Mila University Center
2nd Year – Bachelor's in Computer Science
Course: Object-Oriented Programming

CHAPTER I: Introduction To Oriented Object Programming

Lecturer : DR. SADEK BENHAMMADA
EMAIL : s.benhammada@centre-univ-mila.dz

Course outline

1. Programming Paradigms
2. From procedural programming to object-oriented programming
3. Fundamental Concepts of Object-Oriented Programming (OOP)
4. Overview of the Java Programming Language

Course outline

1. **Programming Paradigms**
2. From procedural programming to object-oriented programming
3. Fundamental Concepts of Object-Oriented Programming (OOP)
4. Overview of the Java Programming Language

1. Programming Paradigms

Programming Paradigms

- A programming paradigm is a fundamental styles or approaches to programming , based on a set of principles or theory
- Each paradigm provides a distinct way of thinking programming and structuring code,

Classification of programming paradigms:

1. **Procedural Programmin (PP)**
2. **Object-Oriented Programming: (OOP)**
3. **Declarative Programming**

1. Programming Paradigms

Classification of programming paradigms:

1. **Procedural Programming**: Organizes code into procedures and/or functions: (**C**,C++, **Python,** etc.).

2. **Object-Oriented Programming: (OOP) :** OOP organizes code into objects that encapsulate data and behavior. **Java, C++, Python, C#.**

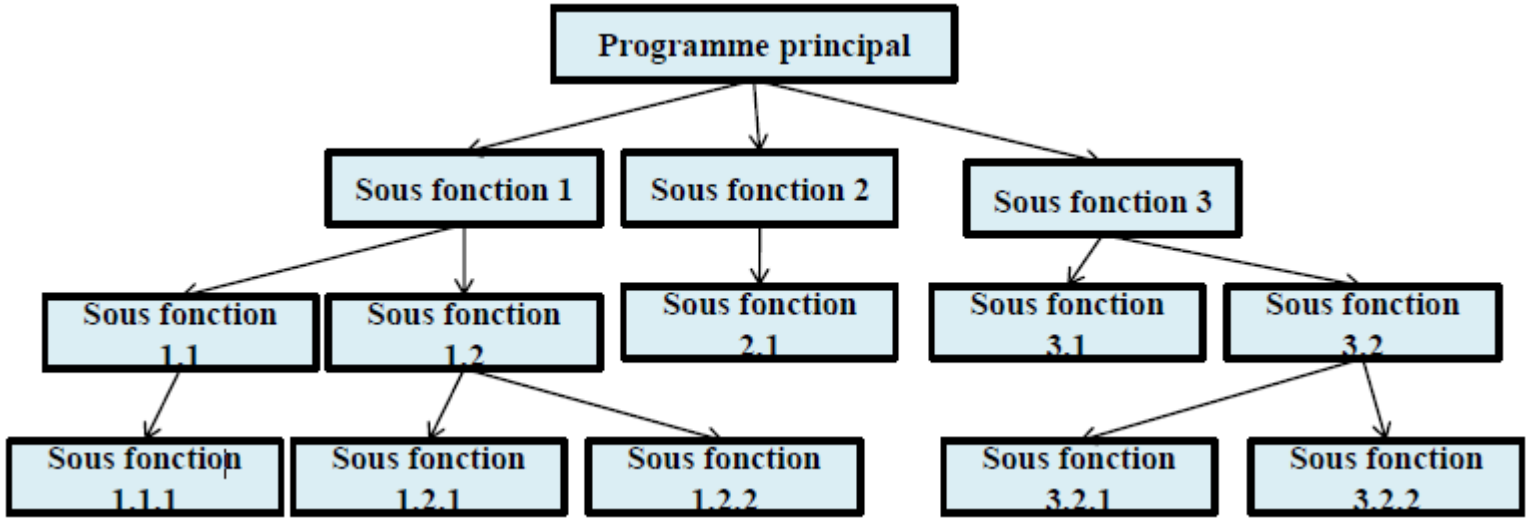
3. **Declarative Programming:** Focuses on **what** the program should accomplish rather than **how** to accomplish it.
   - **Domain-specific language: HTML, XML, LaTeX.**
   - **Functional Programming:** Treats computation as the evaluation of mathematical functions: **Lisp, Haskell**.
   - **Logic Programming :** Uses formal logic to express computations : **Prolog**
   - **Data definition languages (SQL)**

