# CE251::INHERITANCE

**Find Output of the following with step by step execution explanation.**

**1.**

```java
package inheritance.Dispatch;
class Person
{
   private String name;
         public  Person(String  name){this.name  =
name;}
   public boolean isAsleep(int hr){
      return (hr>22 || hr<7);
   }
   public String toString(){ return name;}
   public void status(int hr){
      if(this.isAsleep(hr))
         System.out.println("Now offline:"+ this);
      else
         System.out.println("Now online:"+ this);
   }
}
```

```java
class Student extends Person
{
   public Student(String name)
   {
      super(name);
   }
   @Override
   public boolean isAsleep(int hr){
      return (hr>2 && hr<8);
   }
}
```

```java
public class Polymorphism1 {
   public static void main(String[] args) {
      Person p;
      p = new Student("Kavya");
      p.status(1);
      p = new Person("Vishwa");
      p.status(23);

   }
}
```

**Output :**

```
Now online:Kavya
Now offline:Vishwa
```

**Explanation:** p is an object of super class Person and the object calls the derived class Student of super class Person.The method status in Person class is called by the object.

**2.**

```java
package inheritance.covarient;
class Person{
   public void method1(){
      System.out.println("Person 1");
   }
   public void method2(){
      System.out.println("Person 2");
   }
}
```

```java
class Student extends Person{
   public void method1(){
      System.out.println("Student 1");
      super.method1();
      method2();
      this.method2();
   }
   public void method2(){
      System.out.println("Student 2");
   }
}
```

```java
class Undergraduate extends Student{
   public void method2(){
      System.out.println("Undergraduate 2");
   }
}
```

```java
public class Polymorphism2 {
   public static void main(String[] args) {
      Person u = new Undergraduate();
      u.method1();
   }
}
```

**Output:**

```
Student 1

Person 1

Undergraduate 2

Undergraduate 2
```

**Explanation:** The object of Person class is created and Person class is the super class and the extended class of Person is Student and the extended class of Student is Undergraduate. The method1() is called which is in Student class is called because there is no method1() in Undergraduate class.

## 3. Explain the concept of casting of object. How an instanceof operator can be used to provide run-time check of “is-a” relationship? Explain with example.

All casting really means is taking an Object of one particular type and “turning it into” another Object type. This process is called casting a variable.

The instanceof operator also called type comparison operators used to check if an object is of a particular class or interface type. The instanceof operator is used for object references only to test the class of an object. The instanceof operator works on the principle of IS~A test. In Object Oriented Programming, the concept of IS-A is based on class inheritance or interface implementation. IS-A is a way of saying, "this object is a type of that class." We express the IS-A relationship in Java through the keywords extends (for class inheritance) and implements (for interface implementation).

The instanceof operator has the following syntax:

**object instanceof type**

The instanceof operator evaluates to true if the expression on its left is a reference type that is assignment compatible with the type name on its right, and false otherwise. Term "assignment compatible" means that we can test an object reference against its own class type, or any of its superclasses. Hence, any object reference will evaluate to true if we use the instanceof operator against type Object because in Java every class implicitly inherits Object class. It is important to note that null is not an instance of any type; therefore, instanceof for null always returns false.

**Example**

```
class A {}

class B extends A {}

class C extends A {}

public class InstanceofDemo

{

 public static void main(String args[])

 {

  A a = new A();

  B b = new B();

  C c = new C();

  System.out.println("a instanceof A: " + (a instanceof A));

  System.out.println("b instanceof A: " + (b instanceof A));

  System.out.println("c instanceof A: " + (c instanceof A));

  System.out.println("a instanceof B: " + (a instanceof B));

  System.out.println("null instanceof A: " + (null instanceof A));
 }
}
```

**OUTPUT**

```
a instanceof A: true
b instanceof A: true
c instanceof A: true
a instanceof B: false
null instanceof A: false
```

## 4. Method overloading vs Method overriding with example

Overloading occurs when two or more methods in one class have the same method name but different parameters.

Overriding means having two methods with the same method name and parameters (i.e., method signature). One of the methods is in the parent class and the other is in the child class. Overriding allows a child class to provide a specific implementation of a method that is already provided its parent class.

The real object type in the run-time, not the reference variable's type, determines which overridden method is used at runtime. In contrast, reference type determines which overloaded method will be used at compile time.

