LAB1
(Creating Derived Classes)

Statement: We have the following class:

```
public class Point {
   private int x, y;
   public void initialize(int x, int y) {  this.x = x;     this.y = y;    }
   public void move(int dx, int dy) {        x += dx;        y += dy;    }
   public int getX() {        return x;    }
   public int getY() {        return y;    }}
```

1. Create a project named **LAB1_Inheritance**.
2. Create a class **PointA**, derived from the **Point** class, with a method display that displays (in the console) the coordinates of a point.
3. Write a small test program using both **Point** and **PointA** classes.
4. What would happen if the **Point** class did not have the getX and getY methods?

Answer

2. It is sufficient to define a derived class using the keyword **extends**.

```
public class PointA extends Point {
   void display() {
      System.out.println("Coordinates: " + getX() + " " + getY());
   }
}
```

3. We can then create objects of type **PointA** and apply both the public methods of **PointA** and those of **Point**, as in this program accompanied by an example of execution:

```
public class TsPointA {
   public static void main(String[] args) {
      Point p = new Point();
      p.initialize(2, 5);
      System.out.println("Coordinates: " + p.getX() + " " + p.getY());

      PointA pa = new PointA();
      pa.initialize(1, 8); // Using the `initialize` method from Point
      pa.display(); // Using the `display` method from PointA
   }
}
```

Output on the console:

```
Coordinates: 2 5
Coordinates: 1 8
```

Note that a call such as p.display() would lead to a compilation error since the class of p (**Point**) does not have a display method. If the **Point** class did not have the access methods getX and getY, it would not have been possible to access its private fields x and y from the **PointA** class. Therefore, it would not have been possible to equip it with the display method. Inheritance does not allow bypassing the principle of encapsulation.

LAB2
(Encapsulation in Inheritance)

Statement: We have the following class:

```
public class Point {
   private int x, y;
   public void setPoint(int x, int y) {       this.x = x;       this.y = y;    }
   public void move(int dx, int dy) {       x += dx;       y += dy;    }
   public void displayCoordinates() {   System.out.println("Coordinates: " + x + " " + y); }}
```

1. Create a project named **LAB2_Inheritance**.
2. Create a class **PointName**, derived from **Point**, allowing the manipulation of points defined by two coordinates (int) and a name (char). The following methods should be provided:
   - setPointName to define the coordinates and name of a **PointName** object.
   - setName to define only the name of such an object.
   - displayCoordinatesName to display the coordinates and name of a **PointName** object.
3. Write a small program using the **PointName** class.

Answer

2. The derived class **PointName**:

```
public class PointName extends Point {
   private char name;
   public void setPointName(int x, int y, char name) { setPoint(x, y);
 // Using the `setPoint` method from Point
       this.name = name;    }
   public void setName(char name) { this.name = name;  }
   public void displayCoordinatesName() {    System.out.print("Point name: " + name + " ");
       displayCoordinates();
// Using the `displayCoordinates` method from Point
   }
}
```

3. The test class:

```
public class TsPointN {
   public static void main(String[] args) {
      Point p = new Point();
      p.setPoint(2, 5);
      p.displayCoordinates();
      PointName pn1 = new PointName();
      pn1.setPointName(1, 7, 'A'); // Method from PointName
      pn1.displayCoordinatesName(); // Method from PointName
      pn1.move(9, 3); // Method from Point
      pn1.displayCoordinatesName(); // Method from PointName
      PointName pn2 = new PointName();
      pn2.setPoint(4, 3); // Method from Point
      pn2.setName('B'); // Method from PointName
      pn2.displayCoordinatesName(); // Method from PointName
      pn2.displayCoordinates(); // Method from Point
   }
}
```

LAB3
(Inheritance and Constructor Call)

Statement: We have the following class (this time with a constructor):

```
public class Point {
   private int x, y;
   public Point(int x, int y) {       this.x = x;       this.y = y;    }
   public void displayCoordinates() { System.out.println("Coordinates: " + x + " " + y); }}
```

1. Create a project named **LAB3_Inheritance**.
2. Create a class **PointName**, derived from **Point**, allowing the manipulation of points defined by their coordinates (integers) and a name (char). The following methods should be provided:
   - A constructor to define the coordinates and name of a **PointName** object.
   - displayCoordinatesNom to display the coordinates and name of a **PointName** object.
3. Write a small program using the **PointName** class.