Course outline

1. Programming Paradigms
2. **From procedural programming to object-oriented programming**
3. Fundamental Concepts of Object-Oriented Programming (OOP)
4. Overview of the Java Programming Language

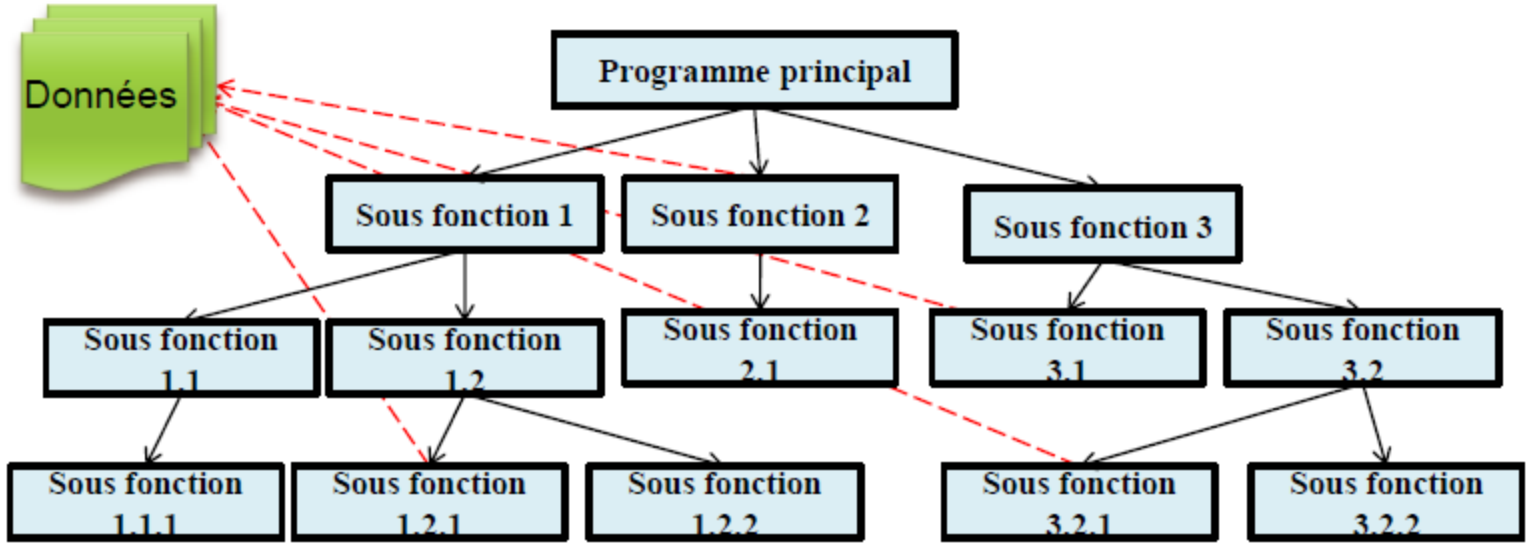
Top-down approach

- The main program is divided down into smaller modules (functions or procedures),
- Each module is then divided into sub-modules,
- The decomposition continues until it reaches controllable components (the length does not exceed one page if possible).
- Principal program is divided into smaller, reusable functions or procedures.



