OOPS NOTES

Java Features are

->simple

->Platform Independent

->Architectural neutral

->portability

->multi-threading

->security

->object-orientation

->Robust

->high-performance

->distrubute

->compiled and interpreted

Datatypes:

Datatypes are used to specify the value to the variable and it helps to convert data into binary format.

In java we have 8 premitive Datatypes are there

1.byte 2. short 3.int 4.long

5.float 6.double

7.boolean 8.char

There are 6pillars / features of object orientation are

->class

->object

->Encapsulation

->inheritance

->Abstraction

->polymorphism

Note: The first object oriented programming language is "simla" introduced in 1967.

About class:

->class is atype.

->class is a collection of objects.

->class contains properties and behaviours

->classes doesn't exist in real world

->classes doesn't occupy memory.

syntax:

```
class classname
{
      // variables
    // methods
  // constructors
// blocks
// interfaces
// abstract class
// enum
}
```

object:

->object is an instance of class

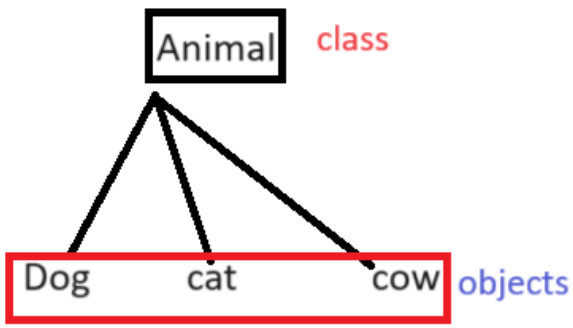
->object existed in real world.

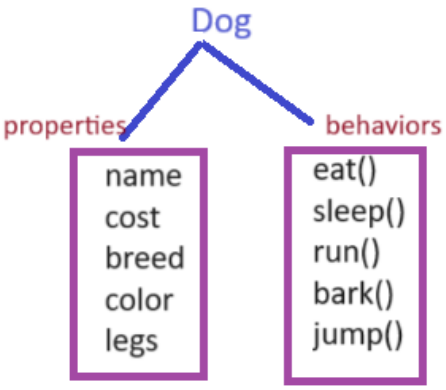
->objects occupy memory

->Object is a collection of Data.

->Each object contains has parts(properties) and does parts(behaviours)

example:






```
class Dog
{
        String name;
        int cost;
        String breed;
        String color;
        int legs;

      void eat()
      {
            System.out.println("Dog is eating");
      }
```

```
    void sleep()
  {
        System.out.println("Dog is sleeping");
 }
}
```

Note:In java If we want to execute any class we have to create object .

creation of object

object can be created by using 5 ways

1.By using new keyword

2.new instanceOf method

3.clone method

4.factory method

5.de-serialization

creation of object using new keyword

object can be created by using new keyword by using 3 steps

1.declaration : Declare variable to the type(class)

2.Instantiation

->allocate the memory(new) and ponited to the reference variable.

3.initialization.

creating object to the Dog class

```
class Execution
{
	public static void main(String[] args)
	{
		Dog d;	// declaration
		d=new Dog();	// creating f object
		d.name="Leo";
		d.color="White";
		d.cost = 10000;
		d.legs=4;

  System.out.println(d.name);
  System.out.println(d.color);
  System.out.println(d.cost);
  System.out.println(d.legs);
```

```
    }
}
```

About public static void main(String[] args)

public :main method made as public so that it can visible to the JVM and JVM can call main method fromany where in the memory.

static :main method made as static so that JVM can access the main method without creation of object.

void:After execution of main method, the main will not return any any value to the JVM

main(): main is the entry point to JVM

String[] args: we call command line argumnets

->we can pass the values during commands

example:

```
class Demo
{
    public static void main(String[] args)
    {
            System.out.println(args[0]);
          System.out.println(args[1]);
```

```
                System.out.println(args[2]);
        }
}
```

javac Demo.java

java Demo 10 20 30

i.e: When we pass values during command those values will be assigned.

Variables:

variables are classified into 2 types

1.premitive variable :The variables which stores the premitive Data.

ex: int a=10; float b=12.9f;

2.reference variable : The variables which stores the address/memory location of the object.

->Based on the position and declaration of the variables they are again divided into 2 categories

1.instance variable

2.local variable

->instance variable

->If a variable declared with in a class then it is called

instance variable.

->Instance variables present inside the heap segment

scope of instance variables

-> Scope means memory allocation and memory-deallocation

->for the instance variable memory will be allocated during object creation.

->for the instance variable memory will be de allocated during obect destruction.

2.local variable

->If a variable declared with in a method then it is called local variable.

->Local variables present inside the stack segment.

scope of local variables

->for the local variables memory is allocated when the control enters into the method

->for the local variables memory will be de allocated when the control leaves from the method.

Q)Differences between instance and local variable?

A:

Instance variable	local variable
->instance variables are declared inside the class	->local variables are declared inside the method or block
->default values are given by the JVM depends upon the datatype	->no default values are provided by the JVM
->memory allocated during the object creation	->memory allocated when the control enters into the method or a block
->memory de-allocated during the object destruction	->memory de-allocated when the control leaves from the method or a block

Array:

->Array is a collection of homogeneous elements.

->The Array indicated with square braces []

->The advantage of array is to store large amount of data can be stored and accessed very simple.

->The dis-advantage of an Array is size can't be changed during runtime.

->To overcome the above problem Java peoples introduced the "COllection Framework".

Note:

->Array is an object since it is created using new keyword.

->The super class of an Array "Object".

->When the array is created default values will be added based on datatype.

->At the time of array declaration size should not be specified

size should be specified during object creation

Types of Arrays:

1.single dimensional Array(1-D)

2.Multi-dimensional Array

creation of an Array

1.declaration

2.Creation of object

3.initialization

Single dimensional Array

In single dimensional Array we can store values either in a rows or a column

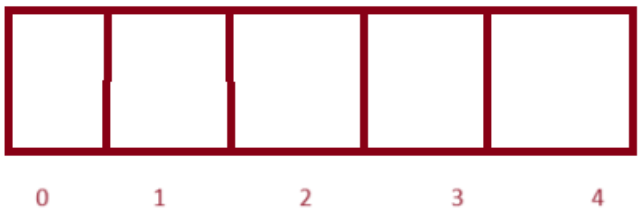
Array declaration of 1-D

int[] a or int []a or int a[]

Example:

class App

```
{
        public static void main(String[] args)
        {
                int a[];
                a=new int[5];    // 5 is size to store elements
into an array
```




```
              //initialization
            a[0]=10;
            a[1]=20;
            a[2]=30;
            a[3]=40;
            a[4]=50;


          System.out.println(a[0]);
        }
}
```

->The above program we can optimize the code

```
public static void main(String[] args)
{
   int[] a=new int[5];
   Scanner scan=new Scanner(System.in);

   for(int i=0;i<=a.length-1;i++)  // storing the values into Array
   {
     System.out.println("Enter the index:"+i);
     a[i]=scan.nextInt();
   }
   for(int i=0;i<=a.length-1;i++)  // Fetching values from an Array
   {
     System.out.print(a[i]+" ");
   }
}
```

>Length is the pre-defined variable it will be loaded internally the value from size

practice questions:

Q)write a java program to store and print the marks of the 8 students?

