mutable classes in java.

1) String Buffer
- String Buffer is a class, which is located in java. language package.
- String Buffer class not follow string constant pool
- Hence it is consider as a mutable class
- String Buffer does not provide concat method to perform concatenation
- to perform concatenation, we can use append() method.

eg.
```
String Buffer sb = new StringBuffer ("India");
System.out.println(sb);
sb.append ("digital");
System.out.println(sb);
```

- StringBuffer is a thread-safe or synchronized.
- StringBuffer is a slower
- StringBuffer introduced in java 1.0 version

2) String Builder
- String Builder is a class which is located in java. language package.
- String Builder does not follow string constant pool
- Hence it is considered as mutable class
- StringBuilder doesn't provide() concat() method to perform concatenation
- to perform concatenation, we can use append() method.

eg.
```
String Builder sb = new StringBuilder ("India");
System.out.println(sb);
sb.append ("digital")
System.out.println(sb);
```

- StringBuilder is a thread-unsafe or non-synchronized
- String Builder is faster than StringBuffer
- StringBuilder introduced in java 1.5 version.

① WAP to shift first character to last from string.

```
class MainClass
{
    public static void main (String args[])
    {
        String s1 = "India";
        String s2 = " ";
        for(int i=1; i<s1.length(); i++)
            s2 += s1.charAt(i);
            s2 += s1.charAt(0);
        System.out.println (s2);
    }
}
```

or

```
StringBuilder sb = new StringBuilder ();
for(int i=; i<s1.length(); i++)
    sb.append (s1.charAt (i));
sb. append (s1.charAt(0));
System.out.println (sb);
```

or

```
StringBuilder sb = new StringBuilder (s1.substring (1));
sb.append (s1.charAt (0));
System.out.println (sb);
```

or

```
StringBuilder sb = new StringBuilder (s1);
    sb.append (sb.charAt(0))
    sb.deleteCharAt (0);
    System.out.println (sb);
```

last character to first from string.

```
class MainClass
{
    public static void main (String args[])
    {
        String s1 = "India";
        StringBuilder sb = new StringBuilder(s);
        String s2 = " ";
    for (int i=0; i<s1.length(); i++)
    {
        s2 i := charAt(s1.length()-1);
        s2 += s1.charAt(s1.length()-1);
        System.out.println(s2);
    }
}
```

or

```
sb.append (s.charAt(s1.length()-1));
for(int i=0; i<s1.length()-1; i++)
    sb.append (s1.charAt(i));
s.o.p(sb);
```

or

```
StringBuilder sb = new StringBuilder (s1.charAt(s1.length()-1));
sb.append (s1.charAt (s1.length()-1));
sb.append (s1.substring(0, s1.length()-1));
System.out.println(sb);
```

or

```
StringBuilder sb = new StringBuilder (s1);
sb.insert (0, sb.charAt(sb.length()-1);
sb.deleteCharAt (sb.length()-1);
s.o.p(sb)
```

* StringBuffer & StringBuilder methods

① append()

② insert()

eg.
```
StringBuilder sb = new StringBuilder("test");
sb.insert(2, 123);
System.out.println(sb)
```

③ reverse()

eg.
```
StringBuilder sb = new StringBuilder("Hello");
sb.reverse();
System.out.println(sb);
```

④ replace()

eg.
```
StringBuilder sb = new StringBuilder("Hello world");
sb.replace(6, 11, "java");
System.out.println(sb);
```

⑤ capacity() // capacity of current object

eg.
```
StringBuilder sb = new StringBuilder();
System.out.println(sb.capacity())
```

⑥ substring

⑦ int capacity()

⑧ char charAt(int index)

⑨ void getChars(int srcBegin, src end, char[], dst, int dstBegin)

⑩ int indexOf(String str)
— (String str, int fromIndex)

⑪ lastIndexOf(String str)

⑫ length()

⑬ setCharAt(int index, char ch)

⑭

// WAP to swap first and last characters from string.
input : India
o/p : ndiI

```
class mainclass
{
    public static void main(String args[])
    {
        String s1 = "India";
        StringBuilder sb = new StringBuilder();
        sb.append(s1.charAt(s1.length()-1));
        sb.append(s1.substring(1, length()-1));
        sb.append(s1.charAt(0));
        System.out.println(sb);
    }
}
```

or

```
StringBuilder sb = new StringBuilder(s1);
sb.append(sb.charAt(0));
sb.deleteCharAt(0);
sb.insert(0, sb.length charAt(sb.length()-2));
sb.deleteCharAt(sb.length()-2)
```

* WAP to short each character from string.

i/p = India

o/p = Indin

```
class Mainclass
{
    public static void main (String args[])
    {
        String s1 = "India";
        char c[] = s1.toCharArray();
        java.util.Arrays.sort(c);
        System.out.println(c);
    }
}
```

* WAP to reverse string

i/p = India

o/p = aidn I

```
class Mainclass
{
    public static void main (String args[])
    {
        String s1 = "India";
        s1.reverse()
        StringBuilder sb = new StringBuilder(s1);
        sb.reverse();
        System.out.println(sb)
    }
}
```

or

```
StringBuilder sb = new StringBuilder()
for(int i = s1.length()-1; i>=0; i--)
    sb.append(s1.charAt(i));
System.out.println(sb);
```

* WAP to count vowels from string.

input : International

o/p : 6

```
class Mainclass
{
    public static void main (String args[])
    {
        String s1 = "International";
        int count = 0;
        for(int i=0; i<s1.length(); i++)
            if(s1.charAt(i) == 'A' || s1.charAt(i) == 'a' ||
                count++;
            System.out.println(count);
    }
}
```

or

```
if char c = s1.charAt(i);
    if(c == 'A' || c == 'a' || c == 'E' || c == 'e' ||
        count++;
```

or

```
String s2 = s1.toUpperCase();
int count = 0;
for(int i=0; i<s2.length(); i++)
{
```

```
char c = s2.charAt(i);
if (c=='A'||c=='E'||c=='I'||c=='O'||c=='U')
    count++;
s.o.p (count);
```

(or)

```
String s2 = "AEIOUaeiou";
int count=0;
for (int i=0; i<s1.length(); i++)
    if(s2.contains (String.valueOf(s1.charAt(i))));
        count++;
```

* WAP to replace at vowels from String.
i/p : Intenational
o/p : In+2rn3t45n6l

```
class program
{
  public static void main (String args[])
  {
    String s1 = "Intenational";
    String s2 = "AEIOUaeiou";
    int count=0;
    StringBuilder sb = new StringBuilder()
    for(int i=0; i<s1.length(); i++)
    if(s2.contains(String.valueOf(s1.charAt(i))));
        sb.append (++count);
    else
        sb.append (s1.charAt(i))
    System.out.println (sb);
  }
}
```

* i/p : Intenational
o/p : Gnt52nd1t32nd1

```
class mainclass
{
  public static void main (String args[])
  {
    String s1 = "Intenational";
    int count=0;
    String s2 = "AEIOUaeiou";
    StringBuilder sb = new StringBuilder (s1);
    for(int i=sb.length()-1; i>=0; i--)
      if(s2.contains (String.valueOf (sb.charAt(i))))
      {
        sb.deleteCharAt(i);
        sb.insert(i, ++count);
      }
    System.out.println (sb);
  }
}
```

Vowels at ~~first~~ last

```
class mainclass
{
  public static void main (String args[])
  {
    String s1 = "Intenational";
    String s2 = "AEIOUaeiou";
    StringBuilder sb = new StringBuilder();
    StringBuilder vow = new StringBuilder();
    for(int i=0; i<s1.length(); i++)
```

```
if(s2.contains(String.valueOf(s1.charAt(i))))
    vow.append(s1.charAt(i));
else
    sb.append(s1.charAt(i));
sb.append(vow);
System.out.println(sb)
}
}
```

* WAP to sort vowels from string

```
class Mainclass
{
    public static void main(String args[])
    {
        String s1 = "Intenabionee";
        String s2 = "AEIOUaeiou";
        StringBuilder sb = new StringBuilder();
        StringBuilder vow = new StringBuilder();
        for(int i=0; i<s1.length(); i++)
            if(s2.contains(String.valueOf(s1.charAt(i))))
                vow.append(s1.charAt(i));
        char c[] = vow.toString().toCharArray();
        java.util.Arrays.sort(c);
        System.out.println(sb);
        int count=0;
        for(int i=0; i<s1.length(); i++)
            if(s2.contains(String.valueOf(s1.charAt(i))))
                sb.setCharAt(i, c[count++]);
        System.out.println(sb);
    }
}
```

```
input : programmee
p
pr
pro
prog
progr
progra
program
programm
programme
programmee

class mainclass
{
    public static void main(String args[])
    {
        String s1 = "programmee";
        for(int i=0; i<s1.length(); i++)
            System.out.println(s1.substring(0, i+1));
    }
}
```