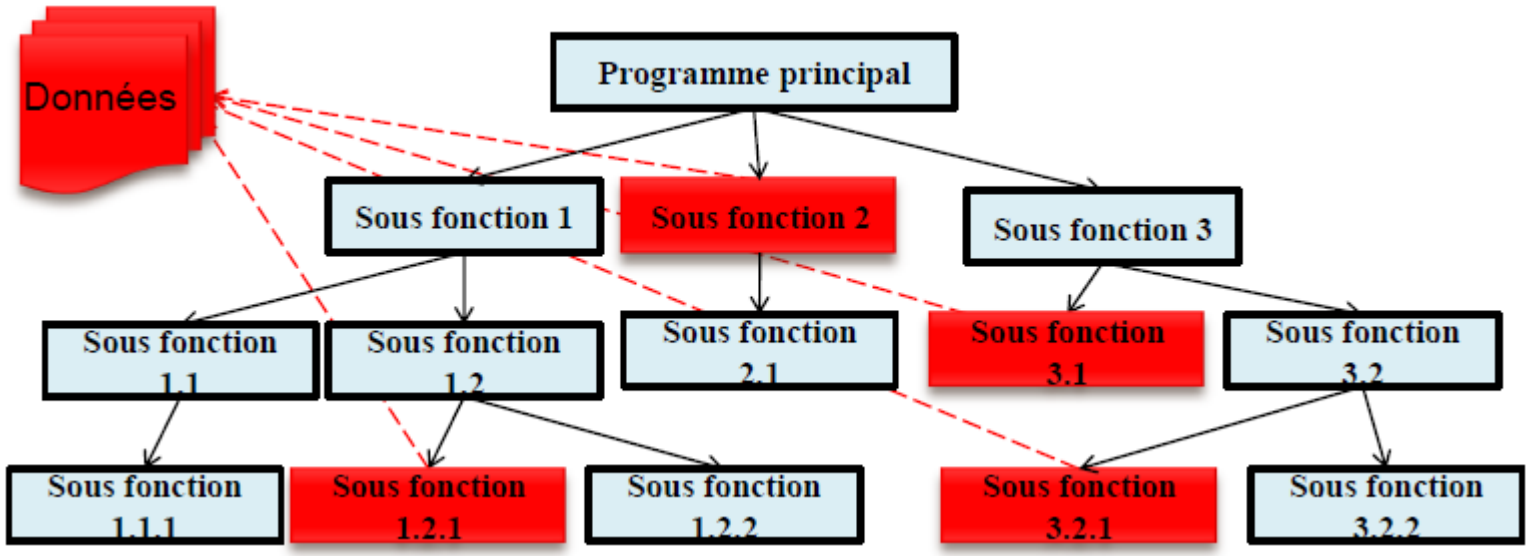
Data and Functions are Separate

- Variables can be global (accessible throughout the program) or local (accessible only within a function).
- Procedural programming separates between data and programs that manipulate them.
- Functions operate on external data rather than being part of the data structure itself.



Limitations:

1. **Maintenance in Procedural Programming is Hard :** Changes to data structures or global variables can require modifications to multiple functions.



Limitations:

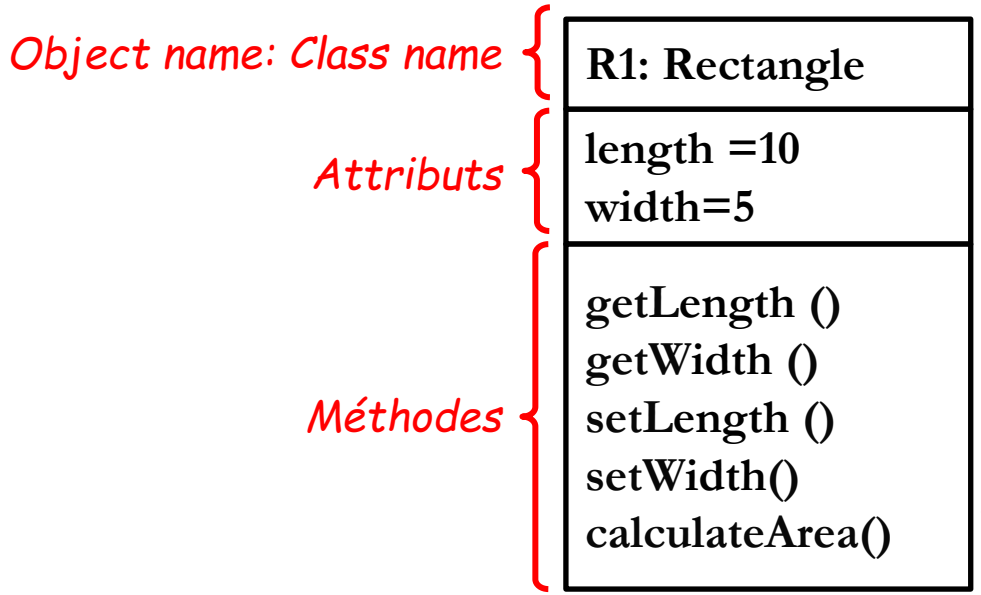
2. **Poor Data Security (Lack of Encapsulation) :** Data is not protected, as global variables can be accessed and modified from anywhere.
3. **Code Duplication :** Procedural programming does not support **inheritance**, so similar functions are often written multiple times.
4. **Scalability Issues in Large Applications:** As the project grows, the number of functions increase significantly, leading to **code complexity**.
5. **Difficult Debugging and Testing :** If multiple functions modify the same global variable, debugging becomes difficult.
6. **Lack of Real-World Modeling :** Procedural programming does not naturally represent real-world entities.
7. **Code Reusability Through Functions : F**unction reuse is limited compared to the inheritance and polymorphism of OOP.

2. From procedural programming to object-oriented programming

A. Characteristics of Procedural Programming (PP) | **B. Characteristics of Oriented Object Programming**

Object-Oriented Programming (OOP) was designed to **overcome the limitations of procedural programming** by :

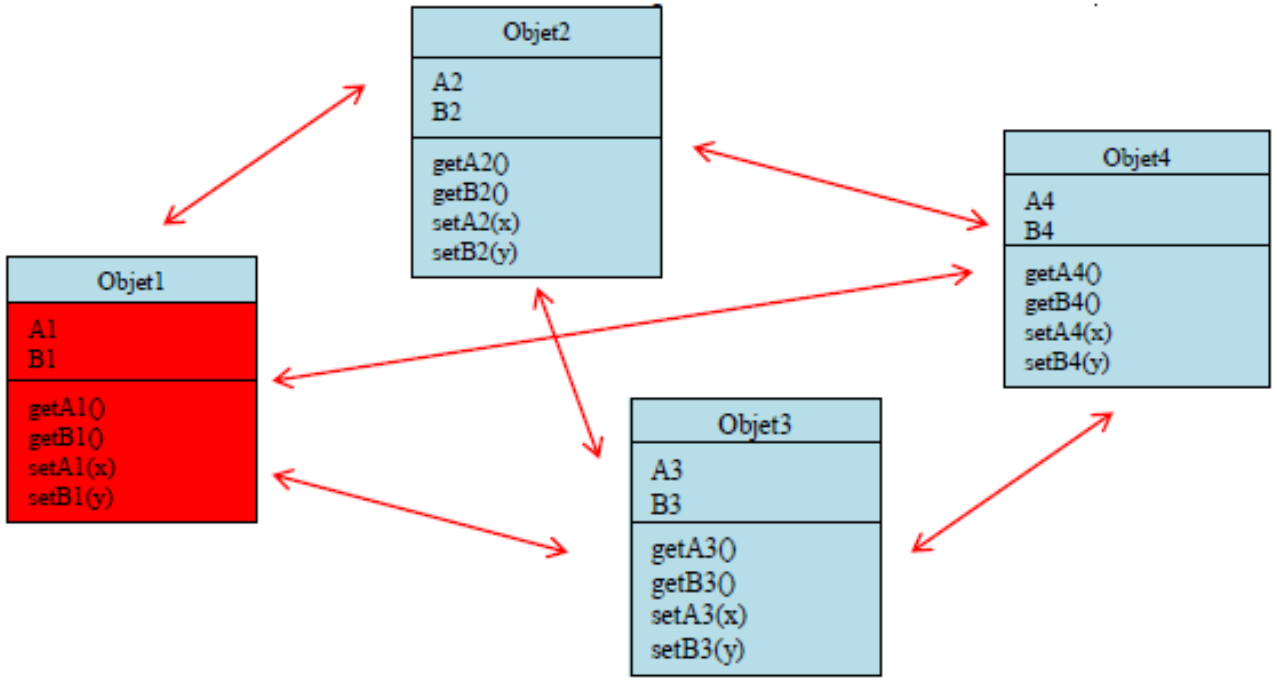
- Association of data structures and the processes that manipulate them in coherent entities
- These entities are **objects**.
- The data structures associated with an object are its **attributes**.
- The processes associated with an object are its **methods**.
- The **attributes** of an object **are only accessible by its methods**.
- **Exemple**