Polymorphism applies to overriding, not to overloading.

example of overloading

```
  public void bark(){
      System.out.println("woof ");
   }
    public void bark(int num){
         for(int i=0; i<num; i++)
                  System.out.println("woof ");
   }
}
```

example of overriding

```
class Dog{
   public void bark(){
       System.out.println("woof ");
   }
}
class Hound extends Dog{
   public void sniff(){
       System.out.println("sniff ");
   }
   public void bark(){
       System.out.println("bowl");
   }
}
public class OverridingTest{
   public static void main(String [] args){
       Dog dog = new Hound();
       dog.bark();
   }
}
```

**5. Consider following scenario. Write a program to use single data structure to point multiple object of that type and test your result.**

**Hint:**

**Person p[] = new Person[3];**

**P[0] = new Person(“Dhara”);**

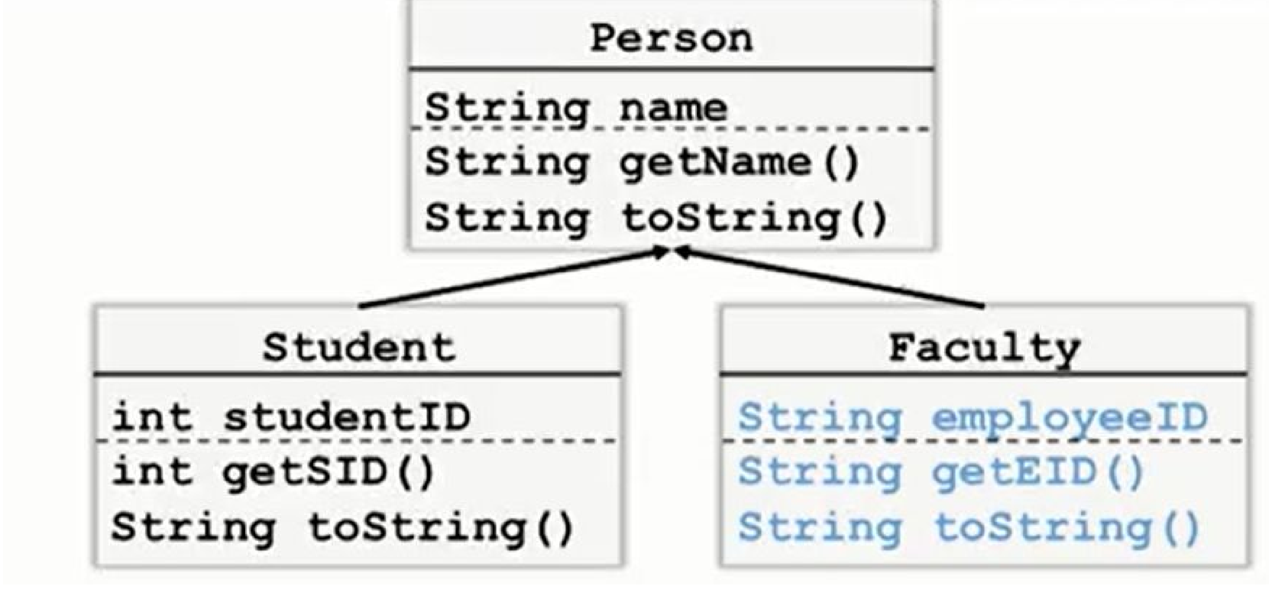
**P[1] = new Student(“Lara”, 16CE007);**

**P[2] = new Faculty(“Swara”, “E1004”);**

**Call toString() methods for all types in a loop and check output.**

**for(int i=0;i<p.length; i++)**

**{ System.out.print(p[i]); }**



```
class Person
{
        String name;
Person(String s)
{
        name = s;
}
public String getname() {
return name;
}
        public String toString() {
        return name;
        }
}
class Student extends Person {
String sname, ID;
Student(String name, String t) {
        super(name);
ID = t;
}
String getSId() {
return ID ;
}
public String toString() {
        return super.name + " " + this.ID;
```

```
            }
}
            class Faculty extends Person {
String namef, IDf;
Faculty(String name, String t) {
            super(name);
            IDf = t;
}
String getId() {
return IDf;
}
public String toString() {
            return super.name + " " + this.IDf;
            }
            }
            public class Main{
public static void main(String[] args){
Person p[] = new Person[3];
p[0] = new Person("Dhara") ;
p[1] = new Student("Lara","18ce190") ;
p[2] = new Faculty("Swara","E1004") ;
for(int i=0;i<p.length;i++)
{System.out.println(p[i]);
p[i].toString();
}
}
}
```