* WAP to count words from string.

```
class mainclass
{
    public static void main(String args[])
    {
        String s1 = "hello i am java developer";
        String s2 = s1.split(" ");
        int count=0;
        for(int i=0; i<s2.length; i++)
            if(s2[i].length()!=0)
                count++;
```

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

* WAP to append length & word at last & each words from string.

input: hello i am java developer

o/p: hello5 i1 am2 java4 developer9

```
class main
{
    public static void main(String args[])
    {
        String s1 = "hello i am java developer";
        StringBuilder sb = new StringBuilder(s1);
        int count = 0;
        for(int i=0; i<sb.length(); i++)
            if(sb.charAt(i) != ' ')
                count++;
            else
            {
                sb.insert(i, count);
                count = 0;
                i++
            }
        sb.append(count);
        System.out.println(sb);
    }
}
```

or

```
StringBuilder sb = new StringBuilder();
String s2[] = s1.split(" ");
for(int i=0; i<s2.length; i++)
{
    sb.append(s2[i]);
    sb.append(s2[i].length());
    if(i<s2.length-1)
        sb.append(" ");
}
System.out.println(sb);
```

* WAP to reverse each words from string

i/p: hello i am java developer

o/p: olleh i ma avaj repoleved

```
class mainclass
{
    public static void main(String args[])
    {
        String s1 = "hello i am java developer";
        StringBuilder sb = new StringBuilder();
        String s2[] = s1.split(" ");
        for(int i=0; i<s2.length; i++)
        {
            sb.reverse(s2[i]);
            if(i<s2.length-1)
                sb.append(" ");
        }
        System.out.println(sb);
    }
}
```

OR `sb.append(new StringBuilder(s2[i]).reverse());`

* i/p: hello I am java developee
o/p: i am java hello developee

```
class mainclass
{
  public static void main (String args[])
  {
    String s1 = "hello i am java developee";
    StringBuilder sb = new StringBuilder();
    String s2[] = s1.split(" ");
    for (int i=0; i<s2.length-1; i++)
      for(int j=i+1; j<s2.length; j++)
      {
        if (s2[i].length() > s2[j].length())
        {
          String temp = s2[i];
          s2[i] = s2[j];
          s2[j] = temp;
        }
      }
    for(int i=0; i<s2.length; i++)
    {
      sb.append(s2[i]);
      if(i<s2.length-1)
        sb.append(" ");
    }
    System.out.println(sb);
  }
}
```

i/p : hello i am java developee
o/p : hllmj v dv ipre Diaaaeeoc

```
class mainclass
{
  public static void main (String args[])
  {
    String s1 = "hello i am java developee";
    String s2 = "AEIOUaeiou";
    StringBuilder c = new StringBuilder();
    StringBuilder v = new StringBuilder();
    for(int i=0; i<s1.length(); i++)
      if(s2.contains(String.valueOf(s1.charAt(i))))
        v.append(s1.charAt(i));
      else
        if(s1.charAt(i)!=' ')
          c.append(s1.charAt(i));
    c.append(v);
    for(int i=0; i<s1.length(); i++)
      if(s1.charAt(i)==' ')
        c.insert(i,' ');
    System.out.println(c);
  }
}
```

PAGE NO
DATE

* WAP sort characters from each words of string
i/p : hello i am java developer
o/p : ehllo i am aajv deeelopmv

```
class mainclass
{
    public static void main (String args[])
    {
        String s1 = "hello i am java developer";
        String s2[] = s1.split(" ");
        StringBuilder sb = new StringBuilder();
        for(int i=0; i<s2.length; i++)
        {
            char c[] = s2[i].toCharArray();
            java.util.Arrays.sort(c);
            sb.append(c);
            if(i<s2.length-1)
                sb.append(" ");
        }
        System.out.println(sb)
    }
}
```

PAGE NO
DATE

WAP to capitalize each word from string
i/p : hello i am java developer
o/p : Hello I Am java Developer

```
class Mainclass
{
    public static void main(String args[])
    {
        String s1 = "hello i am java developer";
        StringBuilder sb = new StringBuilder();
        String s2[] = s1.split(" ");
        for(int i=0; i<s2.length; i++)
        {
            String temp = s2[i];
            sb.append(Character.toUppercase(temp.charAt(0))
            sb.append(temp.substring(1))
            if(i<s2.length-1)
                sb.append(" ");
        }
        System.out.println(sb);
    }
}
```

* WAP to uppercase all vowels from string
i/p : ~~~
o/p : hEllo I Am jAvA dEvElopEr

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

```
String s1 = "hello i am java developer";
StringBuilder sb = new StringBuilder(s1)
String s2 = "AEIOUaeiou";
for(int i=0; i<s1.length(); i++) {
  if(s2.contains(String.valueOf(c)))
  {
    sb.append(Character.toUpperCase(c));
  else
  {
      sb.append(cc)
  }
  System.out.println(sb)
}
```

* WAP to sum of all digits from string
input: welcome to 2025
O/P : 9

```
class MainClass
{
  public static void main(String args[])
  {
    StringBuilder sb = new StringBuilder()
    String s1 = "welcome to 2025";
    int sum = 0;
    for(int i=0; i<s1.length(); i++)
    {
      char ch = s1.charAt(i)
      if(Character.isDigit(ch))
      {
        sb.append(ch);
        sum += Character.getNumericValue(ch);
      }
    }
```

or

```
Integer.parseInt(String.valueOf(ch))
    System.out.println(sum);
  }
}
```

* WAP to remove special characters from string
i/p : h~

```
class ClassMain
{
  public static void main(String args[])
  {
    String s1 = "hello! where are you? i am here!!!";
    StringBuilder sb = new StringBuilder(s1)

    for(int i=0; i<s1.length(); i++)
    {
      char c = s1.charAt(i);
      if(!Character.isLetterOrDigit(c) &&
        !Character.isWhitespace(c))
      {
        sb.append(c);
        sb.deleteCharAt(i);
      }
    System.out.println(sb);
```

compare strings methods

== memory

equals() values case-sensitive

equalsIgnoreCase() value case-insensitive

compareTo() lexicographically case-sensitive

compareToIgnoreCase() ~~~ insensitive

intern()

:eg.

String S1 = new String("India").intern();

String S2 = ~~~

String S3 = "India";

S.O.p(S1==S3); // true

Inheritance

- In java inheritance is a concept, ulex it can reuse our code
- due to inheritance, we can remove code complexity
- in inheritance a class which provides elements to other class consider as a parent class.
- in inheritance, a class which acquires elements from other class considered as a child class.
- to perform inheritance, we can use extends keyword.
- inheritance also known as an IS-A relationship
- NOTE: To prevent inheritance, we can declare a class as a final.

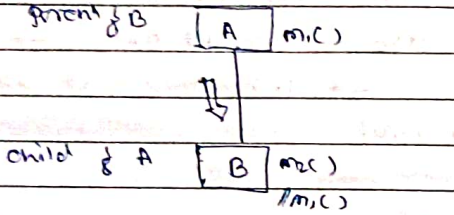
eg.

```
class calculator
{
  double add(double a, double b){return a+b;}
  double sub(double a, double b){return a-b;}
  double mul(double a, double b){return a*b;}
  double div(double a, double b){return a/b;}
}
class scificalculator extends calculator
{
  double square(int a){return a*a;}
  double sq.root(int a){return Math.sqrt(a);}
  double cube(int a){return a*a*a;}
  double cubroot(int a){return math.cbrt(a);}
}

class main class
{
    public static void main(String args[])
    {
        scificalculator sc = new scificalculator()
```

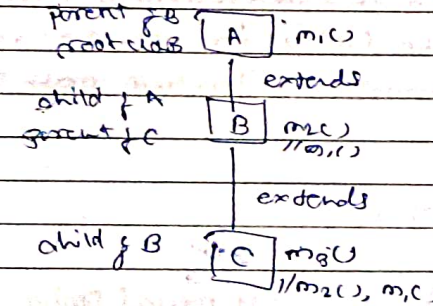
System.out.println(sc.add(10,20));
System.out.println(sc.square(10));
}

Types of Inheritance

① Single Inheritance
- In single inheritance, one parent class can have only one child class & vice-versa

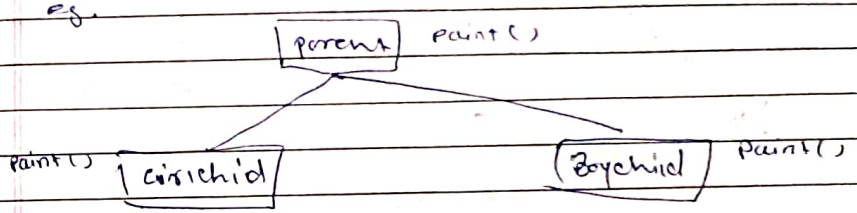


② Multi-level inheritance.
- in multi-level inheritance, one parent class can be child class of other class & vice-versa
- in multi-level inheritance, a class doesn't have parent class considered as a root class
- in multi-level inheritance, a class doesn't have child class considered as a child or leaf class.
- Note: parent class & root class considered as a Super class e.g Object class.