Answer

The derived class **PointName** must handle the construction of the entire corresponding object. A redefinition of the base class constructor is essential. Recall that the call to the base class constructor (using the keyword **super**) must be the first instruction in the derived class constructor.

```
public class PointName extends Point {
   private char name;
   public PointName(int x, int y, char name) {
      super(x, y); // Call to the base class constructor
      this.name = name;    }

   public void displayCoordinatesName() {
      System.out.print("Point name: " + name + " ");
      displayCoordinates(); // Using the `displayCoordinates` method from Point
   }}
```

3. A test program:

```
public class TsPointC {
   public static void main(String[] args) {
      PointName pn1 = new PointName(1, 7, 'A');
      pn1.displayCoordinatesName(); // Method from PointName
      PointName pn2 = new PointName(4, 3, 'B');
      pn2.displayCoordinatesName(); // Method from PointName
      pn2.displayCoordinates(); // Method from Point    }}
```

LAB4
(Inheritance and Constructor Call)

Statement: We will address cases of defining a constructor in a base class and how to call it in the derived class. You will create 4 LABs following the path below.

a. Case 1

1. Create a project named **LAB4_Inheritance_case1**.
2. Create the following two classes: **A** and **B**.

```
public class A { }

public class B extends A {    public B() {       super();    }}
```

3. Explain the errors.
4. Explain the difference between the constructors of classes **A** and **B**.
5. What is the effect on object creation for class **B**?

b. Case 2

1. Create a project named **LAB4_Inheritance_case2**.
2. Create the following three classes: **A**, **B**, and **lab4_inheritance_case2**.

```
public class A {
   public A() {} // Constructor 1
   public A(int n) {} // Constructor 2
}

public class B extends A {
   // No constructor defined
}

public class lab4_inheritance_case2 {
   public static void main(String[] args) {
      B b = new B();
   }
}
```

3. Explain the errors.
4. Explain the difference between the constructors of classes **A** and **B**.
5. What is the effect on object creation for class **B**?

c. Case 3

1. Create a project named **LAB4_Inheritance_case3**.
2. Create the following three classes: **A**, **B**, and **lab4_inheritance_case3**.

```
public class A {
   public A(int n) {} // Constructor 2
}

public class B extends A {
   // No constructor defined
}

public class lab4_inheritance_case3 {
   public static void main(String[] args) {
      B b = new B();
   }
```

```
}
```

3. Explain the errors.
4. What is the effect on object creation for class **B**?

d. Case 4

1. Create a project named **LAB4_Inheritance_case4**.
2. Create the following three classes: **A**, **B**, and **lab4_inheritance_case4**.

```
public class A {
   // No constructor defined
}

public class B extends A {
   // No constructor defined
}

public class tp_heritage_cas4 {
   public static void main(String[] args) {
      B b = new B();
   }
}
```

3. Explain the errors.
4. Explain the difference between the constructors of classes **A** and **B**.
5. What is the effect on object creation for class **B**?

Answers

Case 1

3. No errors.
4. Class **A** has an implicit default constructor. Class **B** defines an explicit constructor, which calls the constructor of the parent class using the super() instruction.
5. To create an object of class **B**, an object of class **A** is created by calling the implicit constructor.

Case 2

3. No errors.
4. Class **A** defines two constructors: one is the default (explicit) constructor, and the other is parameterized. Class **B** does not define any constructor, resulting in the presence of an implicit default constructor from the parent class **A**.
5. To create an object of class **B**, an object of class **A** is created by calling the first explicit constructor of class **A**.

Case 3

3. There is an error in class **B** because the parent class **A** does not define an implicit default constructor. Therefore, class **B** must define an explicit default constructor.
4. Objects of class **B** cannot be created.

Case 4

3. No errors.
4. Both constructors are implicitly defined by default, which implies that the default constructor of **B** calls the default constructor of **A**.
5. Creating an object of class **B** depends on creating an object of class **A**.

LAB5
(The Object Class)

Statement: Every class implicitly derives from the **Object** class. In this TP, we will create a class that models a student object and call the methods of the **Object** class (toString, equals, getClass, getName). In our case, a student is characterized by an ID and a name.