Q)write a java program to store and print the salaries of the 6 employees?

multi-dimensionalArray(2-D):

->In multi-dimensional(2-D) Array the values are stores in rows and columns.

declaration of 2-D Array

int[][] a;

int a[][];

int [][]a;

2-D Array divided into 2 categories

1.Regular 2-D Array : If no.ofrows == no.of.columns

2.Jagged 2-D Array : if no.of.rows != no.of.columns

Regular 2-D Array

example:Write a java programto store and print the marks of the 5 students in 3 classrooms?

classroom = 3;

students =5;

```
public class App1
{
  int[][] a=new int[3][5];
  public void logic()
  {
     Scanner scan=new Scanner(System.in);

     for(int i=0;i<=a.length-1;i++)  // storing the values into Array
     {
       for(int j=0;j<=a[i].length-1;j++){
           System.out.println("Enter the index:"+i);
           a[i][j]=scan.nextInt();
       }
       System.out.println();
     }
     for(int i=0;i<=a.length-1;i++)  // storing the values into Array
     {
       for(int j=0;j<=a[i].length-1;j++){
           System.out.print(a[i][j]+" ");
       }
       System.out.println();
     }
  }
```

practice questions:

Q)write a java program to store and print the salaries of the 10 employees in the 3 departments?

Jagged 2-D Array

example:write a java program to store and print the marks of the students

classrooms	students
1	3
2	5
3	4

```
public class App1
{
  int[][] a=new int[3][];
  public void logic()
  {
    a[0]=new int[3];
    a[1]=new int[5];
    a[2]=new int[4];
     Scanner scan=new Scanner(System.in);
     for(int i=0;i<=a.length-1;i++)  // storing the values into Array
     {
       for(int j=0;j<=a[i].length-1;j++){
           System.out.println("Enter the index:"+i);
           a[i][j]=scan.nextInt();
       }
       System.out.println();
     }
     for(int i=0;i<=a.length-1;i++)  // storing the values into Array
     {
       for(int j=0;j<=a[i].length-1;j++){
           System.out.print(a[i][j]+" ");
       }
       System.out.println();
     }
  }
```

Q)write a java program to store and print the salaries of the employees

Departments	Employees
1	4
2	6
3	2
4	5
5	3

methods:methods are set of statements used to perform a specific task.

syntax:

access-modifier returntype methodname(parameters)

```
{
        // method body
}
```

Different types of methods

1.A method which doesn't accept any parameter and doesn't return any value

example:

```
public class App01
{
  int a=10;
  int b=20;
  public void fun1()
  {
      int sum=a+b;
      System.out.println("Addition of value :"+sum);
  }
  public static void main(String[] args) {
    App01 app=new App01();
    app.fun1();
  }
}
```

2.A method which accepts the parameter but doesn't return any value.

```
public class App01
{

  public void fun1(int a,int b)
  {
      int sum=a+b;
      System.out.println("Addition of value :"+sum);
  }
  public static void main(String[] args) {
    App01 app=new App01();
    app.fun1(12,13);
  }
}
```

3.A method which accepts parameters and return the value.

```java
public class App01
{

  public int fun1(int a,int b)
  {
      int sum=a+b;
      return sum;
  }
  public static void main(String[] args) {
    App01 app=new App01();
    System.out.println("Addition of value :"+app.fun1(12,13));
  }
}
```

4.A method which doesn't accepts parameter and return the value

```java
public class App01
{
  int a=12;
  int b=15;
  public int fun1()
  {
      int sum=a+b;
      return sum;
  }
  public static void main(String[] args) {
    App01 app=new App01();
    System.out.println("Addition of value :"+app.fun1());
  }
}
```

mutators and accessors

Based on the object state modifications methods are classified into 2 types

1.mutators (setters)

2.accessors (getters)

->mutators(setters) are used to set and modify the value in object.

->Accessors(getters) are used to return/get the value from an object.

Encapsulation

->Encapsulation is the one of the important object oriented featue in java.

->It refers to the rapping the data(variables) and code acting on the data(methods) as a single unit.

->Encapsulation can be achieved using in java

by making class variables has private

by creating public setters and getters methods inorder to access and view the data.

examples: Human - heart is private thing

coconut - coconut water is private thing

capsule - tablet

```java
public class StudentApp {
  private String name;
  private int id;
  private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
```

```java
public class Execution
{
    public static void main(String[] args)
    {
        StudentApp s=new StudentApp();
        s.setId(101);
        s.setName("Ravi");
        s.setAge(24);
        System.out.println(s.getId());
        System.out.println(s.getName());
        System.out.println(s.getAge());
    }
}
```

Data hiding: Hiding the Data so that outside person can't access the data directly this can be achieved by making the data members has private.

Advantages of Encapsulation:

1.security.

2.enhancement is easy.

3.Data hiding.

constructor:

object of values can be initialized by using 3 ways

1.using reference variable

2.using setters and getters methods

3.using constructor

->constructor is a block where the name of the constructor is same as class name.

->It is a specialized setters to initialize the values of object.

->constructors doesn't have any return type.

->The access-modifiers associated with constructors are

public, protected,default and private

Note :constructors are called during object creation.

```java
public class StudentApp {
  private String name;
  private int id;
  private int age;

   public StudentApp(String name, int id, int age)
   {
    super();
    this.name = name;
    this.id = id;
    this.age = age;
   }
   public String  getData()
   {
       return "id ="+id+", name ="+name+", age ="+age;
   }
}
```

```java
public class Execution {
    public static void main(String[] args) {
        StudentApp s=new StudentApp("Sai",101,23);
        System.out.println(s.getData());
    }
}
```

->In a class if we are not writing constructor then java compiler define the no-argument constructor.

this keyword: this is a keyword which refers to the current class instance variable

use case:whenever instnace variable and local variable names are same to show differences between them we use this keyword

constrcutor overloading:

overloading is same name having different parameter

```
public class StudentApp
{
  // constructor name is same but having different arguments
  public StudentApp()
  {
      System.out.println("constructor (studentApp) contains no-
arguments");
  }
  public StudentApp(int a)
  {
     System.out.println("construutor (studentApp) contains 1-
argument :"+a);
  }
  public StudentApp(int a,String b)
  {
     System.out.println("construutor (studentApp) contains 2-
arguments");
     System.out.println("a value :"+a+", b value is :"+b);
  }
}
```

```
public class Execution {
    public static void main(String[] args) {
        StudentApp s1=new StudentApp();
        StudentApp s2=new StudentApp(10);
        StudentApp s3=new StudentApp(10,"krishna");
    }
}
```

->In the program even though we are doing constructor overloading, to call the constructor we are creating objects.

