Exercise 01:

Create a class called "Employee" which has 3 private variables (empID, empName, empDesignation) and create getters and setters for each field. Please note that this has no main method since this is just a blueprint not a application. Now crate a test class to invoke the Employee class. Create two objects for Mr.Bogdan and Ms.Bird and set required values using setters and print them back on the console using0 getters.

```
//Create class Employee
public class Employee
{
    //Adding variables
    private int empID;
    private String empName,empDesignation;
    //Getters
    public int getEmpID()
    {
        return empID;
    }
    public String getEmpName()
    {
        return empName;
    }
    public String getEmpDesignation()
    {
        return empDesignation;
    }
    //Setters
    public void setEmpID(int empID)
    {
        this.empID=empID;
    }
    public void setEmpName(String empName)
    {
        this.empName=empName;
    }
    public void setEmpDesignation(String empDesignation)
    {
        this.empDesignation=empDesignation;
    }

    }
```

```
public class Test {
    public static void main(String[] args)
    {
        Employee MrBogdan=new Employee();
        Employee MsBird=new Employee();

        MrBogdan.setEmpID(1);
        MrBogdan.setEmpName("Mr.Bogdan");
        MrBogdan.setEmpDesignation("Manager");

        MsBird.setEmpID(2);
        MsBird.setEmpName("Ms.Bird");
        MsBird.setEmpDesignation("Secretary");

        System.out.println("Employee ID: "+MrBogdan.getEmpID());
        System.out.println("Employee Name:
"+MrBogdan.getEmpName());
        System.out.println("Employee Designation:
"+MrBogdan.getEmpDesignation());

        System.out.println("\nEmployee ID: "+MsBird.getEmpID());
        System.out.println("Employee Name:
"+MsBird.getEmpName());
        System.out.println("Employee Designation:
"+MsBird.getEmpDesignation());
    }
}
```

Exercise 02:

Develop the following class execute and discuss the answer: Please note that each class stored in separate files. Write down the answer.

```
class SuperB {
   int x;
   void setIt (int n) { x=n;}
   void increase () { x=x+1;}
   void triple () {x=x*3;};
   int returnIt () {return x;}
}
class SubC extends SuperB {
   void triple () {x=x+3;} // override existing method
   void quadruple () {x=x*4;} // new method
}
public class TestInheritance {
   public static void main(String[] args) {
      SuperB b = new SuperB();
      b.setIt(2);
      b.increase();
      b.triple();
      System.out.println( b.returnIt() );
      SubC c = new SubC();
      c.setIt(2);
      c.increase();
```

```
        c.triple();
        System.out.println( c.returnIt() );
        }
  }
```

-Answer-

9

6

SuperB b = new SuperB();

* Creating a class called SuperB and its x value is set to 2.

b.increase();

* The increase() method is called on the SuperB instance and it will increase the x value by 1. So, x becomes 3 (2 + 1).

b.triple();

* The triple() method is called on the SuperB instance and it will multiply the x value by 3. So, x becomes 9 (3 * 3).

System.out.println(b.returnIt());

* The return () method is called on the SuperB instance, and the current value of x (which is 9) is printed to the console. So, the output is "9".

SubC c = new SubC();

* Creating a new instance of the SubC class. Since SubC extends SuperB, it inherits all the attributes and methods of the SuperB class. The x value for this instance is set to 2.

c.increase();

* The increase() method is called on the SubC instance. This method increases the x value by 1. So, x becomes 3 (2 + 1).

c.triple();

* The triple() method is called on the SubC instance. However, SubC overrides the triple() method from the SuperB class. Instead of multiplying x by 3, it adds 3 to the x value. So, x becomes 6 (3 + 3).

System.out.println(c.returnIt());

* The returnIt() method is called on the SubC instance, and the current value of x (which is 6) is printed to the console. So, the output is "6".

Exercise 03:

Recall the following scenario discussed during the class. Develop a code base to represent the scenario. Add a test class to invoke Lecturer and Student class by creating atleast one object from each.

Note: All the common attributes and behavior stored in the super class and only the specific fields and behavior stored in subclasses.

Student
- name
- id
- course
+ setName()/getName()
+ setID()/getID()
+ setCourse()/getCourse()

Lecturer
- name
- id
- programme
+ setName()/getName()
+ setID()/getID()
+ setProg()/getProg()

Person
Identify field and attributes to be stored in this class

```
public class Person
{
    private String name;
    private int id;

    //Constructor
    public Person(String name, int id)
    {
        this.name=name;
        this.id=id;
    }

    //Getters
    public String getName() {
        return name;
    }
    public int getId() {
        return id;
    }

    //Setters
    public void setName(String name) {
        this.name = name;
    }

    public void setId(int id) {
        this.id = id;
    }
}
public class Student extends Person {
    private String course;
```

```
    public Student(String name,int id,String course) {
        super(name,id);
        this.course=course;
    }
    public void setCourse(String course) {
        this.course = course;
    }
    public String getCourse()
    {
        return course;
    }
}
```

```
public class Lecture extends Person
{
    private String programme;

    //Constructors
    public Lecture(String name,int id,String programme)
    {
        super(name,id);
        this.programme=programme;
    }

    //Getters
    public String getProgramme() {
        return programme;
    }
    public void setProgramme(String programme) {
        this.programme = programme;
    }
}
```

```
public class Test {
    public static void main(String[] args)
    {
        Student s1=new Student("kavinda",123,"MIS");
        System.out.println("Student Name: "+s1.getName());
        System.out.println("Student ID: "+s1.getId());
        System.out.println("Student Course: "+s1.getCourse());

        Lecture l1=new Lecture("John",101,"Java");
        System.out.println("\nLecture Name: "+l1.getName());
        System.out.println("Lecture ID: "+l1.getId());
        System.out.println("Lecture Programme: "+l1.getProgramme());
    }
}
```

Exercise 04

Develop the following class execute and discuss the answer: Please note that each public class stored in separate files. Write down the answer.

```
public class Animal{}

public class Mammal extends Animal{}

public class Reptile extends Animal{}

public class Dog extends Mammal{
  public static void main(String args[]){
    Animal a = new Animal();
    Mammal m = new Mammal();
    Dog d = new Dog();
    System.out.println(m instanceof Animal);
    System.out.println(d instanceof Mammal);
    System.out.println(d instanceof Animal);
  }
}
```

-Answer-

```
true
true
true
```

- m instanceof Animal:- m is an object of the Mammal class, which is a subclass of Animal. Since Mammal extends Animal, the m object is also an instance of Animal, so it returns true.

- d instanceof Mammal:- The d object is of type Dog, which is a subclass of Mammal. Since Dog is a Mammal, it returns true.

- d instanceof Animal:- Since Dog is a subclass of Mammal, and Mammal is a subclass of Animal, the d object is also an instance of Animal, so it returns true.

- All the instanceof checks return true because the inheritance hierarchy allows objects of derived classes to be treated as instances of their base class.