ASSIGNMENT - 02.

Q1. Compare Concrete Class, Abstract class & Interface

CONCRETE CLASS

→ Has data members and member method
→ All member methods are defined
→ Has constructor
→ Can be ~~no~~ instantiated using new operator and constructor method
→ The keyword "extends" is used to inherit a concrete class into another class
→ Any access modifier can be associated with any member.

ABSTRACT CLASS

→ Has data members and member method
→ The keyword "abstract" has to be used along with the class declaration
→ All or some of the member methods may be defined.
→ Member method without definition body should be declared as abstract.
→ If all the member methods are defined in the class declared as abstract, then the inherited child class does not become abstract class
→ Cannot be instantiated using new operator and constructor methods.
→ The keyword "extends" is used to inherit an abstract class.
→ The child class can use the super() to

```
public class NewClass {
    public static void main (String[] arg) {
        I ob1;
        ob1 = new A();
        ob2. f2();
```

CONCERETE CLASS	ABSTRACT	INTERFACE
Has constructor	Has constructor	Does not have constructor
Can be instantiated using new operater and constructor method	Cannot be instantiated using new operators & constructor method.	Cannot instantiate an interface
Any access modifier can be associated with any member (data as well as methods)	There can be a mix of abstract and concrete methods with different access modifiers. No restriction on data members.	All member methods are compulsorily public abstract. data members are public static & final
Static and instance block are allowed	Static and instance block are allowed	No static and instance blocks are allowed
To inherit, extends key-word is used	To inherit, extends keyword is used	To inherit into class implements keyword is used.

OUTPUT

```
Default
100
```

ABSTRACT CLASS

```
abstract class A
{ .
  private int n;
  public abstract void setN(int n);
  public void getN() { System.Out.println("n="+n);}
  public A() {
    System.out.println("Default Constr.");
    n=0; }
 }
public class MyClass {
  public static void main (String args[]){
    { //A ob1 = new A(); // ERROR :
   } // ob1.getN();              A is abstract class
                                 cannot be instantiated
  }
 }
```

INTERFACE ~~CLASS~~

```
interface I {
   int  n = 100
     void f2();
   }
Class A implements I {
     public void f2() { System.out.println(
       "Class-A f2()");* }
 }
```

call the abstract parent class constructor.

INTERFACE CLASS

→ An interface is a collection of abstract method. A class implements an interface thereby inheriting the abstract methods of the interface.

→ An interface is not a concrete class. writing a class but an interface contains behaviours ~~of an object~~ that its sub-class implements.

→ An ~~st~~ interface is implicitly abstract.

→ Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.

CONCRETE CLASS

```
class A {
      private int n;
      public void setN(int n) { this.n=n;}
      public void getN() {System.out.println("n="+n);}
      public A() { System.out.println("Default");
       n=0; }
    }
 }
public class MyClass {
      public static void main(String args[]){
      A ob1 = new A();
      ob1.setN(100);
      ob1.getN();
    }
  }
```

Q2. Explain with proper example the concept of Singleton classes in Java.

→ when we are allowed to create only a single instance of class then that class is called singleton class.

→ when several processes having the same requirement then a single object-instance can be created that can be used by all the processes, here singleton class is useful.

→ Memory utilization improves, as many object instances are not created but a single instance is reused.

→ Here, constructor cannot be used to create instance outside the class, instead factory method are used.

→ A factory method is a member method of singleton class that returns the same existing instance of the singleton class

→ Every constructor is declared private so that any instance cannot be created from outside the class.

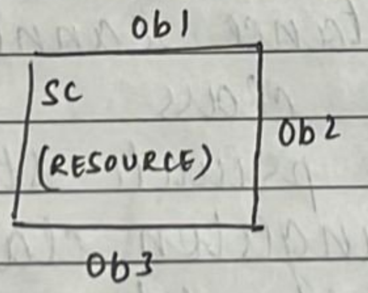
→ There has to be private static instance of the same singleton class as data-member and it should be initialized by any ~~pv~~ of the private constructors, usually the default constructors is used~~ally the~~.

→ There has to be a public static factory method. that should return the same static singleton class instance declared as the data member in the class.