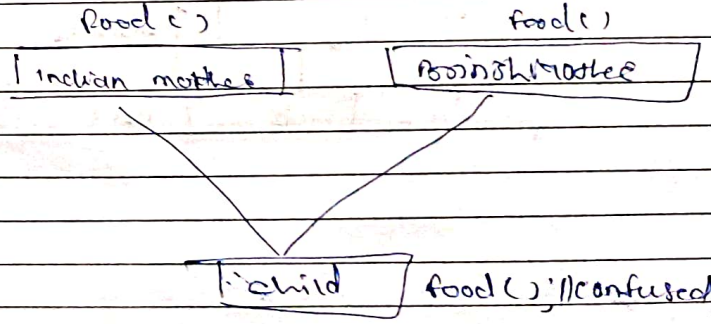


③ Hierarchical inheritance.
- In hierarchical inheritance one parent class can have two or more childclass
- we cannot invocked method of ~~subclass~~ subsidiary class.

e.g.

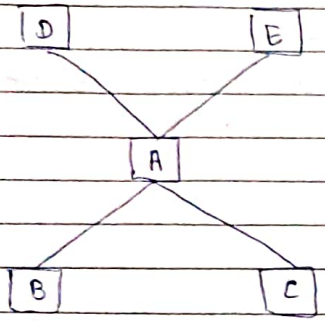


④ Multiple Inheritance
- In Java multiple inheritance, one child class can have two or more parent class
- Java allows to extend only one class at a time hence multiple inheritance is not supported.
- Java doesn't support multiple inheritance due to ambigious.



⑤ Hybrid inheritance.
- hybrid inheritance is a comb^{n} of multiple & multi-level inheritance.
- hybrid inheritance is not supported due to multiple inheritance.

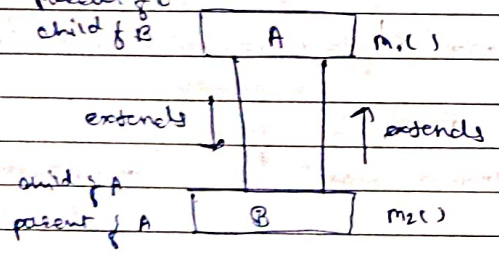
e.g



⑥ cyclic inheritance

- in cyclic inheritance one class can be child and parent of same class
- cyclic inheritance is not supported due to method ambigious.

e.g.



* method overriding.

- in java method overriding is a concept, where one class contains 2 or more method with the same name & same parameter
- we cannot create two or more method with same name & same parameter in one class
- To perform method overriding, we requires atleast 2 class in Is-A relationship

eg.

```
class Demo
{
    void m1()
    {
        System.out.println("m1 method of Demo1");
    } // overriden method
}

class Demo2 extends Demo1
{
    void m1()
    {
        System.out.println("m1 method of Demo2");
    } // override method
}
```

- method overriding is a concept where we can execute method as per object.

e.g.

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

```
Demo1 d1 = new Demo1();
d1.m1(); // method of Demo1

Demo1 d2 = new Demo2();
d2.m1(); // method of Demo2
}
}
```

- in method overriding method can be execute differently on the basis of object, hence it is considered as a runtime Polymorphism.
- method overriding is a concept, where we can execute method on the basis of object
- method overriding is useful for developer
- to prevent method overriding, we can declare method as a final.
- we cannot assign weaker access specifier to override method.
- we cannot override static method.

* Method overloading vs overriding

1) method overloading is a concept where one class contains 2 or more method with same name but diff parameter	1) method overriding is a concept where one class contains 2 or more method with same name & same parameter
2) we can overload method in single class	2) to perform method overriding we requires atleast 2 class in IS-A relationship
3) method overloading is a concept, where we can execute methods as per their parameter	3) method overriding is a concept where we can execute method as per object
4) method overloading can be considered as a compile-time polymorphism	4) method overriding can be considered as a runtime polymorphism
5) method overloading is useful for end-user	5) method overriding is useful for developer
6) we can not prevent method overloading	6) to prevent method overriding we can use final keyword
7) we can overload static method	7) we can not override static method.

* Upcasting vs downcasting.

1) upcasting
- In java upcasting is a concept, where object of child class reference to parent class
- upcasting also known as a widening or generalization
- during upcasting process, method invoked as per object & properties will be considered of reference.

eg.
```
Demo d1 = new Demo2();
Demo d2 = new Demo3();
Demo d3 = new Demo3();
```

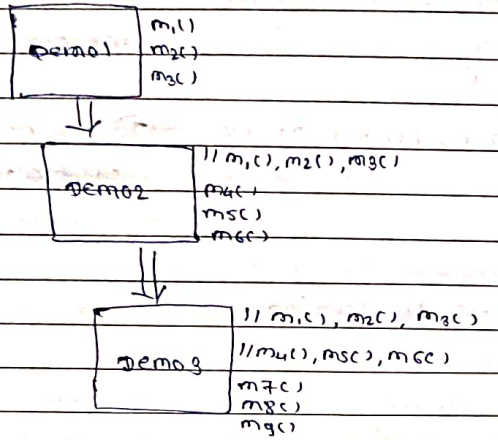
2) downcasting
- In java downcasting is a concept, where object of parent class reference to child class
- downcasting also known as a narrowing or degeneration
- downcasting is not permissible.

eg.

```
Demo2 d1 = new Demo1();
Demo3 d2 = new Demo1();
Demo3 d3 = new Demo2();
```

during upcasting process, only common & closest method will be invoked

eg.




```
class Demo1
{
    void m1(){System.out.println("m1 method of Demo1");}
    void m2()  ——  (
    void m3()  ——  (m3
}

class Demo2 extends Demo1
{
    void m4(){ ——  ("m4 method of Demo2");}
    void m5()  ——  ("m5 ——
    void m6()  ——  ("m6 ——
}

class Demo3 extends Demo2
{
    void m7(){System.out.println("m7 method of Demo3");}
    void m8(){ ——  ("m8 ——
    void m9(){ ——  ("m9 ——
}

class MainClass
{
    public static void main(String args[])
    {
        Demo1 d1 = new Demo3();
        d1.m1();
        d1.m2();
        d1.m3();
    }
}
```

this vs super

1) this

- this keyword can be used to represent current class object
- this keyword can be used with object, variable & constructor
- this keyword with variable can be used to diff bet local vs non-local variable.

eg.

```
this a;
```

- this keyword can be used to represent invoked user-defined constructor from default & vice-versa

eg

```
this();
```

NOTE:
This keyword with constructor should be first statement in constructor.

e.g.

```
class Demo3
{
    int a;
    void display()
    {
      int a = 20;
    System.out.println("a :" +this.a);
    s.o.p(this);
    }
  Demo3()
    {
        this(100);
        s.o.p ("default const called");
    }
  Demo3(int a)
    {
      this.a=a;
      s.o.p("para const called");
    }
}

class mainclass
{
    public static void main (String args[])
    {
        Demo3 d1 = new Demo3();
        s.o.p(d1.a);
    }
}
```

4 Super

- in java super is a keyword, which can be used to denote parent class.
- Super keyword, we can used to with variable, method & constructor
- by using super keyword, we can denotes parent class variable

e.g.

```
super.a;
```

- by using super keyword, we can denotes parent class method

e.g.

```
super.display();
```

- by using super keyword, we can invoked parent class constructor

e.g.

```
super();
```

Note: super keyword with constructor, should be at first statement

super keyword gives a compile-time error, if mentioned element not found.