->whenever we are doing overloading in constructor we can call one constructor inside another constructor by using this().

```
public class StudentApp
{
  // constructor name is same but having different arguments
  public StudentApp()
  {
      System.out.println("constructor (studentApp) contains no-arguments");
  }
  public StudentApp(int a)
  {
     this();
     System.out.println("construutor (studentApp) contains 1-argument :"+a);
  }
  public StudentApp(int a,String b)
  {
     this(100);
     System.out.println("construutor (studentApp) contains 2-arguments");
     System.out.println("a value :"+a+", b value is :"+b);
  }
}
```

```
public class Execution {
    public static void main(String[] args) {
        StudentApp s3=new StudentApp(10,"krishna");
    }
}
```

Q)Differences between constructor and method?

A:

constructor	method
=>constructor name should be same as class name =>constructor's doesn't have return type =>constructors are called implicitly during object creation =>constructors are called only once during object creation =>access specifiers associated with constructors are public,private,protected,default	=>name of the method can be anything =>methods do have return type =>methods are called explicitly by programmer => methods can be called several types =>Apart from these 4 access specifiers can also have static,final

method-overloading:

In c-language we have one drawback

i.e:we can write n number of functions but function name should be unique(no two functions should have the same name)

Disadvantage of c-language

Rememberring the method name to difficult

readability problem

To overcome that problem in java they have introduced method-overloading concept.

method-overloading:

->2 or more methods within a class having same name

with different parameters is called method overloading.

->whenever a class is made for a overloaded methods then compiler will resolve the method call in the following ways

1.name of the method

2.no.of.parameters present inside the method

3.Datatype of the parameter

4.order of the parameter

Note: In java method overloading will not consider return type.

Q)In java method overloading what is overloaded?

A:In method overloading nothing is overloaded

i.e: the programmer is under an illustion that one method is overloaded with multiple activities

but in fact there are different methods doing there respective activities.

->using method overloading we are achieving virtual polymorphism

->using method overriding we can achieve true polymorphism.

example:

```java
public class App01 {
  // add: method  having different parameters
  public int add(int a,int b){
     int res=a+b;
     return res;
  }
  public float add(int a,float b){
     float res=a+b;
     return res;
  }
  public double add(int a,double b){
     double res=a+b;
     return res;
  }
  public float add(float a,float b){
     float res=a+b;
     return res;
  }
  public double add(float a,double b){
    double res=a+b;
    return res;
  }
  public double add(double a,int b){
     double res=a+b;
     return res;
  }
  public float add(float a, int b){
      float res=a+b;
      return res;
  }
}
```

```java
public class Execution {
    public static void main(String[] args) {
        App01 app=new App01();
        int a=10,b=20,c=30;
        float m=12.3f, n=14.5f, p=15.8f;
        double x=120.99,  y=150.67,  z=143.67;

        System.out.println(app.add(a, m));
        System.out.println(app.add(a, b));
        System.out.println(app.add(x, a));
    }
}
```

static keyword:

->static is a keyword which can be associated to the

1.variables

2.blocks

3.methods

static variable : whenever a value is shared to multiple objects at that point we are making that variable has static.

```java
public class Student
{
    private String name;
    private int id;
    private int age;
    static String schlname;

    public Student() {}
    public Student(String name,int id,int age){
        this.name=name;
        this.id=id;
        this.age=age;
    }
    public String display(){
        return "Students info :"+"id ="+id+", name ="+name+", age ="+age+", schlname ="+schlname;
    }
}
```

```java
public class Execution {
    public static void main(String[] args) {
      Student std=new Student("Krishna", 101, 35);
      Student.schlname="Sri chaitanya";
      System.out.println(std.display());
    }
}
```

static variables can be accessed by using class reference.

Note:

1.In static methods we can't access the instance members

2.In instance/concrete method we can access the static members but we may loose static priority.

```java
public class App
{
  static int a,b,c;
  int x,y,z;

  // static block
  static {
    a=10;
    b=20;
    c=30;
  }
  // instance block
  {
      x=100;
      y=200;
      z=300;
  }
  // instance method
  public void fun1() {
      System.out.println(a);
      System.out.println(b);
      System.out.println(c);
      System.out.println(x);
      System.out.println(y);
      System.out.println(z);
  }
  // static method
  public static void fun2(){
      System.out.println(a);
      System.out.println(b);
      System.out.println(c);
      System.out.println(x);
      System.out.println(y);
      System.out.println(z);
  }
}
```

error

x,y,z are non-static members

```java
public class Execution {
    public static void main(String[] args) {
      App app=new App();
      app.fun1();
      app.fun2();
    }
}
```

Q)Differences between instance method and static method?

A:

instance method	static method
=>instance methods are also called as Non-static methods	=>static methods are also called as class methods
=>Inorder to call the method object should be created	=> No need to create object
=>it can be accessed by using object reference	=> it can be accessed by using class name
=>it deals with objects	=> it deals with classes

Q)Differences between instance,local and static variable?

A:

	Local variables	instance variables	static variables
declaration	inside the methods/constructors/ blocks	inside the class but not present in methods/constructors//blocks	inside the class with static keyword but not allowed insid ethe constructor/method/block
scope	inside methods/constructors/ blocks	inside class anywhere except static block & static methods	inside static/instance methods , constructors/ blocks
memory allocation	memory gets allocated when the control enters method/constructor/block & de-allocated when control leaves from the constructor/ block/method. memory allocated in stack segment.	memory gets allocated during the object creation & gets de-allocated when object gets destroyed. memory allocated in heap segment.	during class loading memory gets allocated. and class file is un-loaded memory gets de-allocated. memory allocated in static segment
default values	no default values	default values depends upon datatype	default values depends upon datatype
Access modifier	not applicable	access specifiers are allowed (private,public protected,default)	access specifiers are allowed (private,public protected,default)

Inheritance:

->Inheritance is the process of a class acquiring the properties and behaviours of another class.

->inheritance can be achieved by using extends keyword.

->parent class is a class which gives the properties and behaviours to the another class.

parent class another name super class or base class.