2. From procedural programming to object-oriented programming

A. Characteristics of Procedural Programming (PP) | **B. Characteristics of Oriented Object Programming**

- The object-oriented approach views software as a **collection of interacting objects**.
- Each object represents an **independent entity** with its own **data (attributes)** and **behavior (methods)**.
- The software's functionality emerges from the **collaboration and interactions** between these objects, promoting **modularity, reusability, and scalability**..



Course outline

1. Programming Paradigms
2. From procedural programming to object-oriented programming
3. **Fundamental Concepts of Object-Oriented Programming (OOP)**
4. Overview of the Java Programming Language

3. Fundamental Concepts of Object-Oriented Programming (OOP)

Object	Class	Message	Inheritance	Encapsulation	Polymorphism

A. Object

- An object is a coherent runtime entity that that has :
  - **Attributes**: Variables that store the **state** or data of the object.
  - **Methods**: Functions that define the **behavior** or actions the object can perform.

Objet = Identity + State (attributs) + Behavior (methods)

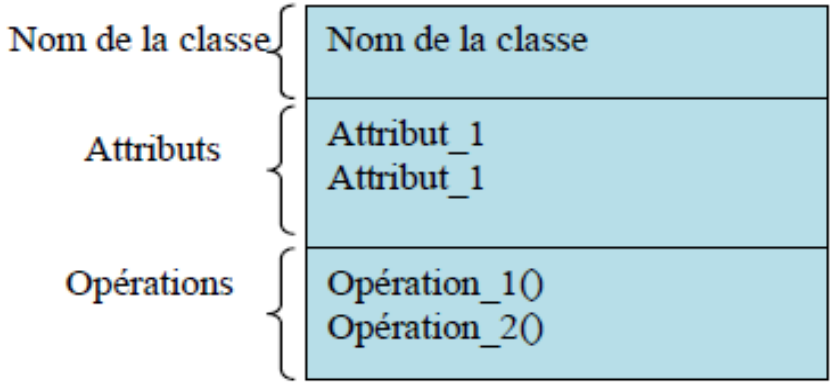
  - L'identité: L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état: (Deux objets demeurent distincts même si leurs attributs contiennent les mêmes valeurs)

3. Fundamental Concepts of Object-Oriented Programming (OOP)

B. Class

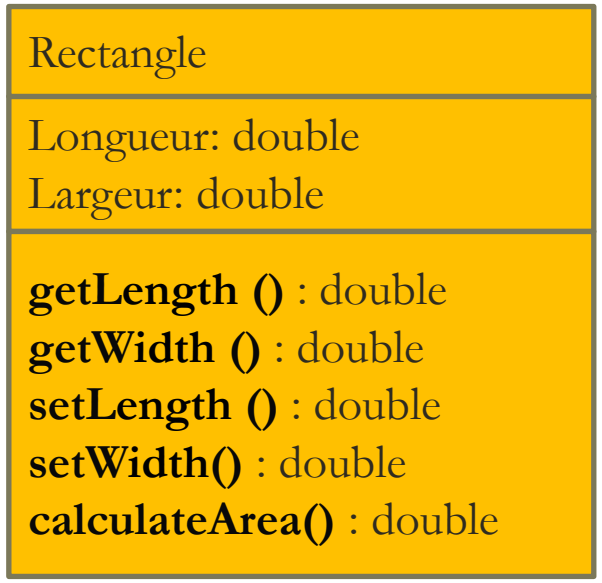
- **Definition of a Class**
- A **class** is a template for creating objects. It defines the structure and behavior that its objects will have.
- A class defines:
  - **Attributes** with their **names** and types (but not their values which are specific to each object).
  - **Methods** (operation) with their **signatures** and the **code** that describes the associated behavior.