1. Create a project named **LAB5_Inheritance_classObject**.
2. Create the **Student** class with the following methods:
   - Two constructors: one with variable initialization and the other parameterized.
3. Create a test class named **lab5_inheritance_classObject**.
   - Show how to call the methods mentioned above from the parent class **Object**.

Answer

2. **Student** class:

```
public class Student {
   private int id;
   private String name;
   public Student() {        id = 1;        name = "salim";    }
   public Student(int id, String name) {        this.id = id;        this.name = name;    }
   @Override
   public String toString() {        return id + " " + name;    }}
```

3. Test class:

```
public class lab5_inheritance_classObject {
   public static void main(String[] args) {
      Student e1 = new Student();
      Student e2 = new Student(2, "fateh");
      System.out.println(e1.toString()); // Call to the `toString` method from Object
      System.out.println(e1.getClass()); // Call to the `getClass` method
      System.out.println(e1.getClass().getName()); // Call to `getClass` and `getName`
      e2 = e1; // e2 and e1 point to the same object
      if (e2.equals(e1)) {
         System.out.println("Two identical objects");
      } else {
         System.out.println("Two different objects");       }    }}
```

LAB6
(Type Casting: Implicit and Explicit)

Statement: The TP should contain two demonstrations: one for applying type casting between primitive types (**byte, int, short, long, float, double**) and the other between classes defined by us. In the second case, we will reuse the **Student** class from LAB5.

The rules for type casting between primitive types.

byte---→ short---→ int---→long---→float---→ double

I. Type Casting Between Primitives

1. Create a project named **LAB6_Inheritance_typeCasting**.
2. Create a test class named **lab6_inheritance_classObject** (contains the main method).
3. Insert the following instructions:

```
byte b1 = 15;
int i1 = 100;
long l1 = 2000;
float f1 = 2.5f; // 'f' for float; by default, it is of type double
double d1 = 2.5;
```

3.1. Insert the following instruction: b1 = b1 + 1;
- State your observation? Propose a correction?

3.2. Insert the following instruction: byte b2 = i1;
- State your observation? Propose a correction?

3.3. Insert the following instruction: long l2 = i1;
- State your observation? Justify?

3.4. Insert the following instruction: int i2 = l1;
- State your observation? Propose a correction?

3.5. Insert the following instruction: float f2 = l1;
- State your observation? Justify?

3.6. Insert the following instruction: long l3 = f1;
- State your observation? Propose a correction?

3.7. Insert the following instruction: double d2 = f1;
- State your observation? Justify?

3.8. Insert the following instruction: float f3 = d1;
- State your observation? Propose a correction?

3.9. Insert the following instructions:
int i3 = 15;
long l4 = i3 * (long) 2.15f;
long l5 = (long) (i3 * 2.5f);
System.out.println("\nl4: " + l4 + "\nl5: " + l5);
What does the System.out.println instruction display? Justify?
3.10. Insert the following instructions:
byte b3 = 10;
short s4 = 2 * b3;
State your observation? Propose a correction?
3.11. Insert the following instructions:
int i4 = 10;
long l6 = 1000;
int i5 = i4 + l6;
State your observation? Propose a correction?

II. Type Casting for Objects (Student Class)

3.12. Reuse the **Student** class from LAB5 and copy it into the package of LAB5, then insert the following instructions:

```
Object o2 = new Object();
Student e2 = new Student();
e2 = o2; // Error: Cannot cast Object to Student

Object o1 = new Object();
Student e1 = new Student();
o1 = e1; // No error
```

State your observation? Propose a correction?

Answer

3.1. A compilation error, we cannot assign an integer to a byte, a cast is necessary. Add the following instruction: b1 = (byte) (b1 + 1);
3.2. A compilation error, we cannot assign an integer to a byte, a cast is necessary. Add the following instruction: byte b2 = (byte) i1;
3.3. No error, we can do this without a cast since from **int** to **long**.
3.4. A compilation error, we cannot assign a long to an int, a cast is necessary. Add the following instruction: int i2 = (int) l1;
3.5. No error, we can do this without a cast since from **long** to **float**.
3.6. A compilation error, we cannot assign a float to a long, a cast is necessary. Add the following instruction: long l3 = (long) f1;
3.7. No error, we can do this without a cast since from **float** to **double**.
3.8. A compilation error, we cannot assign a double to a float, a cast is necessary. Add the following instruction: float f3 = (float) d1;
3.9. It displays:
l4: 30
l5: 37
For the instruction long l4 = i3 * (long) 2.15f;, there is a cast of 2.15f, resulting in 2, then 2 * 15 equals 30, so l4 is 30.