->child class is a class which takes / inherits the properties and behaviour from the other class.

child class is also called as sub class / derived class.

Advantages of Inheritance

1.code reusability

2.less time developing the softwares.

Note 1: In order to relate 2 classes we use extends keyword.

->Inheritance promotes IS-A relation ship between them.

```
public class Parent
{
  int i=10;
}
public class Child extends Parent
{
 int j=20;
}
public class InheritanceDemo {

    public static void main(String[] args)
    {
        Child c1=new Child();
    }
}
```

Note 2: whenever the object of child class is created the memory is not allocates for the instance variables of child class but it also allocates for the instance variables of

parent class.

```
public class Parent
{
   int i=10;
}
public class Child extends Parent
{
  int j=20;
}
public class InheritanceDemo {

      public static void main(String[] args)
      {
            Child c1=new Child();
            System.out.println(c1.i);
            System.out.println(c1.j);
      }
}
```

Note 3: private members will not participate in inheritance.

This rule is made to promote Encapsulation

Note 4: The child class can call instance methods of a parent class directly and

child class can call inherited static methods directly.

```
public class Parent
{
   int i=10;

   public int display()
   {
       return i;
   }
}
public class Child extends Parent
{
  int j=20;

  public int info()
  {
      return j;
  }
}
```

```
public class InheritanceDemo {

    public static void main(String[] args)
    {
        Child c1=new Child();
        System.out.println(c1.display());
        System.out.println(c1.info());
    }
}
```

Note-5:unlike private members even constructors doesn't participate in inheritance.
Parent class constructor could not be inherited to child class rather the control from the child class will goto super class and code gets executed.

```
public class Parent {
   int i=10;
   public Parent(){
       System.out.println("constructor(parent) has no argumnets");
   }
}

public class Child extends Parent{
  int j=20;
  public Child(){
      super();
      System.out.println("constructor(child) has no arguments");
  }
}

public class InheritanceDemo {
    public static void main(String[] args) {
        Child c1=new Child();
    }
}
```

Note-6:In the child class if we call parameter constructor first it will execute parent class no argument constructor after that child class parameter constructor will be executed.

```java
public class Parent {
   int i=10;
   public Parent(){
       System.out.println("constructor(parent) has no argumnets");
   }
}

 public class Child extends Parent{
   int j=20;
   public Child(){
       super();
       System.out.println("constructor(child) has no arguments");
   }
   public Child(int a,int b) {
       System.out.println("constructor(child) has 2 arguments");
       System.out.println(a);
       System.out.println(b);
   }
 }

  public class InheritanceDemo {
      public static void main(String[] args) {
          Child c1=new Child(10,12);
      }
  }
```

example-1:

```java
public class Parent {
    int i=10;
    public Parent(){
        System.out.println("constructor(parent) has no argumnets");
    }
    public Parent(int a,int b){
        System.out.println("constructor(parent) has 2 arguments");
        System.out.println(a);
        System.out.println(b);
    }
}

public class Child extends Parent{
  int j=20;
  public Child(){
      super();
      System.out.println("constructor(child) has no arguments");
  }
  public Child(int a,int b) {
      super(100,200);
      System.out.println("constructor(child) has 2 arguments");
      System.out.println(a);
      System.out.println(b);
  }
}

public class InheritanceDemo {
    public static void main(String[] args) {
        Child c1=new Child(10,12);
    }
}
```

example of inheritance:

```java
public class Employee {
    private String name;
    private int id;
    private String email;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String toString() {
        return "id ="+id+", name ="+name+", email ="+email+", age ="+age;
    }
    public Employee(){
        System.out.println("constructor (Employee) class has no arguments");
    }
    public Employee(int id,String name,int age,String email){
        this.id=id;
        this.name=name;
        this.age=age;
        this.email=email;
    }
}
```

```java
public class SalariedEmployee extends Employee
{
  private double salary;

public double getSalary() {
    return salary;
}

public void setSalary(double salary) {
    this.salary = salary;
}

 public String toString()
 {
     return super.toString()+", salary ="+salary;
 }

 public SalariedEmployee()
 {
     System.out.println("constructor (SalariedEmployee) has no arguments");
 }

 public SalariedEmployee(int id,String name,int age,String email,double salary)
 {
     super(id,name,age,email);
     this.salary=salary;
 }
}
```

```java
public class ContractEmployee extends Employee
{
   private double csalary;

public double getCsalary() {
    return csalary;
}

public void setCsalary(double csalary) {
    this.csalary = csalary;
}

  public String toString()
  {
      return super.toString()+", csalary ="+csalary;
  }

  public ContractEmployee()
  {
      System.out.println("constructor (ContractEmployee) has no arguments");
  }

  public ContractEmployee(int id,String name,int age,String email,double csalary)
  {
      super(id,name,age,email);
      this.csalary=csalary;
  }
}
```

```java
public class Execution {

    public static void main(String[] args) {
        System.out.println("Instantiation of slariedEmployee");
        SalariedEmployee s=new SalariedEmployee(101,"Raviteja",24,"teja@gmail.com",60000.00);
        System.out.println(s.toString());

        System.out.println();

        System.out.println("Instantiation of ContractEmployee");
        ContractEmployee c=new ContractEmployee(102,"Sai",23,"sai@gmail.com",2000);
        System.out.println(c.toString());
    }
}
```

Method-overriding:

->Inherited methods are such methods which are inherited by child class and used as it is without any changes

->The method which is inherited by the subclass from super class and the sub class will re-define or provide new implementation for the inherited method and this process is called as method overriding.

->In other words redefine the inherited methods in the child class is called as method overriding.

->The method which is overridden in parent class is called over-ridden method

->The method which is redefined in child class is called over-riding methods.