- An object is an instance (entity) of a class.
- Multiple objects can be created from a single class, each w
-

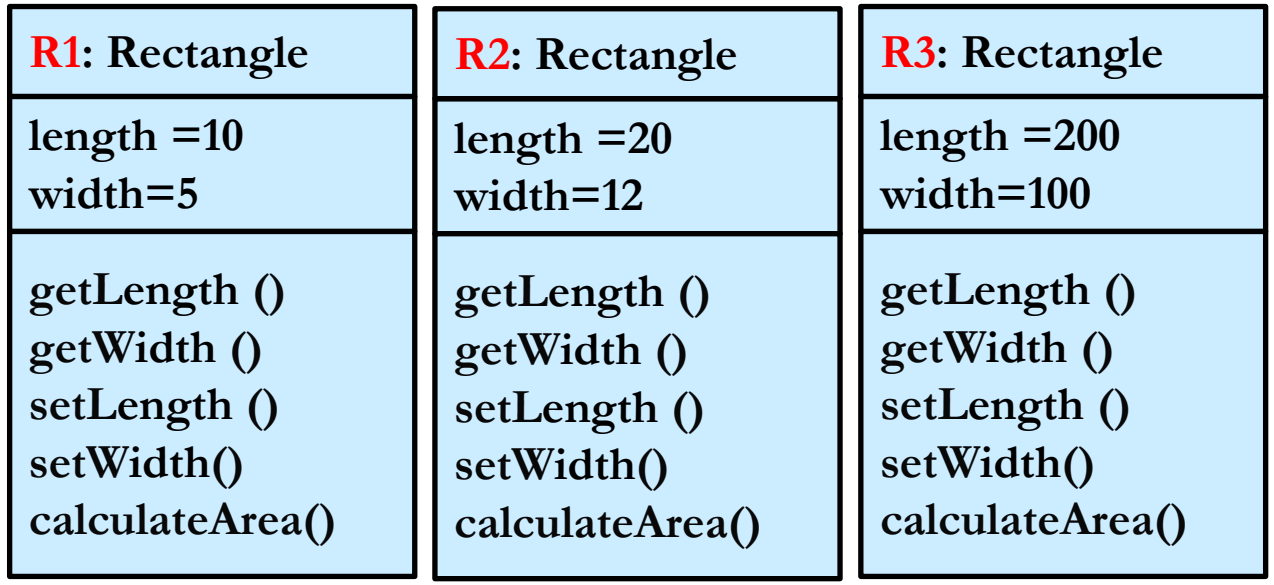


B. Class

Example



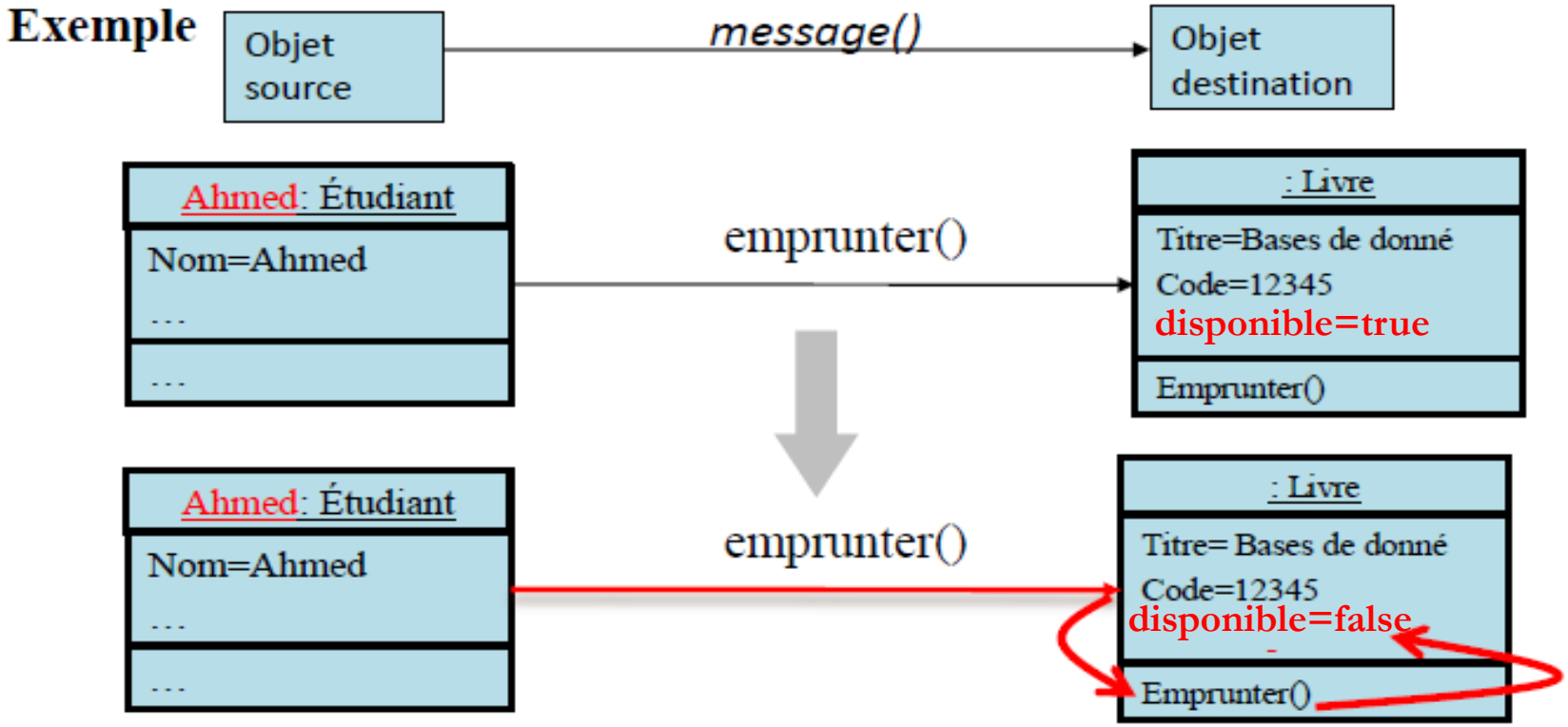
The Rectangle class



Objects R1, R2, and R3 of the Rectangle class.

C. Message

- Objects communicate with each other by sending messages.
- A message is a request sent from one object to another, asking the receiving object to perform a specific action
- Typically : A **message** represents the **call** of a **method** of the **destination object** by a **source object**.



