USTHB **By Dr MEKAHLIA Fatma Zohra**
Faculty of Computer Science
MOVEP Laboratory **PW OOP1**

PW 04 : Inheritance and Polymorphism

Exercise 01:

1- Create a class Person containing the following attributes: int id, String name, String surname, int age.
Note that id must be accessible only within the Person class.
name, surname, and age must be accessible from the child classes.

2- Add a constructor and a method display().

3- Create a Person object in the Main class and call display().
Observe the encapsulation of the attributes (try to access id directly from main() — what do you notice?).

4- Create a class Employee that inherits from Person, adding two attributes: String position and double salary.

5- Create a complete constructor for the Employee class.

6- Redefine the display() method while practicing code reuse and specialization: inheritance avoids code duplication and allows adding new behaviors.

7- Create a class Teacher that inherits from Employee, adding the attribute String specialty.

8- Create a complete constructor for the Teacher class.

9- Redefine the display() method in the Teacher class.

10- Create a Teacher object in Main, call display(), and observe the chain of super calls.

11- Delete the constructor from the Person class, then compile your code. What error do you get?

12- Add another child class AdministrativeStaff, including the field String department.

13- Create a constructor and completely redefine the display() method without calling super.display(). Explain the difference in behavior compared with Teacher.

14- Add the method getPerson() in Person, then override it in Teacher. Execute and observe: does it work?

stop here

15- Create a class AdjunctTeacher that is final and inherits from Teacher.

16- Create a class PartTimeAdjunctTeacher that inherits from AdjunctTeacher. What error do you get?

17- In the Employee class, add the method calculateBonus() which is public and final, and try to override it in Teacher. What error do you get?

In the main() method, create a Teacher object and an AdministrativeStaff object. Verify that the constructors of the parent classes (Person, Employee) are properly called using super.
Notice that the display() methods are overridden and that each displays both basic and specific information.
Understand how inheritance promotes code reuse without duplication.

18- Test partial overriding and the call to super.display() with two objects of type Teacher and AdministrativeStaff. Explain the difference.

19- Try to access id, name, surname, or age directly from main(). Explain the results.

20- Call the calculateBonus() method from a Teacher object. Explain the result.

21- Then, redefine the calculateBonus() method in Teacher. Explain the results.

Exercise 02:

Determine the results by applying the mechanism of static and dynamic binding.

```java
public class A {

	void m(A a) {
	System.out.println("m de A");
	}

	void m(B b) {
	System.out.println("m de A version2");
	}

	void n(A a) {
	System.out.println("n  de A");
	}
}
```

```java
public class B extends A {

	void m(A a) {
		System.out.println("m de B");
	}
	void n(B b) {
		System.out.println("n de B");
	}
}
```

```java
public class Application {

	public static void main(String[] args)
{
	// TODO Auto-generated method stub

		A a = new B();
		B b = new B();
		A a1 = new A();

		a.m(b);
		a.n(b);

		b.m(b);
		b.n(b);

		a.m(a1);
		a.n(a1);

		a1.m(b);
		a1.n(b);
	}
}
```

Exercise 03:

A media library contains different types of media. However, regardless of the type, each medium has a title. When a medium is created, its title is provided at creation time and cannot be changed afterward.

1. Define the class Media with its public constructor, a private field title, and its accessor. Which attribute should you add to prevent modifications of the title?
2. We want to assign a unique registration number as soon as a Media object is created: the first medium created should have number 0, then this number should increment by 1 with each new medium created. Add the necessary fields to the Media class, modify the constructor, and then add a method getNumero that returns the registration number of the medium. Which attribute(s) should be added to ensure that the registration number remains unique and non-modifiable?
3. Define the method afficher() that displays the registration number and the title of the medium.
4. Define the method plusPetit(Media doc) that checks whether the registration number of the current instance is smaller than that of doc.

We now want to define different types of media: books, dictionaries, and later, other types of media (comic books, bilingual dictionaries, etc.).
Each book has, in addition, an author and a number of pages.
Dictionaries, on the other hand, have as additional attributes a language and a number of volumes.
We would like to manipulate all items (books, dictionaries, etc.) through the same representation: that of Media.

1. Define the classes Book and Dictionary extending the class Media. Define for each a constructor allowing you to initialize all its respective instance variables.
2. Redefine the afficher() method in the classes Book and Dictionary.
3. Then define a few additional classes, for example ComicBook, Manga, BilingualDictionary, etc., each with extra properties. For each of these new classes, which class does it extend?
4. Add a main method to test your code. For example, in the Book class, successively create two books: “Harry Potter” (written by J. K. Rowling, 240 pages) and “A Game of Thrones” (written by George R. R. Martin, 789 pages). Display these books and make sure the registration number is correct.

We now want to define a media library to store all our media.
This will be a class containing the main function and an array baseDeDonnees describing the contents of the media library.
Fill the baseDeDonnees with several media items (of different types), then display it.

Exercise 04:

A drawing application must handle multiple geometric shapes (circle, rectangle, triangle, etc.). Each shape has a method to calculate its area, but the implementation varies depending on the type of shape.

1. Create a class Form that contains:

- an attribute name (type String)
- a constructor taking the name as a parameter
- a method calculateArea() that returns 0 or displays a generic message: "Area not defined"

2. Then create three child classes:

- Circle with an attribute radius
- Rectangle with two attributes length and width
- Triangle with two attributes base and height

3. Override the calculateArea() method in each child class to return the correct area:

- Circle → $\pi \times \text{radius}^2$
- Rectangle → length × width
- Triangle → (base × height) / 2

4. Create a Main class:

- declare an array or a list of Form
- add a Circle, a Rectangle, and a Triangle to it
- loop through the list and display the name and area of each shape by calling calculateArea()

Observation: although the list is of type Form, the method of the specific class (Circle, Rectangle, etc.) is executed. This is runtime polymorphism (dynamic binding).

5. In the Form class, add an overloaded method display():

```
void display() {
    System.out.println("Generic form");
}

void display(String message) {
    System.out.println("Form: " + name + " - " + message);
}
```

6. In the main method, test both versions:

```
Form f = new Form("Form");
f.display();
f.display("Overload test");
```

<u>Observation:</u> here, the method called is determined at compile time, based on the method signature. This is compile-time polymorphism (static binding).

7. Declare a variable of type Form:

```
Form f = new Circle("Circle", 3.5);
System.out.println("Area: " + f.calculateArea());
```

Which method is called? Why?

8. Add a method in the Form class:

```
void describe() {
    System.out.println("I am a generic form.");
}
```

Override it in Circle, Rectangle, and Triangle. Then create an array of Form and call describe() for each.

<u>Observation:</u> the overridden methods are called according to the actual type of the object (dynamic binding). Overloaded methods are chosen according to the signature (static binding).

9. Questions:
   a) What is the difference between overriding (@Override) and overloading?
   b) Why is polymorphism said to promote code reuse and flexibility?
   c) What would happen if calculateArea() were not overridden in the child classes?
   d) Which binding (static or dynamic) is used in each case above?