JAVA : INHERITANCE

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

```
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);
   }
}
```

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

```
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:

In this practical Single level inheritence is used and the parent class reference is used to refer the object of both parent class object and child class object and toString() method is overridden in parent class

So method of object class will be hidden and parent class method is called.

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:

In this code parents class reference variable is refer to Undergraduate class object and then we are calling method 1 by using child class object so method 1 is not available in child class then it is calling method 1 of its parent class and in method 1 defination we are calling Further methods of parent class using super keyword and by using this keyword we are calling current class method…

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

What is Casting in Java?

Well, 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.
This topic is not specific to Java, as many other programming languages support casting of their variable types. But, as with other languages, in Java you cannot cast any variable to any random type.

What are the Rules behind Casting Variables?

- We know about how everything in Java is an Object, and any Object you create extends from Object? This was inheritance, and this is important to understand when dealing with casting.
- If you are going to cast a variable, you’re most likely doing what’s known as a downcast. This means that you’re taking the Object and casting it into a more “specific” type of Object. Here’s an example:

-
  ```
  Object aSentenceObject = "This is just a regular sentence";
  String aSentenceString = (String)aSentenceObject;
  ```
- What we’ve done here? Since the type Object is a very broad type for a variable, we are “downcasting” the variable to be a String type. Now, the question is, is this legal? Well, we can see that the value stored in the aSentenceObject is just a plain old String, so this cast is perfectly legal. Let’s look at the reverse scenario, an “upcast”:

-
  ```
  String aSentenceString = "This is just another regular sentence";
  Object aSentenceObject = (Object)aSentenceString;
  ```
- Here we are taking a variable with a more specific type (String) and casting it to a variable type that’s more generic (Object). This is also legal, and really, it is **always legal to upcast**.

What are the benefits of casting objects :

- Upcasting is done automatically in Java ! say for example , Car is a superclass and Porsche and Ford are subclasses .If you want to make a object of the Porsche class ,you just have to write:
  ```
  Car c=new Porsche();[Upcasting is done automatically]
  but :
  Porsche p=new Car();
  ```
  will throw an exception! you have to explicitly downcast here:
  ```
  Porsche p=(Porsche)new Car();
  ```
- will be the way to go. This is down casting and you have to do it manually! It's because All Porsche models are of the type car, but all objects of the class Car may not be Porsche objects!
  Java supports both. To master these things, pick little examples like this from real life and try to implement them in your program.

See the image attached! Object class is at the top of the Hierarchy . Java extends every class from Object class by default.

What are the Downsides of casting :

- Well, there is a certain amount of risk that goes along with downcasting your variables. If you were to try to cast something like a Date object to an Integer object, then you'll get a ClassCastException. This is what's known as a run-time exception, as it's really only detectable when your code is running. So, unless you're doing something with error handling, then your program will likely exit or you'll get an ugly error message on your webpage.
- So just make sure that if you are doing any downcasting, that you're well aware of the type of object you'll be casting.

4. Method overloading vs Method overriding with example.

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class.*	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

Java Method Overloading example

```
class OverloadingExample
{
static int add(int a,int b) {return a+b;}
 static int add(int a,int b,int c) {return a+b+c;}
 }
```

Java Method Overriding example

```
class Animal
{
     void eat(){System.out.println("eating...");}
}
class Dog extends Animal
{
void eat(){System.out.println("eating bread...");}
}
```

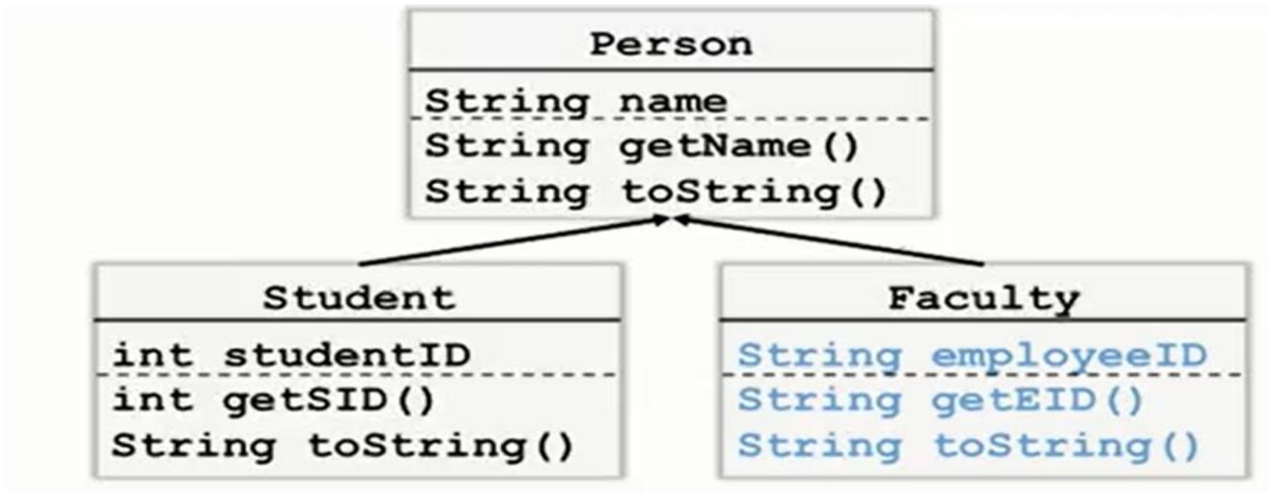
5. Consider following scenario. Write a program to use single data structure to point multiple object of that type and test your result.

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]); }
```



Program:

```
package test2;
class Person
{
   public String name;
   Person(String d)
   {
      name=d;
   }
   @Override
   public String toString()
   {
      return "name :"+name;
   }
```

```
}
class Student extends Person
{
    String studentID;
     public Student(String a,String id)
     {
         super(a);
         studentID=id;
     }

      @Override
    public String toString()
    {
        return "name :"+name+"\n"+"studentid :"+studentID;
    }
}
class Faculty extends Person
{
    String facultyID;
    public Faculty(String b,String id)
    {


        super(b);
        facultyID=id;


    }

     @Override
    public String toString()
    {
        return "name :"+name+"\n"+"facultyid :"+facultyID;
    }
}
    public class Test2 {
    public static void main(String[] args) {

        Person p[] = new Person[3];
        p[0] = new Person("Dhara");
        p[1] = new Student("Lara", "16CE007");
        p[2] = new Faculty("Swara", "E1004");

        for(int i=0;i<p.length; i++)
    {
        System.out.print(p[i]);
        System.out.println("\n");
    }
    }
}
```

Output:

```
name :Dhara
name :Lara
studentid :16CE007
name :Swara
facultyid :E1004
```