Advantage of method overriding is Polymorphism

->Using parent reference to child object in overriding

example : 1

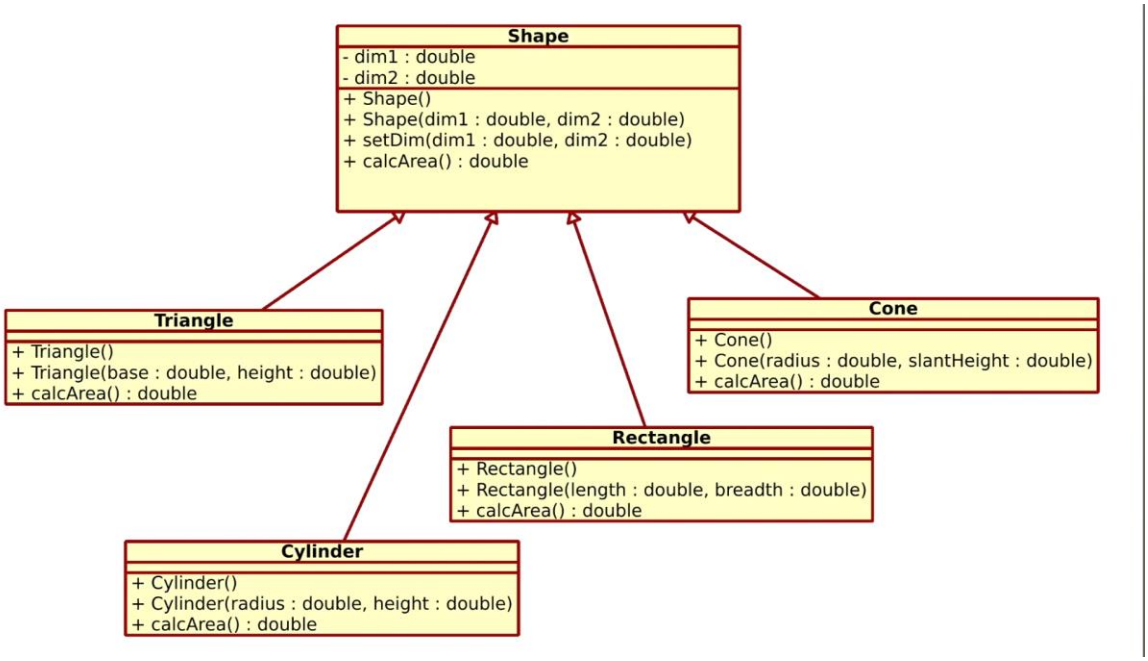
```
public class A {
  public void marry(){
      System.out.println("Parents decided to marry at the age of 25");
  }
}

public class B extends A {
  @Override
  public void marry(){
      System.out.println("child decided to marry at the age of 30");
  }
}

public class C extends A{
  @Override
  public void marry(){
      System.out.println("C has decided to marry at the age of 28");
  }
}

public class UseAB {
    public static void main(String[] args) {
        A a;
        a=new B();
        a.marry();
        System.out.println();
        a=new C();
        a.marry();
    }
}
```

Example-2:



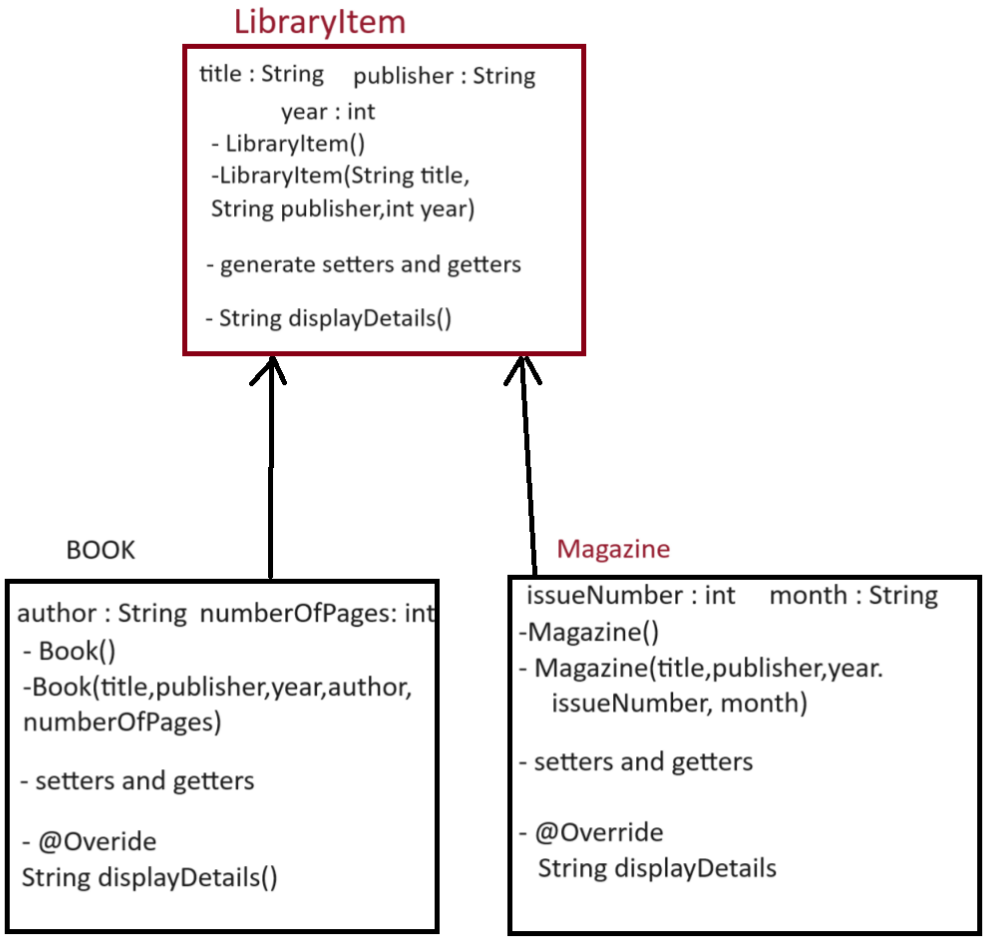
```java
public class Shape {
    double dim1,dim2,area;
    public Shape(){
      dim1=0;
      dim2=0;
    }
    public Shape(double dim1,double dim2){
        this.dim1=dim1;
        this.dim2=dim2;
    }
    public void setDim1(double dim1) {
        this.dim1 = dim1;
    }
    public void setDim2(double dim2) {
        this.dim2 = dim2;
    }
    public double calcArea(){
        return 0.0;
    }
}
```

```java
public class Triangle extends Shape
{
  public Triangle(double base,double height)
  {
     dim1=base;
     dim2=height;
  }
  @Override
  public double calcArea()
  {
      area=0.5*dim1*dim2;
      return area;
  }
}
```