e.g.

```
class Demo1
{
    int a = 10;
    void display()
    {
    System.out.println("display method of demo1");
    }
```

```
Demo1()
{
  System.out.println("default const of Demo1");
}
Demo1(int a)
{
  System.out.println("para const of Demo1");
}
}

class Demo2 extends Demo1
{
  int a = 25;
  void display()
  {
    System.out.println("display method of Demo2");
  }
  Demo2()
  {
    System.out.println("default const of Demo2");
  }
  Demo2(int a)
  {
    System.out.println("para const of Demo2");
  }
}

class Demo3 extends Demo2
{
  int a = 30;
  void display()
  {
    System.out.println("display method of Demo3");
    System.out.println("a = "+super.a);
    super.display();
  }
  Demo3()
  {
    super(10);
    System.out.println("default const of Demo3");
  }
  Demo3(int a)
  {
    System.out.println("para const of Demo3");
  }
}

class MainClass
{
  public static void main(String args[])
  {
    Demo d1 = new Demo3();
    d1.display();
  }
}
```

* this vs super

this	super
① this keyword can be used to represent current class of it's elements	① super keyword can be used to represent parent class of its elements

this keyword	super keyword
① this keyword can be used with variable, object and constructor	① super keyword can be used with variable, method & constructor
② By using this keyword with variable, we can diff betn local vs non-local variable	② by using super keyword with variable we can diff betn child & parent variable
③ this keyword can not be used with method	③ by using super keyword we can invoked parent class method from child class
④ this keyword can be used to represent current object	④ super keyword can not used with object
⑤ by using this keyword we can invoked default constructor from para constructor & vice-verso	⑤ by using super keyword we can invoked parent class constructor from child class
⑥ this keyword doesn't give compile-time error	⑥ super keyword gives compile-time error if mentioned element not found.

* package
- in java package is a folder or sub-directory which can be used to store classes & interfaces
- by using package we can organised our appln
- if appln is well organised, we can easily access required classes & interfaces
- to create a package, we can use package keyword.
- package name should be small cased

eg.

```
Package a.b.c.d.e;
public class Demo
{
    int a;
public void display()
    {
        System.out.println('Display method of demo');
    }
}
// javac -d . Demo.java

class MainClass
{
    Public static void main (String args[])
    {
        a.b.c.d.e.Demo d1 = new a.b.c.e.Demo();
    }
}
```

- to use other package classes, we can use qualified name.

a.b.c.d.e.Demo

- to use other package classes frequently we can use import keyword also

import a.b.c.d.e.Demo;

* Access specifier

- in java access specifier is a concept, where we can set scope of visibility to class or it's elements
- Types of access specifier.

1) private

- to use private access specifier can be used with method & variable
- scope of private access specifier upto within a class

2) Default

- to use default access specifier, there is no keyword
- default access specifier can be used with class method & variable
- scope of default access specifier also known as a package access specifier

3) protected

- to use protected access specifier to use protected keyword
- protected access specifier can be used methods & variable only
- scope of protected access specifier upto within a package or throughout the package or outside the package only through child class

4) public

- to use public access specifier to use public keyword
- public access specifier can be used with class method & variable
- scope of public access specifier throughout the project.

* Encapsulation

- encapsulation is a concept, where we can hide our data
- to hide our data we can declare our properties as private
- to initialize private data we can use setter method.
- to represent private properties / data we can use getter method.
- by using setter & getter, we can set or get individual properties of class.
- to initialize all private variable directly we can use constructor
- to represent all private properties directly we can override toString() methods.
- By using encapsulation, we can send data from one end to other
- encapsulation also known as data binding or data hiding process
- encapsulation also known as a wrapping of class
- encapsulation also known as a concrete class
- encapsulation also known as a model or entity class.

eg.

```
class employee
{
    private int id;
    String Name;
    String designation;
    String company;
    double salary;

// setter method.
public void setId (int id) { this.id = id; }
    setName (String Name) { this.Name = Name; }

// getter method
public int getId() { return id; }
    String getName() { return Name; }

Employee() { } //default constructor
Employee (int id, String name, String designation, String company
double salary)
    {
        this.id = id;
        this.name = name;
        this.designation = designation;
        this.company = company;
        this.salary = salary;
    }
public String toString()
    {
        return "Employee [id =" + id + ", name =" + name + ", designation
=" + designation + ", company =" + company + ", salary =" + salary + "]";
    }
}

class mainclass
{
    public static void main (String args[])
    {
        Employee e = new Employee();
        e.setId (10);
        e.setName ("raghav Chatma");
        e.setDesignation ("Java developer");
        e.setCompany ("ibm");
        e.setSalary (40000);
        System.out.println ("ID:" + e.getID());
            ("Name:" + e.getName());
            ("designation:" + e.getDesignation());
            ("company:" + e.getCompany());
            ("Salary:" + e.getSalary());

        Employee e1 = new Employee (20, "umang patil", "c++",
"angular developer", 12000);
        System.out.println (e1);
    }
}
```

PAGE NO
DATE

* Abstraction

- in java abstraction is a concept, where we can hide our method.
- by using abstraction we can hide our logic or implementation
- to achieve abstraction, we can use abstract class.

what is abstract class?

- In java abstract class contains abstract method.
- abstract method does not contain implementation
- abstract method also considers as a ~~imp~~ incomplete method
- if class contain any method as abstract class becomes abstract implicitely
- here abstract keyword denotes incomplete things
- we cannot create an object of abstract class

eg.

```
abstract class Demo
{
void m1() // method signature
{
   System.out.println("m1 method of demo");
}
  // non-abstract
   abstract void m2(); //abstract method
}

class mainclass
{
   public static void main (String args[])
   {
        Demo d2 = new Demo(); // not possible
   }
}
```

PAGE NO
DATE

- abstract class also consider as 0 to ∞ implemented class
- to prevent object creation we can declare a class as an abstract
- to use abstract class we can use extends keyword
- we have to override abstract method into child class of child class becomes abstract implicitely.

eg.

```
class Demo2 extends Demo
{
    void m2()
    {
      System.out.println("m2 method of Demo2");
    }
}

class mainclass
{
  public static void main (String args[])
  {
      Demo2 d1 = new Demo2();
      d1.m1();
      d1.m2();
  }
}
```

NOTE: we can not use abstract keyword with static or final

abstract class can contain constructor too because of 0 to 100 implemented class.

eg.

```
abstract class Demo
{
  Demo()
  {
    System.out.println("default const of Demo");
  }
  Demo(int a)
  {
    System.out.println("para cost of Demo");
  }
}

class Demo extends Demo
{
  Demo2()
  {
    Super(10);
  }
}

// Main class is same as before
```

- abstract class can be used to hide implementation
- abstract class can be used to allocate abstraction

eg.

```
package a.b.c.d.e;
public abstract class RBI
{
    public final double r = 4.5;
    protected final double countSI(double p, int n)
    {
        return (p*r*n)/100;
    }
}

final class SBI extends a.b.c.d.e.RBI
{
  public double countSimpleInterest(double p, int n)
  {
    return countSI(p,n);
  }
}

class mainclass
{
  public static void main(String args[])
  {
    SBI s = new SBI();
    System.out.println(s.countSimpleInterest(10000,4));
    System.out.println(s.r);
  }
}
```

OR

```
abstract class RAM
{
  void performance()
  {
    System.out.println("32GB ready to work");
  }
  abstract void assembled();
}
```

```
class mobile extends RAM
{
  void assembled()
  {
    System.out.println("RAM assembled in mobile successfully");
  }
}

class mainclass
{
  public static void main (String args[])
  {
    Laptop l = new laptop();
    l.performance();
    l.assembled();
    Mobile m = new mobile();
    m.performance()
    m.assembled();
  }
}

Class Laptop extends RAM
{
  void assembled()
  {
    System.out.println("Ram is assembled to laptop");
  }
}
```

Interface

- to create an interface, we can use interface keyword.
- interface name should be capitalize or unique.
- interface contains all methods are abstract by default (public)
- hence abstract keyword doesn't required to use
- interface considered as a 0 implemented class
- interface can not have constructor or initialization because it is 0 implemented class.
- we can not create object of interface.
- interface variables are public, static & final by default.
- to use interface we can use implements keyword
- we have to override all abstract method into child class or child class becomes abstract implicitely

eg:

```
interface Demo1
{
  int a = 10;
  void m1();
}

class Democlass Implements Demo1
{
  public void m1()
  {
    System.out.println("m1 method of Demo class");
  }
}

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

```
        Democlass d = new Democlass();
        d.m1();
    }
}
```

NOTE:
an interface can extends other interface too.

eg.

```
interface Demo1
{
    void m1()
}

interface Demo2 extends Demo1
{
    void m2()
}

class Democlass implements Demo2
{
    public void m1()
    {
    S.o.p("m1 method of Democlass");
    }
    public void m2()
    {
    S.o.p("m2 method of Democlass");
    }
}
```

```
class Mainclass
{
    public static void main (String args[])
    {
        Democlass d = new Democlass()
            d.m1();
            d.m2();
    }
}
```

-NOTE
one class can implements multiple interfaces too.

eg.

