JAVA BASICS

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.
In java, we use method overloading and method overriding to achieve polymorphism.
Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.
In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.
A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

CODE 1:

```
public class Main {
   public static void main(String args[]) {
      System.out.println("main");
      Main m = new Main();
   }
   static{
      System.out.println("static");
   }
```

```
    {
        System.out.println("class initializers");
    }
    Main(){
        System.out.println("constructor");
    }
}
```

Output:
static
main
class initializers
constructor

CODE 2:

```
public class Main {
    public static void main(String args[]) {
        System.out.println("main");
    }
    static{
        System.out.println("static");
    }
    {
        System.out.println("class initializers");
    }
    Main(){
        System.out.println("constructor");
    }
}
```

Output:
static
main

CODE 3:

```
public class Main {
    static{
        System.out.println("static");
    }
    {
```

```
        System.out.println("class initializers");
    }
    Main(){
        System.out.println("constructor");
    }
}
```

Output:
Compilation error #stdin compilation error #stdout 0.08s 27688KB
spoj: The program compiled successfully, but main class was not found.
Main class should contain method: public static void main (String[] args).

CODE 4: "static" Keyword

(https://www.javatpoint.com/static-keyword-in-java)

Java static variable

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.

Java static method

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.
- The static method cannot use non static data member or call non-static method directly.
- this and super cannot be used in static context.

Java static block

- Is used to initialize the static data member.
- It is executed before main method at the time of class loading.

CODE 5: "this" keyword

this is used to call current class default constructor: this(); , parent class methods: this.method(); and parent class variables: this.var;

```
class ABC{
    int id; // by default 0

    ABC(int id){
        id = id;
```

```
    }
    void display()
    {
        System.out.println(id);
    }
}

class Rextester
{
    public static void main(String args[])
    {
        ABC a = new ABC(100);
        a.display();

    }
}
```

Output:

0

Solution:-

```
class ABC{
    int id;

    ABC(int id){
          this.id = id;
    }
    void display()
    {
        System.out.println(id);
    }
}

class Rextester
{
    public static void main(String args[])
    {
        ABC a = new ABC(200);
        a.display();
```

```
    }
}
```

Output:
200

CODE 6: Inheritance (IS-A relationship)

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

```
class Animal{
    void bark(){
        System.out.println("animal");
    }
}
class Dog extends Animal{
    void bark(){
        super.bark();
        System.out.println("dog");
    }
}
class Codechef
{
        public static void main (String[] args) throws java.lang.Exception
        {
                Animal d=new Dog();  // or Dog d = new Dog();
                d.bark();
        }
}
```

Output:
animal
dog

CODE 7: Inheritance "super" keyword

Super is used to call parent class constructors: super(); , parent class methods: super.method(); and parent class variables: super.var;

```
class Animal{
```

```
   Animal(){
      System.out.println("animal");
   }
}

class Dog extends Animal{
   Dog(){
        //super(); is called by automatically….whether we write or not.
      System.out.println("dog");
   }
}


class Rextester
{
   public static void main(String args[])
   {
     Animal a = new Animal();                  //animal
     Dog d = new Dog();                 //animal dog
       Animal a1 = new Dog();                  //animal dog
     Dog d1 = new Animal();               //error: Animal cannot be converted to Dog
   }
}
```

CODE 8: Multiple Inheritance is not possible in java

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 Public Static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Output:

Compile time error

CODE 9: Aggregation (HAS-A relationship)

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

```
public class Address {
String city,state,country;

public Address(String city, String state, String country) {
   this.city = city;
   this.state = state;
   this.country = country;
}
}

public class Emp {
int id;
String name;
Address address;

public Emp(int id, String name,Address address) {
   this.id = id;
   this.name = name;
   this.address=address;
}

void display(){
System.out.println(id+" "+name);
System.out.println(address.city+" "+address.state+" "+address.country);
}

public static void main(String[] args) {
Address address1=new Address("gzb","UP","india");
Address address2=new Address("gno","UP","india");

Emp e=new Emp(111,"varun",address1);
Emp e2=new Emp(112,"arun",address2);

e.display();
```

```
e2.display();

}
}
```