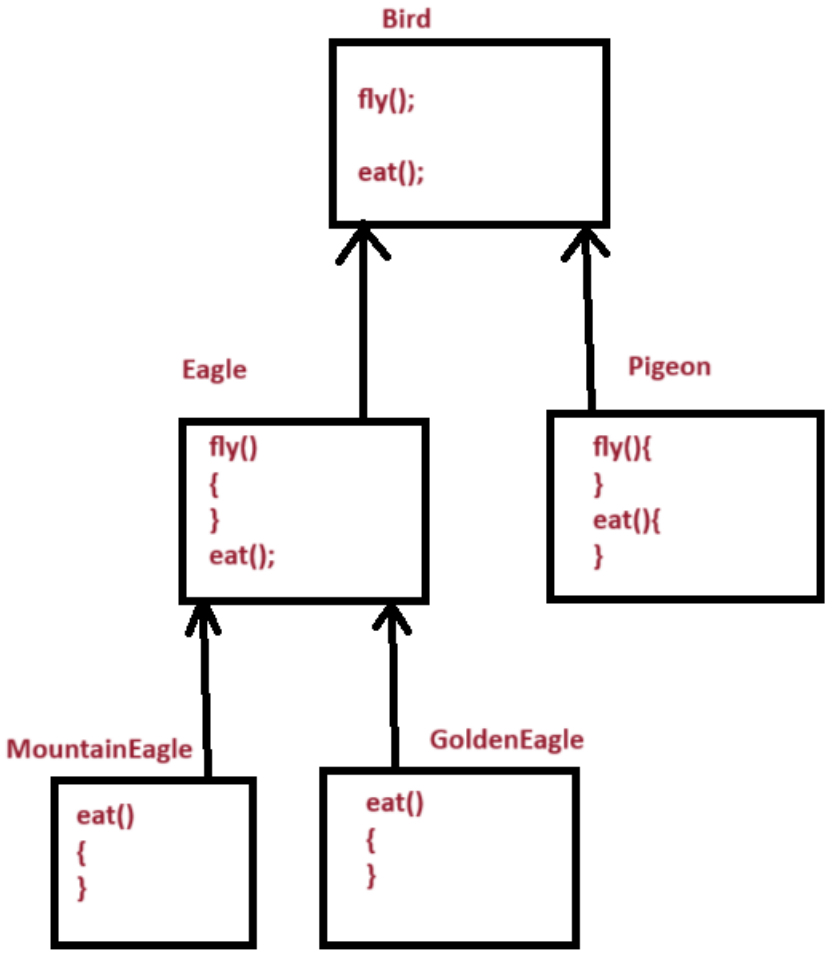
```java
public class Rectangle extends Shape {
  public Rectangle(double len,double bre){
      dim1=len;
      dim2=bre;
  }
  @Override
  public double calcArea(){
      area=dim1*dim2;
      return area;
  }
}
```

```java
public class UseShape {
    public static void main(String[] args) {
        Shape s;
        s=new Triangle(8.7, 5.6);
        double areaTri=s.calcArea();
        System.out.println("Area of the triangle :"+areaTri);
    }
}
```

practice Example-1:

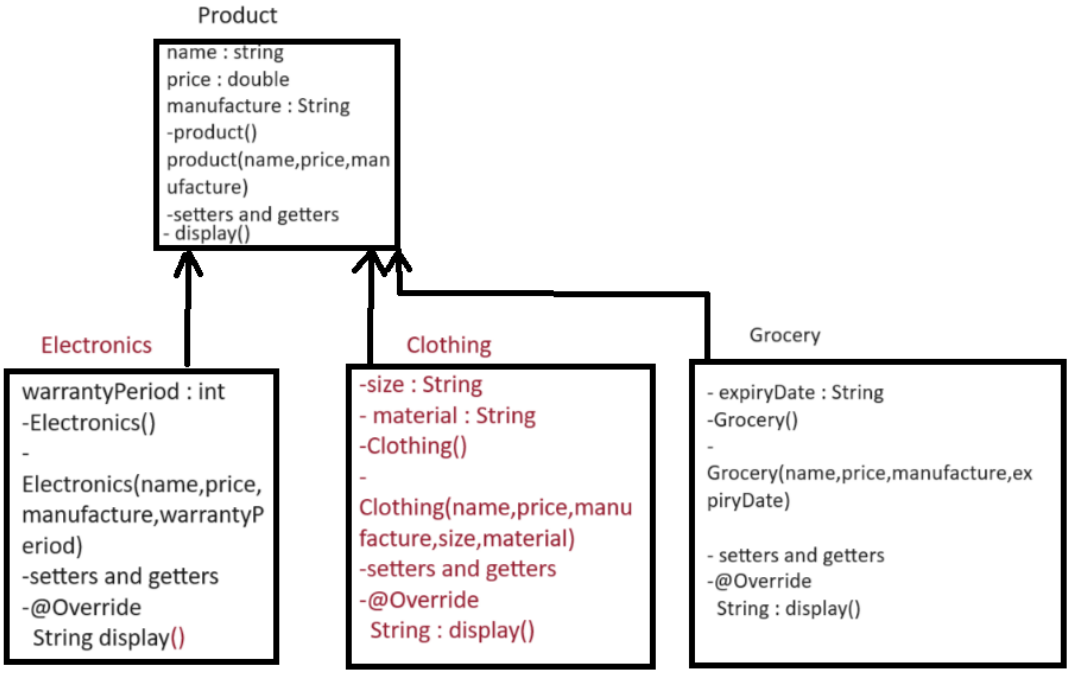

LibraryItem
title : String publisher : String
year : int
- LibraryItem()
-LibraryItem(String title,
String publisher,int year)
- generate setters and getters
- String displayDetails()
BOOK
author : String numberOfPages: int
- Book()
-Book(title,publisher,year,author,
numberOfPages)
- setters and getters
- @Overide
String displayDetails()
Magazine
issueNumber : int month : String
-Magazine()
- Magazine(title,publisher,year.
issueNumber, month)
- setters and getters
- @Override
String displayDetails

Example-2:


Bird
fly();
eat();
Eagle
fly()
{
}
eat();
Pigeon
fly(){
}
eat(){
}
MountainEagle
eat()
{
}
GoldenEagle
eat()
{
}

---

example-3:



---

polymorphism:

polymorphism is one of the most important object oriented feature.