```
interface Demo1
{
    void m1();
    void m3();
}

interface Demo2
{
    void m2();
    void m3();
}

class Democlass implements Demo1, Demo2
{
    public void m1()
    {
        "
    }
    public void m2()
    {
        "
    }
```

```
public void m3()
{
    //
}
}

class mainclass
{
    public static void main(String args[])
    {
        Democlass d = new Democlass();
        d.m1();
        d.m2();
        d.m3();
    }
}
```

NOTE

onwards java 1.8 version, we can create default method.

eg.

```
interface Demo1
{
    default void m1()
    {
        //
    }
    default void m3()
    {
        //
    }
}
```

```
interface Demo2
{
    default void m2()
    {
        //
    }
    default void m3()
    {
        //
    }
}
```

(class implements)

```
interface Democlass extends Demo1, Demo2
{
    ~~default~~ public void m3()
    {
        Demo1.super.m3();
        Dem2.super.m3();
    }
}

class mainclass
{
    public static void main(String args[])
    {
        Democlass d = new Democlass();
        d.m1();
        d.m3();
    }
}
```

- NOTE

onwards java 1.8 version we can create static method also

e.g.

```
interface Demo1
{
    static void m1()
    {
        //
    }
    static void m3()
    {
        //
    }
}

interface Demo2
{
    static void m2()
    {
        //
    }
    static void m3()
    {
        //
    }
}

class Democlass implements Demo1, Demo2
{
    public void m1()
    {
        Demo1.m1()
    }
}
```

```
class mainclass
{
    public static void main(String args[])
    {
        Demo1.m1();
        Demo2.m2();
        Demo1.m3();
        Demo2.m3();
        Democlass d = new Democlass();
        d.m1();
    }
}
```

* Usage of Interface

① By using interface, we can acheive multiple inheritance

e

eg.

```
Interface IndianMotter
{
    default void food()
    {
        S.O.p("chapati ready");
    }
}

Interface BritishMotter
{
    default void food()
    {
        S.O.p("maggi ready");
    }
}
```

```
class son implements IndianMother, BritishMother
{
    public void food()
    {
        IndianMother.super.food();
        BritishMother.super.food();
    }
}

class mainclass
{
    public static void main(String args[])
    {
        son s = new son();
        s.food();
    }
}
```

② interface can be used as a blueprint or skeleton for appln

```
eg.
interface car
{
    int maxspeed = 120;
    void color();
    void type();
    void cc();
}

class Honda implements car
{
    int minspeed = 40;
    public void color()
    {
        S.o.p("Honda available in all colors");
    }
    public void type()
    {
        S.o.p("Hondacar available in sedan suv");
    }
    public void cc()
    {
        S.o.p("Hondacar available in 1000cc to 2000cc");
        S.o.p("minspeed=" + minspeed + " maxspeed=" + maxspeed);
    }
}

class BMW implements car
{
    ---
}

class ferrari implements car
{
    ---
}

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

```
Car c = new Hondacar();
c.color();
c.type();
c.cc();
}
}
```

3) By using interface we can make appln loosely couple

eg.

```
interface Sim
{
    void startSim()
}

class vodafone implements Sim
{
    public void startSim()
    {
        System.out.println("welcome to vodafone network");
    }
}

class Idea implements Sim
{
    ---
}

class Airtel implements Sim
{
    ---
}
```

```
class RelianceMobile
{
    private Sim r = new Ideasim();
    RelianceMobile()
    {
        startMobile();
        r.startSim();
    }
    private void startMobile()
    {
        S.o.p("welcome to reliance mobile");
    }
}

class Mainclass
{
    public static void main(String args[])
    {
        new RelianceMobile();
    }
}
```

* Helper class

- Helper class contains methods that help in assessing the program. This class intends to give quicke implementation & basics methods such that programmers do not have implement again & again.

eg.

```
interface Demo1
{
    void m1()
    void m2()
    void m3()
    void m4()
    void m5()
}
```

```
class Helperclass implements Demo
{
    public void m1() { }
    -- m2() { }
    m3() { }
}

class Democlass extends Helperclass implements Demo
{
    public void m1()
    { S.O.P ("m1 method of Democlass") }
}

class Mainclass
{
    public static void main (String args[])
    {
        Democlass d = new Democlass();
        d.m1();
    }
}
```

interface	abstract class
① to create an interface, we can use interface keyword	① to create abstract class, we can use abstract keyword
② interface contains all methods abstract by default	② abstract class contains abstract & non-abstract methods
③ interface considered as 0 implemented class	③ abstract class considered as a 0 to 100 implemented class
④ interface cannot contain constructor or initializers	④ abstract class contains constructor or initializers
⑤ interface contains all variables are public final & static by default	⑤ we can create public final & static variable into abstract class
⑥ interface contains all elements are public by default	⑥ we can access specifier as per requirement in interface
⑦ we can not use abstract class in interface	⑦ we can use interface in abstract class
⑧ to use interface, we can use implements keyword	⑧ to use abstract class we can use extends keyword
⑨ we can create default method in interface	⑨ we cannot create default method in abstract class
⑩ interface can be used - to achieve multiple inheritance - as blueprint of appn - to achieve loosely coupled appn - to achieve powerful abstracn	⑩ abstract class can be used to achieve abstracn

Polymorphism

- in java polymorphism is a concept where we can execute method differently
- by using polymorphism, we can execute method as per requirements

Types of polymorphism

① compile-time polymorphism
- compile-time polymorphism also known as a static or early binding
- compile-time polymorphism can be achieved by using method overloading
- in method overloading, we can execute method differently on the basis of parameter
- compile time polymorphism is useful for end-user
- we cannot prevent, compile-time polymorphism

② runtime polymorphism
- runtime polymorphism also known as a late or dynamic binding.
- runtime polymorphism can be achieve by using method overriding

- in method overriding, we can execute method differently on the basis of object
- runtime polymorphism is useful for developer
- to prevent runtime polymorphism, we can use final keyword

Exception Handling.

- In java exception handling is a concept, where we can maintain flow of execution

What is exception

- in java exception is an unwanted interruption betn execut^n & appl^n
- Due to exception execution of app^n can be disturb

Types of Exception

① checked Exception

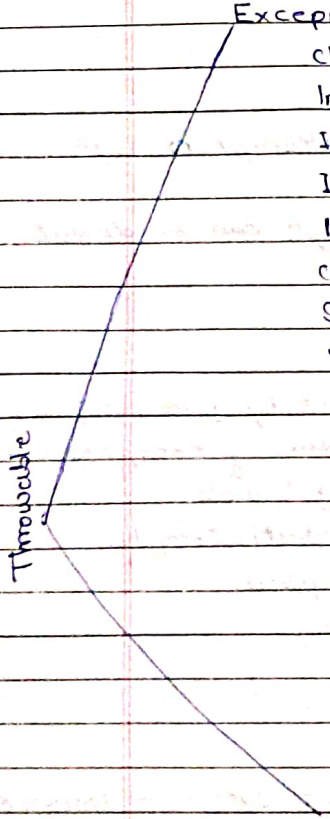
- checked exception can be detect by java compiler
- checked exception is a child class of exception class
- checked exception is mandetory handle.

eg. ClassNotFoundException, InstantiationException
IllegalAccessException, IOException
InterruptException, SQLException
CloneNotFoundException

② Unchecked Exception

- Unchecked exception can be detect JVM not compiler
- unchecked exception is a child of RuntimeException class
- Unchecked exception is optional to handled

eg. NullPointerException
ArithmeticException
NumberFormatException
IllegalThreadStateException
ArrayIndexoutofBoundException
StringIndexoutofBoundException



Examples of Exception

① ClassNotFoundException
- ClassNotFoundException located in java.language pkg.
- ~~it~~ is a child class of Exception hence it is considered as checked exception
- ClassNotFoundException occured if we try to load a class file, and mentioned class file not found.

② InstantiationException
- InstantiationException is a class which is located in java.lang pkg.
- ~~it~~ is a child class of exception class, hence it is considered as checked excepth
- ~~it~~ occured if we try to create instance and it wont happened.

③ IllegalAccessException
- IllegalAccessException is a class, located in java.lang.pkg,
- ~~it~~ is a child class of exception class hence it is considered as checked exception
- ~~it~~ occured if we try to access class or its elements & things are not accessible.