Output:

```
111 varun
gzb UP india
112 arun
gno UP india
```

Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

CODE 10: Abstract Class

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

- An abstract class can have data member, abstract method, method body, constructor and even main() method.
- If there is any **abstract method** in a class, **that class must be abstract**.
- If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or **make this class abstract.**
- **A class can be defined as abstract without any abstract method(can't be instantiated). But an abstract method cannot be defined as abstract only if the class is abstract.**

```
abstract class Bike{  // class must be defined abstract as it contains abstract method
  Bike(){System.out.println("bike is created");}
   abstract void run();
  void changeGear(){System.out.println("gear changed");}
}

 class Honda extends Bike{
void run(){System.out.println("running safely..");}
}
class TestAbstraction2{
```

```
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

Output:

bike is created
running safely..
gear changed

```
abstract class ABC{
   abstract void display();
}
class ABC_child extends ABC{
  void display()
   {
      System.out.println(100);
   }
}

class Rextester
```

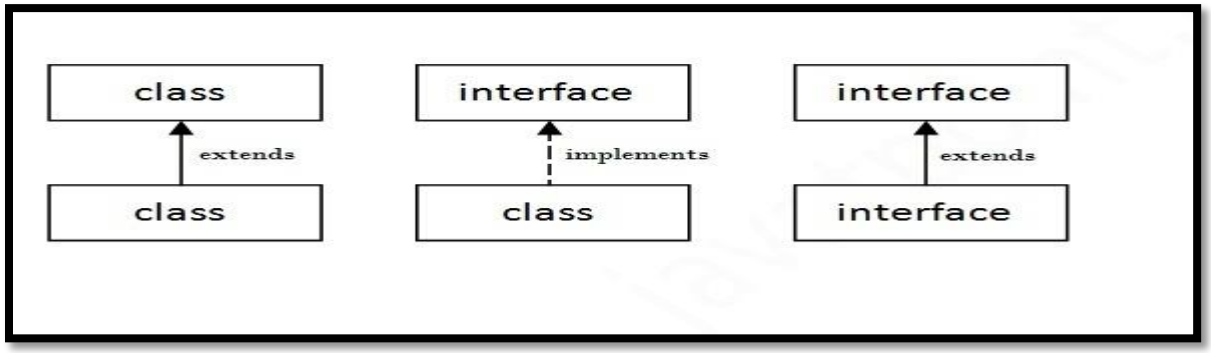
```
   public static void main(String args[])
   {
      ABC aaaaaa = new ABC();                  // error: ABC is abstract; cannot be instantiated
      ABC a1 = new ABC_child();
      ABC_child a2= new ABC_child();

       a1.display();                           //100
       a2.display();                           //100

   }
}
```

CODE 11: Interface (IS-A relationship)

An interface in java is a blueprint of a class. It has static constants and abstract methods. It cannot be instantiated just like abstract class. It is used to implement "multiple inheritance in java".

- The java compiler adds public and abstract keywords before the interface method. More, it adds public, static and final keywords before data members.

- **Interface is a collection of abstract method only (no normal methods).** When we inherit the interface then we will have to define and override all the abstract method in that class.
- Keyword **"implements"** is used.
- Interface does not have a constructor.




```
interface Printable{
void print();
}

interface Showable{
void show();
}

class A7 implements Printable, Showable{ // we will have to define all the methods of interface otherwise the class A7 should be defined as abstract class……
public void print(){
System.out.println("Hello");
}
public void show(){
System.out.println("Welcome");
}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Output:

Hello
Welcome

Interface adds public, static and final keywords before data members and variables.

```
interface Showable{
int i = 10;
void show();
}

class A7 implements Showable{
public void show(){
i = i +10;      // error: cannot assign a value to final variable i
System.out.println("Welcome");
}
}

class Rextester
{
   public static void main(String args[])
   {
      A7 obj = new A7();
       obj.show();

   }
}
```

CODE 12: Multiple inheritance is not supported through class in java but it is possible by "interface"

```
interface Printable{
int i = 10;
void print();
}
interface Showable{
int i = 1000;
void print();
}

class TestInterface3 implements Printable, Showable{
public void print(){//there will be no ambiguity for method print() as it not defined in both parent classes
System.out.println("Hello");
```

```
System.out.println(Printable.i); // To access the variables we use: <interface_name>.<variable_name>
System.out.println(Showable.i);
}
public static void main(String args[]){
TestInterface3 obj = new TestInterface3();
obj.print();
 }
}
```

Output:

```
Hello
10
1000
```

CODE 13: Marker or Tagged Interface

An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

```
public interface Serializable{
}
```

Exception Handling