poly : many morphism : many forms

->polymorphism means an object with many forms

Advantage of polymorphism

1.without flexibility code reduction

2.with flexibility code reduction

1.polymorphism without flexibility code reduction

```java
public class Plane {
  public void takeOff(){
      System.out.println("plane is taking off!!!");
  }
  public void fly(){
     System.out.println("plane is flying");
  }
  public void land(){
     System.out.println("plane is landed");
  }
}
```

```java
public class CargoPlane extends Plane
{
  @Override
  public void takeOff(){
     System.out.println("cargoplane is takingOff from longrun way");
  }
  @Override
 public  void fly(){
     System.out.println("cargoplane is flying with slow speed");
  }
  @Override
  public void land(){
      System.out.println("cargoplane is landed in longrun way");
  }
  /**
   * specialization method :
   * */
  public void carryCargo()
  {
      System.out.println("cargoplane carries goods");
  }
}
```

```java
public class PassengerPlane extends Plane{
     @Override
     public void takeOff(){
        System.out.println("PassengerPlane is takingOff from midsize run way");
     }
     @Override
     public void fly(){
        System.out.println("PassengerPlane is flying with medium speed");
     }
     @Override
     public void land(){
         System.out.println("PassengerPlane is landed in midsize runway");
     }
     // specialization method
     public void carryPassenger(){
         System.out.println("passenger plane carries peoples");
     }
}
```

```java
public class FighterPlane extends Plane {
     @Override
     public void takeOff(){
        System.out.println("FighterPlane is takingOff from small run way");
     }
     @Override
     public void fly(){
        System.out.println("FighterPlane is flying with high speed");
     }
     @Override
     public void land()
     {
         System.out.println("FighterPlane is landed in small runway");
     }
     // specialized method
     public void carryArms()
     {
         System.out.println("fighter plane carries arms");
     }
}
```

```java
// without flexibility and code reduction
public class UsePlane {
    public static void main(String[] args) {
        Plane ref;
        ref=new CargoPlane();
        ref.takeOff();
        ref.fly();
        ref.land();
        ((CargoPlane)(ref)).carryCargo(); // calling specialized method
        ref=new PassengerPlane();
        ref.takeOff();
        ref.fly();
        ref.land();
        ((PassengerPlane)(ref)).carryPassenger(); // calling specialized method
        ref=new FighterPlane();
        ref.takeOff();
        ref.fly();
        ref.land();
        ((FighterPlane)(ref)).carryArms(); // calling specialized method
    }
}
```

2.with flexibility code reduction

->same code from the above program only change in main method

```
public class Airport {
 public void allow(Plane ref){
     ref.takeOff();
     ref.fly();
     ref.land();
 }
}
```

```
// with flexibility and code reduction
public class UsePlane {
    public static void main(String[] args) {
        CargoPlane c=new CargoPlane();
        PassengerPlane p=new PassengerPlane();
        FighterPlane f=new FighterPlane();
        Airport a=new Airport();
        a.allow(c);
    }
}
```

->In methodoverriding we can do method overloading also.same code just replace this.

```
// allow : method overloading
public class Airport {
 public void allow(PassengerPlane ref){
     ref.takeOff();
     ref.fly();
     ref.land();
     ((PassengerPlane)(ref)).carryPassenger();
 }
 public void allow(CargoPlane ref){
     ref.takeOff();
     ref.fly();
     ref.land();
     ((CargoPlane)(ref)).carryCargo();
 }
 public void allow(FighterPlane ref){
     ref.takeOff();
     ref.fly();
     ref.land();
     ((FighterPlane)(ref)).carryArms();
 }
}
```

Abstraction:

->Abstraction is the of the important object oriented feature.

->It is the process of hidding the internal implementation

and showing wanted functionality to the user is called abstraction.

Real time example on abstraction

->If I want to send a sms to friend from my phone we just type a text but we are not aware of internal processing.

examples: car, bike,ATM etc....

-> Abstraction is functionality hiding

->Encapsulation is Data hiding

->In java abstraction can be achieved by using 2 ways

1.abstract class

2.interface

->Using abstract class we can achieve 0 to 100% abstraction

->Using interface we can achieve 100% abstraction

Abstract class and Method

->A class which is declared as abstract is called abstract class.

->An abstract class contains zero or more abstract methods.

->object can't be created for abstract class since it is incomplete class.

->we can create reference to the abstract class.

->abstract is a non-access modifier which can be used for

1.class 2. method 3. interface 4. inner class

example:

```
public abstract class BankAccount {
  private String accountNumber;
  private double balance;
    public BankAccount() {
        super();
        // TODO Auto-generated constructor stub
    }
    public BankAccount(String accountNumber, double balance) {
        super();
        this.accountNumber = accountNumber;
        this.balance = balance;
    }
    public String getAccountNumber() {
        return accountNumber;
    }
    public void setAccountNumber(String accountNumber) {
        this.accountNumber = accountNumber;
    }
    public double getBalance() {
        return balance;
    }
    public void setBalance(double balance) {
        this.balance = balance;
    }
    public void deposit(double amount){
        if(amount > 0) {
            balance += amount;
            System.out.println("Deposited :"+amount);
        }
        else {
            System.out.println("Invalid deposit amount");
        }
    }
    public void withdrawn(double amount) {
        if(amount > 0 && amount <= balance) {
           balance -=amount;
           System.out.println("withdrawn :"+amount);
        }
        else {
            System.out.println("Invalid withdrawn amount/ insufficient balance");
        }
    }
    public abstract void calculateInterest();
    public abstract void displayAccountDetails();
}
```

```java
public class CurrentAccount extends BankAccount {
    private double overdraftLimit;

    public CurrentAccount() {
        super();
    }

    public CurrentAccount(String accountNumber, double balance,double overdraftLimit) {
        super(accountNumber, balance);
        this.overdraftLimit=overdraftLimit;
    }

    public double getOverdraftLimit() {
        return overdraftLimit;
    }

    public void setOverdraftLimit(double overdraftLimit) {
        this.overdraftLimit = overdraftLimit;
    }

    @Override
    public void calculateInterest() {
        // current accounts may not have interest calculation
        System.out.println("current accounts don't have interest");
    }

    @Override
    public void displayAccountDetails() {
        System.out.println("current Account Number :"+getAccountNumber());
        System.out.println("Current Balance :"+getBalance());
        System.out.println("overdraft limit :"+getOverdraftLimit());
    }

    @Override
    public void withdrawn(double amount)
    {
        if(amount > 0 && amount <= (getBalance() + getOverdraftLimit())) {
            double newBalance = getBalance() - amount;
            if(newBalance < 0) {
                overdraftLimit +=newBalance; // adjust overdraft limit
            }
            super.setBalance(newBalance);
            System.out.println("withdrawn :"+amount);
        }
        else {
            System.out.println("Insufficient Funds/overdraft limit");
        }
    }
}
```

```java
public class SavingsAccount extends BankAccount {
    private double interestRate;

    public SavingsAccount() {
        super();
    }

    public SavingsAccount(String accountNumber, double balance, double interestRate) {
        super(accountNumber, balance);
        this.interestRate = interestRate;
    }

    public double getInterestRate() {
        return interestRate;
    }

    public void setInterestRate(double interestRate) {
        this.interestRate = interestRate;
    }

    @Override
    public void calculateInterest() {
        double interest = getBalance() * getInterestRate()/100;
        deposit(interest);
        System.out.println("Interest calculated :"+interest);
    }

    @Override
    public void displayAccountDetails() {
        System.out.println("Saving account number :"+getAccountNumber());
        System.out.println("Balance :"+getBalance());
        System.out.println("Interest rate :"+interestRate + "%");
    }

}
```

```java
public class UseBank {

    public static void main(String[] args) {
        BankAccount bank;
        bank=new CurrentAccount("c12345", 5000.0,1000.0);
        bank.displayAccountDetails();
        bank.withdrawn(6000);
        bank.displayAccountDetails();
        System.out.println();
        bank=new SavingsAccount("s87964", 1000.0, 3.5);
        bank.displayAccountDetails();
        bank.calculateInterest();
        bank.withdrawn(200);
        bank.displayAccountDetails();
    }
}
```