Eg:-

```
Package a.b.c.d.e;
public abstract class demo
{
    public int a;
    public void display()
    {
```

```
        s.o.p ("display method of Demo");
    }
}
class Mainclass
{
    public static void main (String args[])
    {
        s.o.p("program started");
        try {
            class c = class.forName ("a.b.c.e.Demo");
        a.b.c.d.e.Demo d1 = (a.b.c.d.e.Demo) c.newInstance();
            d1.a = 100;
            d1.display ();
        } catch (ClassNotFoundException | InstantiationException
|IllegalAccessException e)
        {
            s.o.p (e);
        }
        s.o.p("program ended");
    }
}
```

④ IOException
- IOException class which is located in java.io.pkg.
- — is a child of Exception class, hence it is considered as a checked exception
- — occurred if we try to perform I/O operatn & there is an interruption.

eg.

```
import java.io.BufferedReader;
  —                InputStreamReader;
  —                IOException;
class Mainclass
{
    public static void main (String args[])
    {
        s.o.p ("program started");
        InputStreamReader isr = new InputStreamReader (system.in
        BufferedReader br = new BufferedReader (isr)
        try {
            br.readLine ();
        } catch (IOException e)
        {
            System.out.println (e);
        }
        s.o.p ("program started");
    }
}
```

⑤ CloneNotSupportedException
- CloneNotSupportedExceptn class which is located in java.lang.pkg.
- — is a child class of Exception class
- — occurred if we try to perform clonning process & wont happen, due to class doesn't implement cloneable interface.

eg.

```
package a.b.c.d.e;
public class demo implements cloneable
{
    public int a;
```

```
public void display()
{
    S.O.p ("display method of demo");
}
public demo clone()
{
    demo d1 = null;
    try {
    d1 = (demo) super.clone();
    }
    catch (CloneNotSupportedException e) {
        System.out.println(e);
    }
    return d1;
}
}

import a.b.c.d.e.Demo;
class mainclass
{
    public static void main (String args[])
    {
    System.out.println ("program started...");
    Demo d1 = new Demo();
    d1.a = 10;
    Demo d2 = d1.clone();
    System.out.println(d1.a);
            (d2.a);
    S.O.p("program ~~started~~ ended");
    }
}
```

6) InterruptedException
- InterruptedException is a class which is located in java.lang.pkg.
- InterruptedException is a child class of exception class.
- InterruptedException occured if we try to interrupt any thread & it won't happen

7) SQLException
- SQLException is a class, which is located in java.sql pkg.
- SQLException is a child class of Exception class, hence it is considered as a checked exception
- SQLException occured, if any interruption in SQL operation

unchecked

8) ArithmeticException
- ArithmeticException is a class which is located in java.lang.pkg.
- ~~it~~ is a child of RuntimeException class hence it is called as unchecked exception
- it occurred if we try to divide any number with integer zero.

e.g.
```
class mainclass
{
    public static void main (String args[])
    {
    S.O.p ("program started...");
    int a = 10, b = 0, c = 0;
    try {
    c = a/b;
    } catch (ArithmeticException e) {
        S.O.p("e = " + e);
```

```
    }
    S.O.p("c = "+c);
    S.O.p("program ended...");
    }
}
```

9) NullpointerException
- it is a class which is located in java.lang.pkg.
- it is a child class of RuntimeException hence it is unchecked exception
- it occured, if try to access non static element of a class by using object & object is null

e.g
```
class mainclass
{
  public static void main (String args[])
  {
    S.O.p ("program started...");
    String s1 = null;
    try {
    System.out.println (s1.length());
    } catch (NullPointerException e)
      {
        S.O.p (e);
      }
    S.O.p("program ended...");
  }
}
```

10) NumberFormatException
- it is a class which is located in java.lang.pkg.
- it is a child class of runtimeexcepth hence it is called as unchecked excepth
- it occured if, we try to convert any string into number & it wont happened.

e.g
```
class mainclass
{
  public static void main (String args[])
  {
    S.O.p ("program started");
    String s1 = "4A";
    int num = 0;
    try {
    num = Integer.parseInt (s1);
    } catch (NumberFormatException e)
    {
      S.O.p(e);
    }
    S.O.p("num = " +num);
    S.O.p("program ended");
  }
}
```

~~11) ArrayIndex out of B~~

11) ArrayIndexoutOfBoundException
- it is a class which is located in java.lang.pkg
- it is a child class of runtimeException hence it is called as unchecked excepth
- it occured if we try to access index of an array which is not present

```
class Mainclass
{
  public static void main (String args[])
  {
     S.o.p ("program started...");
     int a[] = {10, 20, 30};
     try {
      System.out.println (a[5]);
     } catch (ArrayIndexOutOfBoundException e)
      {
        S.o.p(e);
      }
     S.o.p("program ended...");
    }
}
```

12) StringIndexOutOfBoundException

- It is a class which is located in java.lang.pkg
- It is a child class of RuntimeException, hence it is considered as an unchecked exception
- It occured if we try to access index of string which is not present

eg.

```
class mainclass
{
  public static void main (String args[])
  {
     S.o.p ("program started ...");
     String s1 = "India";
     try {
      S.o.p (s1.charAt[5]);
     }
     catch (StringIndexOutOfBoundsException e)
     {
        S.o.p (e);
     }
     S.o.p("program ended...");
  }
}
```

13) NegativeArraySizeException

- It is a class which is located in java.lang.pkg
- It is a child class of RuntimeException class hence it is called as unchecked exception
- It occured if we create any array in -ve size.

eg.

```
class Mainclass
{
  public static void main (String args[])
  {
     S.o.p ("program started.");
     try {
      int a[] = new int[-1];
     }
     catch (NegativeArraySizeException e)
     {
        S.o.p(e)
     }
     S.o.p ("program ended");
  }
}
```

14) ClassCastException
- it which is located in java.lang.pkg
- it is a child class of RuntimeException hence it is called as unchecked exception
- it occured if we try to perform downcasting explicitely

15) IllegalThreadStateException
- it it
- it it
- it occured if we try to access state of thread which is not possible.

Types of blocks in exception handling.
① try block
- to create try block, we can use try keyword.
- try block associated with catch Finally resource
- try block use to execute doubtful statement
- Syntax
```
try{
  //doubtful statement;
}catch(exception e){
      //Statement;
}
```

2) Catch block
- to create catch block, we can use catch keyword
- catch block associated with try
- catch block invoked implicitely by matching of suppose occured exception
- catch block can be used to represent occured exception or to provide user-defined msg or alternatives

e.g.
```
String s1 = null;
try{
 s.o.p(s1.length());
}catch (RuntimeException e) {
     s.o.p(e);
}
```

- NOTE: we can use handle multiple catch block in single try as well as to provide appropriate soln as per occured soln
- in this approach super catch always called at last

eg.
```
String s1[] = {"123", "23 ", "444", "ABC"};
int num = 0;
try{
 num = Integer.parseInt(s1[3].substring(4));
}catch (ArrayIndexOutOfBoundsException e)
 {
    s.o.p("catch 1 called")
    s.o.p(e)
 }
 catch (NullPointerException e)
 {
    ---
 }
 catch (NumberFormatException e)
 {
    ---
 }
 catch (Exception e)
 {
    ---
 }
```

NOTE: We can handle multiple exception by using single catch to provide common solution to all
- In this approach we cannot use parent exception with child.

e.g.

```
String s1[] = {"123", "231", "421", "123"}
try{
int num = Integer.parseInt(s1[2])
} catch (all exception differ by 1){
.
.
}
```

same example as exception handling 1st program

3) finally block
- to create finally block, we can create finally keyword.
- finally block associated with try & catch block.
- finally block invoked implicitely no matter exception occurred or handled
- finally block can be used to ~~handle~~ execute important like closing connect^ etc.

```
eg. S.o.p("program started");
String s1 = "India"; or null;
try{
System.out.println(s1.length())
} catch (NullPointerException e){
s.o.p(e)
} finally{
System.out.println("finally blocked");
}
System.out.println("program ended");
}
```

what is finalize?
- In java finalize method of object class, use by garbage collector implicitely to destroy unreferenced object from app^n
- by using finalize() method, we can close connection.

eg.

```
class mainclass
{
public static void main(String args[])
{
s.o.p("program started");
new mainclass();
system.gc();
s.o.p("program ended");
}
public void finalize()
{
s.o.p("finalized method called");
}
}
```

Final vs finaly vs finalize

① final
- in java final is a keyword, which can be used with class, method & variable.
- if we declare a class as a final, class will not extends anymore
- if we declare a method as a final then we can not override that method.
- if we declare a variable as a final, variable becomes immutable.