→ This factory method will be used to get the singleton class instance

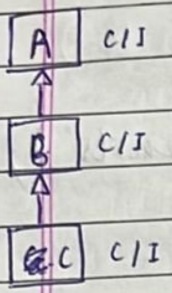
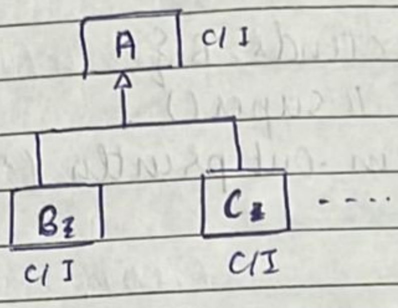
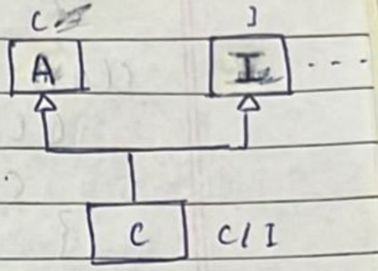
~~class~~

```
Class sc {
    private static SC ob1 = new.SC();
    private SC() { }
    public static SC getInstance() { return ob1; }
}
public class NewClass {
  public static void main (String [] arg) {
      // sc ob1 = new SC();
    SC ob1 = SC.getInstance();
    SC ob2 = SC.get Instance();
    SC ob3 = SC.get Instance ();
   System.out.println ("ob1 =" + ob1 +"ob2 =" + ob2+
                           "ob3 =" + ob3);
  }
}
```



RESOURCE SHARING.

Q3. compare Multilevel, Hierarchical & Multiple Inheritance.

MULTILEVEL	HIERARCHICAL	MULTIPLE
		

class A
class B extends A
class C extends B
	class A
	class B extends A
	class C extends A
		class A
		Interface ~~B~~ I
		Class C extends A implements I

It involves of atleast two or mor than two classes. One inherits the features of parent class & the newly created sub-class becomes the base class for another new class.	It consist of one parent class and other child class inherit the properties of parent class.	It consist of one child class which inherit properties of multiple parent class.

CODE. : MULTILEVEL (DEFAULT)

```
Class A {
```

```
A () { System.out.println ("Class A consts"); }
}
Class B extends A {
   B() { System.out.println (" Class B consts"); }
  }
class C extends B {
   C() { // super()
     System.out.println ("Class C consts"); }
  }
  }
public class NewClass {
   public static void main( String [] arg){
      C b1 = new C ();
```

(PARAMETERISED)

```
Class A {
   A (int n) { System.out.println ("ClassA consts"); }
  }
class B extends A {
   B(int n) {
      super (100);
     System.out.println ("Class B-consts");
    }
  }
Class C extends B {
   C() {
     super(200);
     System.out.println ("Class C- consts");
  }
}
```

```
public class NewClass {
      public static void main ( String[]arg) {
            Cb1 = newC();
      }
```

OUTPUT [DEFAULT]

```
class B consts
Class C consts
```

[PARAMETERISED]

```
Class B consts
Class C consts.
```

HIERARCHICAL

DEFAULT

```
class A {
     A() { System.out.println ("Class A consts"); }
  }
  class B extands A {
       B() { System.out.println ("Class B consts"); }
   }
class C extends A {
       C() { System.out.println ("Class C consts"); }
  }
public class NewClass {
      public static void main ( String[] arg) {
           B ob1 = new B();
           C ob2 = new C();
    }
}
```

OUTPUT

```
class A consts
class B consts
class A consts
class C consts.
class A {
    A ( int n) { System. out. println ("class A consts");}
      }
  class B extends A
     B(int n){ super(100);
      System.out. println ("Class B consts");
     }
   }
  class C extends A
      C (int n){ super(200);
      system.out. println ("Class C consts"); }
    }
  public class NewClass{
  public static void main (string[] arg) {
          B ob1 = new B();
          C ob2 = new C();
    }
   }
```

OUTPUT

```
class A consts
class B consts
class A consts
Class C consts.
```

MULTIPLE (DEFAULT)

```
class A {
  A () { System.out.println ("Class A consts"); }
   }
 interface B {
   }
```

```
class C extends A implements B {
    C() { System.out.println("class C constr");}
   }
public class Newclass {
    public static void main( String[ ]arg ){
            C ob2 = newC();
    }
  }
```

OUTPUT

```
Class A constr
Class B consts
Class C constr.
```

(PARAMETERISED)

```
Class A {
    A(int n){ System.out.println("Class A constr"); }
    }
   interface B {
   }
Class C extends A implements B {
    C(int n){ system.out.println(" Class C constr");}
    }
public class NewClass {
  public static void main( string [ ] arg){
            C ob2 = newC();
      }
   }
```

OUTPUT

```
Class A consts.
Class B consts
class C consts
```