Abstract rules:

Rule-1:

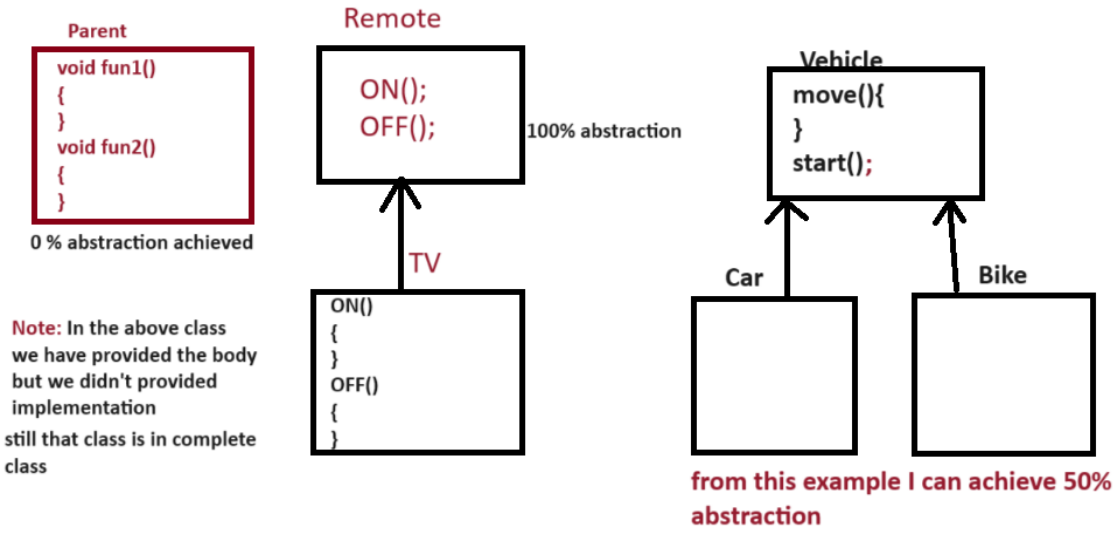
If a method is made as abstract then compulsory then child class should override the method.

If the child class doesn't override the abstract method then error is dispalyed during compilation time.

Rule-2:

If a child class is not in a position to override the abstract method then in the child class.

i.e: redeclaring the abstract and corresponding class should also be made as abstract.



Rule-3: Inside the abstract class we can write all type of variables, constructor , static methods,concrete methods,final methods and abstract methods.

Rule-4: illegal combination with abstract

1.static

2.final
3.private
4.strictFp
5.synchronized
6.native

Interface:

->Interface is an specification
->Interface promotes standardization using which polymorphism can be achieved.
->Interface is a collection abstract methods and constants.
->Using interface we can achieve 100% abstraction.

syntax :

```
interface <interface-name>
{
    // abstract methods
    // constants
}
```

Note: by default all the abstract methods in interface are "public abstract".
->By default all the variables are "final static".

Advantages of interfaces:

1.Interface promotes standardization

example for standardization is Traffic lights

2.Interface promotes polymorphism

3.100% abstraction is achieved.

4.multiple inheritance achieved

Rules of interface:

1.when a class is implementing the interface the methods access modifier should be public,
because by default all the abstract methods in interface are public in nature.

2.If a class is implements interface then it should provide the body for all the methods present in interface.
if the class is not in a position to provide the body for all methods present in in interface then my implementation class will become as abstract class.

3.An object of interface can't be created but reference can be created.

4.An class and interface are related using implements keyword

5.using interface reference implemented methods of the classes can be accessed and
interface reference polymorphismcan be achieved.

6.using interface reference only over-ridden methods can be accessed.In order to access the specialized methods downcasting has to performed.

7.A class can implements any no.of interfaces

8.if 2 interfaces contains same method signature and same return type then implementation class can provide the body for only 1 method.

9.if 2 interfaces contains same method name,same return type and changing arguments then the implementation class should provide the body for all the methods.

10.if 2 interfaces contains same method signature but return type is different then implementation class can't provide the body.

11.An interface extends any no.of interfaces

12.An class implements an interface, interface extends another inetrface then the implementation class should provide the body for all the methods.

13.An interface can't implement the another interface.
A class can extends another class simultaneously we can implement interface also.In this case the implementing class should extends 1st followed by implements.

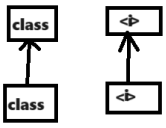
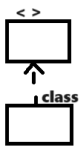
Q)Differences between interface and abstract class?

A:

interface	Abstract class
-> 100% abstraction can be achieved	->0 to 100% abstraction can be achieved
->interfaces are used when we don't know complete implementation	-> abstract class is used when we know partial implementation
->only abstract methods are allowed	->both abstract and concrete methods are allowed
->Every methods inside interfaces are by default public abstract	->Every method present inside abstract class need not to be the public abstract
->Every variable inside the interfaces are by default public static final	->Every variable present in abstract class need not to be the public static final
->following modifiers are not allowed for abstract methods are static,private,strictfp,protected ,final	->No such restrictions for methods
->interface cannot have static blocks ->interface cannot have instance block ->constructors are not allowed	->abstract class can have static block ->abstract class can have instance block ->constructors are allowed

Q)Differences between extends and implements?

A:

Extends	implements
	
A class can inherit other class using extends keyword An interface can inherit another interface by using extends keyword	A class can be implements other interface by using implements keyword
A subclass which extends super class may or may not over-ride all methods of super class	class implementing interface should provide body for all the methods of interface
An interface can extends any no.of interfaces	An interface can never implements another interface
A class can extends only one super class	A class can implements multiple interfaces simultaneously

Q)can we overload the main method?

A: yes, we can overload the main method.

```
public class App01 {
    public static void main(int args) {
        System.out.println("Integer value is:
"+args);
    }
    public static void main(float args) {
       System.out.println("float value is:
"+args);
    }
    public static void main(double args) {
        System.out.println("double value is:
"+args);
    }
    public static void main(String[] args) {
        App01.main(10);
        App01.main(12.87f);
        App01.main(123.87);
    }
}
```

Q)can we override the main method()?

A:No, we can't override the main method.

*** ALL THE BEST ***