2) Finally

- in java finally is a block, which can be used in exception handelling.
- Finally blocked invocked implicitely no matter exception occured or handled.
- Finally block can be used to close connecth.

3) Finalize

- In java finalize is a method of object class
- Finalize() method used by garbage collector implicitly to destroy unrefferenced objects from appln

Throw vs throws

* throw

- In java throw keyword can be used to throw an exception.
- throw keyword can be used to throw unchecked exception only
- by using throw keyword, we can throw single exception at a time.
- by using throw keyword, we stopped execution of appn explicitely
- throw keyword is usefull for end-user

eg.

```
class Site
{
    public void login(int age)
    {
        System.out.println("welocome to our site");
        if (age >= 70)
            home();
        else
        {
            throw new ArithmeticException();
        }
        System.out.println("thank you for visit");
    }
    private void home()
    {
        System.out.println("welcome to 2nd home");
    }
}

class mainclass
{
    public static void main(String args[])
    {
        System.out.println("program started")
        Site s = new Site()
        try{
            s.login(30);
        }catch(ArithmeticException e)
        {
            System.out.println(e);
        }
        System.out.println("program started...");
    }
}
```

PAGE NO
DATE

* throws
- In java throws keyword can be used to declare checked exception only
- by using throws keyword, we can declare multiple exception as per requirement
- by using throws keyword, we can execute unreported method
- in java unreported method means, we have to handle mentioned exception to invocked method method.
- throws keyword is useful for developer

Q.
```
class Site
{
    public void login(int age) throws
ClassNotFoundException, InstantiationException, InterruptException
    {
        System.out.println("welcome to our site");
        if(age >= 70)
            home();
        else
        {
            throw new ArithmeticException();
        }
        System.out.println("thank you for visit");
    }
    private void home()
    {
        System.out.println("welcome to your 2nd home");
    }
}
```

PAGE NO
DATE

```
class mainclass
{
    public static void main(String args[])
    {
        System.out.println("program started");
        Site s = new Site();
        try {
            s.login(30);
        } catch(ArithmeticException | ClassNotFoundEx |
InstantiationException | InterruptException e) {
            System.out.println(e);
        }
        System.out.println("program ended");
    }
}
```

throw vs throws

throw	throws
① throw keyword can be used to declare an exception	① throws
② can be used to unchecked exception only	② used with checked exception only
③ by using throw we can throw single exception at a time	③ by using throws, we can declare multiple exception as per requirement
④ throw keyword can be used to stopped execution of appn explicitely	④ throws keyword can be used to create unreported method

⑤ throw is useful for end-user

⑥ throw can be used within a method

⑤ throws is useful for developer

⑥ throws can be used with method

User defined Exception (custome exception)

- user-defined Exception also known as custom exception
- by using user-defined exception we can provide the defined msg.
- by using user-defined msg, we can make application user friendly
- to create user-defined exception, we can extends RuntimeException class

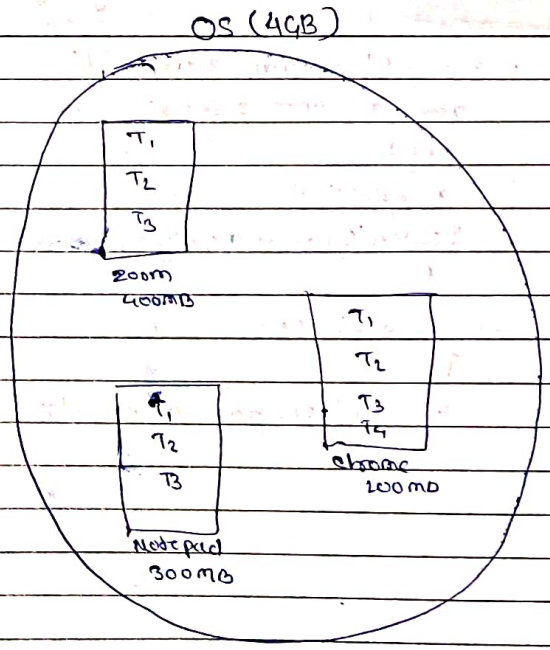
eg.

```
class InvalidAgeException extends RuntimeException
{
    private String msg = "Invalid age !!";
    InvalidAgeException (String msg) { this.msg = msg; }
    InvalidAgeException () { }
    public String toString ()
    {
        return "InvalidException: "+msg;
    }
}

class Bike
{
    public void login (int age)
    {
        S.o.p ("welcome to our bike");
        if (age >= 20)
        {
            Home ();
        }
        else
        {
            throw new InvalidAgeException ("age should be
                                            atleast 20 or above");
            S.o.p ("thanks for visit");
        }
    private void home ()
    {
        System.out.println ("welcome to your 2nd home");
    }
}

class MainClass
{
    public static void main (String args[])
    {
        S.o.p ("program started");
        Bike b = new Bike();
        try {
        b.login (30);
        } catch (InvalidAgeException e)
        {
            S.o.p (e)
        }
        S.o.p ("program ended..");
    }
}
```

Multithreading

- In java multi-threading is a concept, where we can execute 2 or more app^n simultaneously
- What is thread.
  - In java thread is a small app^n which can be consider as a subtask or sub-process
  - thread is a lightweight app^n
  - thread is a shared-resource app^n
  - switching bet^n 2 thread is cheapest

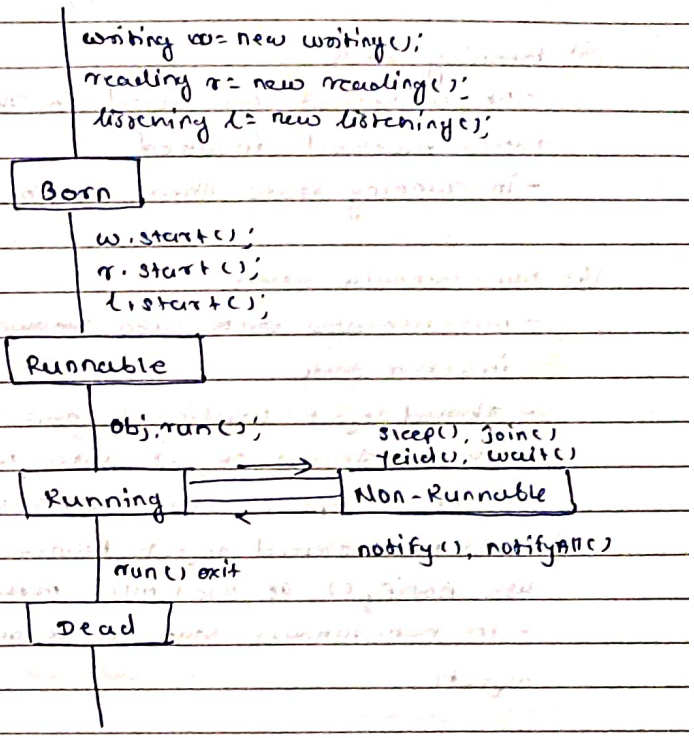
- What is process
  - In java process is a collect^n of thread.
  - process is a heavy weight app^n
  - process is an unshared resource app^n
  - switching bet^n two process is expensive




```
class writing extends Thread
{
  public void bookwriting()
  {
    for(int i=1; i<=10; i++)
    {
      System.out.println("book writting process");
    }
  }
  public void run()
  {
    bookwriting();
  }
}

class Reading extends Thread
{
  public void bookreading()
  {
    for(int i=1; i<=10; i++)
    {
      s.o.p("book reading process");
    }
  }
  public void run()
  {
    bookreading();
  }
}
```

```
class listening extends Thread
{
    public void musicListening()
    {   for(int i=1; i<=10; i++)
        S.O.p("music listening process");
    }
    public void run()
    {
        musicListening();
    }
}

class mainclass
{
    public static void main(String args[])
    {
        writing w = new writing();
        reading r = new reading();
        listening l = new listening();
        w.start();
        r.start();
        l.start();
    }
}
```

lifecycle of thread



① Born State
- Born state also known as a new or start state
- thread object considered in born state during object creation
- in born state, there can be multiple thread object

② Runnable State
- runnable state also known as thread pool
- thred object considered in runnable state during start() method invocked
- in runnable state, there can be multiple thread object

NOTE: thread object can be passed into runnable state once in lifecycle

we cannot invoked start() method twice, or will get IllegalThreadException

③ Running state
- Thread object considered in running state during run() method invoked
- in running state, there can be only one thread.