For the instruction long l5 = (long) (i3 * 2.5f);, there is a calculation of 2.5 * 15, resulting in 37.5, then a cast of this result, which gives 37.
3.10. A compilation error, we cannot assign an int to a short, a cast is necessary. Add the following instruction: short s4 = (short) (2 * b3);
3.11. A compilation error, we cannot assign a long to an int, a cast is necessary. Add the following instruction: `int i5 = (int) (i4 + l6);

II - Type Casting for Objects (Student Class)

3.12. A compilation error occurs at the instruction e2 = o2;. We cannot assign a reference of an object of type Object (parent class) to an object of type Student (subclass of Object). Casting is required. Add the following instruction:
e2 = (Student) o2;

For the instruction o1 = e1;, there is no error because the inheritance relationship allows us to do this.

LAB 7
(Polymorphism, Overloading, Method Overriding, and Abstract Class)

Statement:
In an educational institution, there are three types of people: secretaries, teachers, and students. Each person is characterized by their first name, last name, and address (street and city). The instance variables (attributes) are first_name, last_name , street, and city. nbPerson is a class variable that counts the total number of Person in the institution. All attributes are declared as protected.
Create a project named **LAB7_poly_Over_Overl_Abstractclas**.
Define the following public methods for the abstract class Person:

1. The constructor Person(String first_name, String last_name, String street, String city): creates and initializes an object of type Person.
2. String toString(): provides a string corresponding to the characteristics (attributes) of a person.
3. displays_person(): displays the characteristics of a person. It does nothing and is declared abstract.
4. static nbPerson(): displays the total number of people and the number of people by category. It is a class method.
5. modify_person(String street, String city): modifies the address of a person and calls displays_person() to verify that the modification has been made.

A Secretary is a Person with an additional attribute: idOffice (office number), which is of type int. The static variable nbSecretary is incremented in the constructor of the Secretary class.

Define the following public methods for the Secretary class:

1. The constructor Secretary: creates and initializes an object of type Secretary.
2. Override the String toString() method: provides a string corresponding to the characteristics of a Secretary.
3. Override the displays_person() method: displays the characteristics of a Secretary.
4. static nbSecretary (): returns the total number of Secretary.
5. Overload the modify_person (String first_name, String last_name, String street, String city, int numb) method from the base class Person to allow modification of the first_name, last_name, and office number.

A Student is a Person pursuing a degree. The attribute degree is of type String. The static variable nbStudent counts the number of students; it is incremented in the Student constructor.

Define the following public methods for the Student class:

1. The constructor Student: creates and initializes an object of type Student.
2. Override the String toString() method: provides a string corresponding to the characteristics of an Student.
3. Override the displays_Person() method: displays the characteristics of an Student.
4. static nbStudent(): returns the total number of Student.
5. Overload the modify_Person(String first_name,String last_name, String street, String city) method from the base class Person to allow modification of the first_name and first_name.

A Teacher is a Person teaching a specialty. The attribute specialty is of type String. The static variable nbTeacher counts the number of teachers; it is incremented in the Teacher constructor.

Define the following public methods for the Teacher class:

1. The constructor Teacher: creates and initializes an object of type Teacher.
2. Override the String toString() method: provides a string corresponding to the characteristics of an Teacher.
3. Override the displays_Person() method: displays the characteristics of an Teacher.
4. static nbTeacher(): returns the total number of Teacher.

Write a test class Institution that includes the main method, following the order of operations below:

1. Declare an array of 5 person, then create 5 objects (02 Secretary, 02 Students, 01 Teachers).
2. Fill the array with the 5 objects, then display the characteristics of person whose city is "Ain defla".
3. Test the static methods nbPerson() and modify_Person.

LAB 8 (Interface)

Statement: Some animals can make sounds, while others are mute. We will represent the act of making a sound using a method that displays the animal's sound on the screen.

1. Create a project named **LAB8_interface**.
2. Write an interface containing the method that allows an animal to make a sound.
3. Write classes for cats, dogs, and rabbits (which are mute).
4. Write a program with an array for animals that can make sounds, fill it with dogs and cats, and then make all these animals produce their sounds.
5. Describe what is displayed on the screen when this program is executed.