④ non-runnable state
- non-runnable state also known as a waiting, blocked inactive state
- thread object waiting in this state for their release
- to put thread object into non-runnable state, we can use sleep(), join(), yeild(), wait(), etc method
- to free thread object from non-runnable we can use notify(), or notifyAll() method.
- in non-runnable state, there can be multiple thread objects

⑤ Dead state
- thread object considered in dead state if run() method exit
- in dead state, there can be multiple thread state.

How to create thread in Java
① by using thread class

② by using runnable interface
- runnable is an interface, which is located in java.lang.pkg.
- Runnable interface contains run() method only
- Runnable interface considered as a best way to create thread, because we can extends required class further

eg.

```
class writing implements Runnable
class reading implements Runnable
class listening implements Runnable

class mainclass
{
    public static void main (String args[])
    {
        writing w = new writing();
        reading r = new reading();
        listening l = new listening();
        Thread t1 = new Thread(w);
        Thread t2 = new Thread(r);
        Thread t3 = new Thread(l);
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Thread vs Runnable

Thread	Runnable
① Thread is a class, which is located in java.lang.pkg	① Runnable is an interface which is located in java.lang.
② Thread class contains sleep(), join(), yeild(), getId(), etc methods	② Runnable interface contains run() method

① to use Thread, we can use extends keyword

② By using thread class, we can create thread

① to use Runnable, we can use implements keyword

② Runnable interface considered as a best way to create thread.

Methods of thread

1) start()
2) run()
3) sleep()
4) join()
5) currentThread()
6) getId()
7) getName()
8) setName()
9) getPriority()
10) setPriority()
11) getThreadGroup()
12) getParent()
13) isAlive()
14) isDaemon()
15) setDaemon()
16) Yeild()
17) wait()
18) notify()
19) notifyAll()

1) start()
- start() method can be used to start execution of thread.
- Thread object considered in runnable state during start() method invoked
- thread object passed into runnable state once in an appln
- hence we can not invoked start() method twice or we will get IllegalThreadStateException.

2) run()
- run() method considered in running state of thread
- In running state there can be only one thread
- run method invoked implicitely by thread scheduler
- if we invoked run() method explicitely, it will treat like process not thread

3) sleep()
- sleep() static method of thread class can be used to put thread object into non-runnable state ~~of~~ for certain amount of time
- here amount of time can be specify in miliseconds
- sleep() method throws with InterruptedException.

4) join()
- join() method can be used to put other thread object into non-runnable state for certain amount of time
- here amount of time can be specified in miliseconds
- if amount of time not specified, other thread will wait until run() method exist.
- join() method throws with InterruptedException

Eg:

```
class writing implements Runnable
{
    public void bookwriting()
    {
        for(int i=1; i<=10; i++)
        {
        System.out.println("book writing processing");
        try{
        Thread.sleep(1000);
        }catch (InterruptedException e){
                System.out.println(e);
        }
    }
}
public void run()
{
    bookwriting();
}
}

class Reading implements Runnable
{

    //

}

class Listening implements Runnable
{

    //

}
```

```
class Mainclass
{
    public static void main(String args[])
    {
        writing w = new writing();
        Reading r = new Reading();
        Listening l = new listening();
        Thread t1 = new Thread(w);
        Thread t2 = new Thread(r);
        Thread t3 = new Thread(l);
            t2.start()
            t1.start();
            try{
            t2.join(3000);
            }catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            t3.start();
        }
}
```

5) currentThread()

- currentThread() static method of thread class can be used to represent current executing thread object.

e.g

```
class Site
{
    public void doReservation()
    {
        ~~Thread t = new~~
        Thread t = Thread.currentThread();
        for(int i=1; i<=10; i++)
        System.out.println(t+"It reservation processing");
```

```
}
}

class user implements Runnable
{
    private Site site;
    User(Site site) { this.site = site; }
    user() { }
    public void run()
    {
        site.doReservation();
    }
}

class mainclass
{
    public static void main(String args[])
    {
        Site s = new Site();
        User u1 = new User(s);
        User u2 = new User(s);
        User u3 = new User(s);
        User u4 = new User(s);
        Thread t1 = new Thread(u1);
        Thread t2 = new Thread(u2);
        Thread t3 = new Thread(u3);
        Thread t4 = new Thread(u4);
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```

6) getName()
- getName() method can be used to represent name of thread

7) setName()
- setName() method can be used to set name of thread
- defaut name of thread set as a Thread-0, Thread-1 & so on.

8) getId()
- getId() method can be use to represent ID of thread
- id of thread initialized by thread scheduler implicitely
- this id remains immutable in it's lifecycle.
- Id can be reused after execution of thread.

9) getPriority()
- getPriority() can be used to represent priority of thread

10) SetPriority()
- setpriority() can be used to set priority of thread
- priority of thread can be set in 1, 5 & 10 form
- 1-MIN_PRIORITY
- 5-NORM_ "
- 10-MAX_ "

11) getThreadGroup()
- getThreadGroup() can be used to represent source of a thread

12) getPatent()
- getPatent() can be used to represent source of process

13) isAlive()
- isAlive() can be used to check thread is alive or not
- in multithreading thread object considered as an alive until it exist from run() method.

14) isDaemon()
- isDaemon can be used to check thread is daemon or not

15) SetDaemon()
- SetDaemon() method can be used to denotes thread is daemon thread
- in multithreading daemon thread executes only if non-daemon thread execute
- by using daemon thread, we can perform background task like closing connect^n, cleanup process etc

16) yeild()
- yeild method can be used to execute other lower priority thread object.

17) wait()
- wait() method can be used to put thread object into non-runnable state until notify() or notifyAll method invoked
~~- notif~~

18) wait(long milisecond)
- wait() method can be used to put thread object into non-runnable state until notify() or notifyAll() method should invoked before mentioned time or thread will passed into dead state.

19) notify
- notify() method can be used to free thread object from runnable state.

20) notifyAll()
- notifyAll() method can be used to free all thread from non-runnable state.

Synchronization
- In Java synchronizat^n is a concept, where only one thread allow to use resource
- To perform synchronizat^n we can used synchronized keyword
- if any threads run methods invokes synchronized keyword, thread lock itself with that synchronized keyword until synchronized keyword processed.
- till thread other thread can not interfere, hence it is considered as a slower
- synchronizat^n also known as a thread-safe implementation

Types of synchronizat^n
① Method synchronizat^n
- in method synchronizat^n, method declared with synchronized keyword

- if any threads run() method invoked synchronized method thread lock itself with that method, until exists from synchronized method, till that other thread can not interfere
- method synchronizatn mostly used in reservatn booking etc

eg.

```
class Site
{
    private boolean booked = false;
    synchronized public void doReservatn()
    {
        Thread t = Thread.currentThread();
        if (!booked)
        {
            for (int i=1; i<=10; i++)
            System.out.println (t.getName()+"\t reservatn processing");
            booked = true;
            s.o.p (t.getName() + " your ticket booked");
        }
        else
            s.o.p (t.getName()+"ticket not available");
    }
}

class MainClass
{
    public static void main (String args[])
    {
        --//--
    }
}
```

② block synchronizatn

- In block synchronizatn, synchronized block can be used
- if any threads run() method invocked synchronized block, thread ~~to~~ lock itself with that block until it exist from that block, till that other thread can not interfere
- block synchronizatn mostly used in gaming appln

eg.

```
class Site
{
    private boolean booked = false;
    public void doReservatn()
    {
        Thread t = Thread.currentThread();
        for (int i=1; i<=10; i++)
            System.out.println (t.getName()+"\t reservatn processing");
        synchronized (this)
        {
            if (!booked)
            {
                for (int i=1; i<=10; i++)
                s.o.p (t.getName()+"\t payment processing");
                booked = true;
                s.o.p(t.getName() + " your ticket booked successfully");
            }
            else
                s.o.p("Sorry " +t.getName() + "ticket not available");
        }
    }
}
```

marshalling
unmarshalling

PAGE NO
DATE

③ Static Synchronization

- in static synchronizeth, method declared with static & synchronized keyword
- if any thread's run() method invoked synchronized method, thread locks itself with that synchronized method until it exists from that method, till that other thread can not interfere
- static synchronizeth can be used to provide resource for limited time

e.g. slw activath, antivirus, for limited device, voucher etc.

e.g.

```
class Site
{
  private static int count=0;
  static synchronized public void doReserveth()
  {
    Thread t = Thread.currentThread();
    if(count<2)
    {
      for(int i=1; i<=10; i++)
      S.o.p(t.getName()+" It reserved processig ");
      count++;
      S.o.p(t.getName()+" It your ticket booked
                                    succesfully");
    }
    else
    S.o.p("sorry "+t.getName()+" ticket not
                                    available");
  }
}
```