3. Fundamental Concepts of Object-Oriented Programming (OOP)

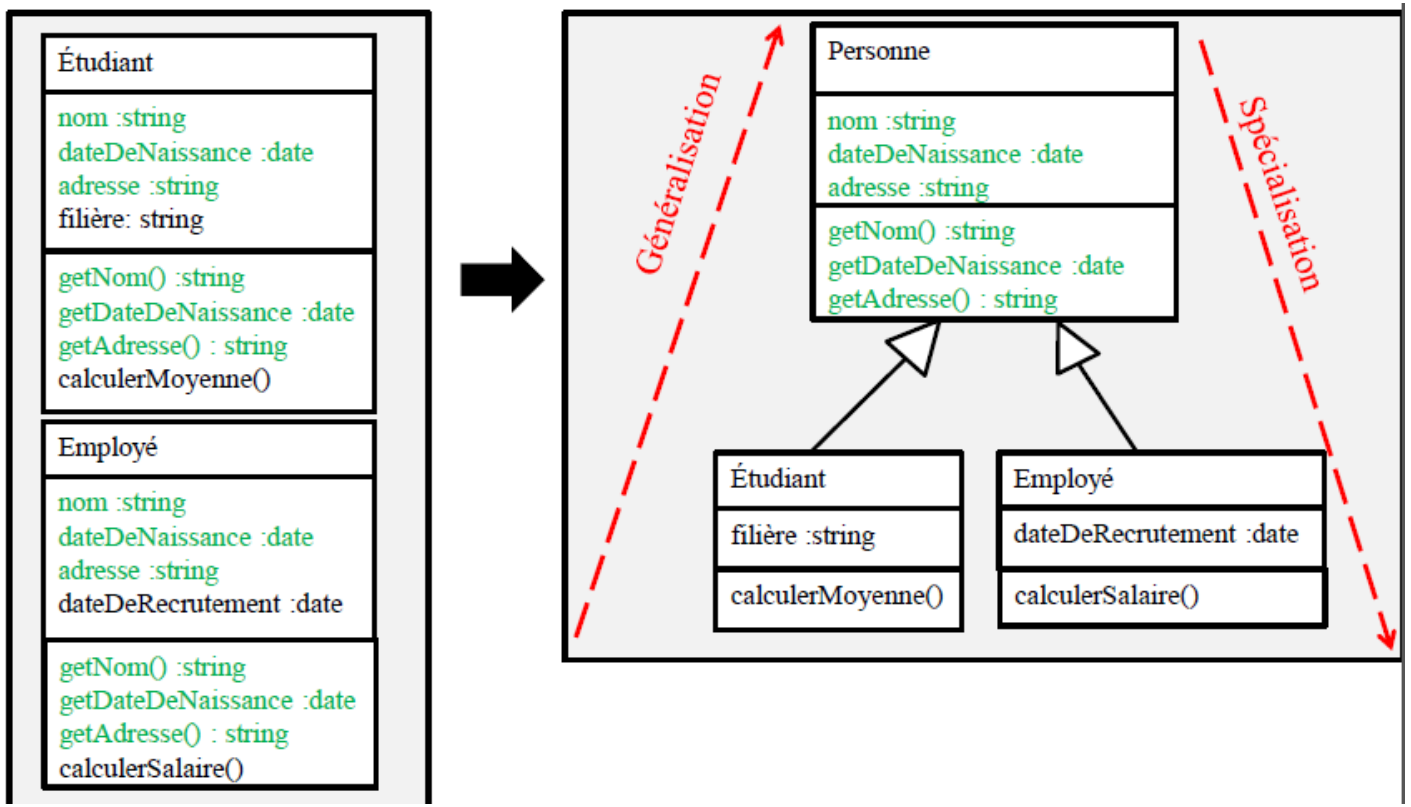
D. Inheritance :

- Inheritance describes a relationship between a **parent class or superclass** and a **child class** or **subclass**.
- The **child class (subclass)**:
  - **Inherits all attributes** and **methods** from the **parent class (superclass)**
  - **Can add** new **attributes** and **methods**

- **Purpose on Inheritance**
  - Reuse existing code of the **superclass** (or **parent class**) in a new class the **subclass** (or **child class**).
  - **Avoid duplication :** Reuse existing code reduces redundancy, and makes programs more efficient and maintainable.
  - Create a logical hierarchy of classes.
  - Extend or modify the behavior of the parent class.

3. Fundamental Concepts of Object-Oriented Programming (OOP)

Example :



3. Fundamental Concepts of Object-Oriented Programming (OOP)

Inheritance enables **generalization** and **specialization**

1. Generalization

- Generalization is the process of extracting common attributes and methods from multiple classes and combining them into a more general, higher-level class. This higher-level class is often referred to as a **superclass** or **parent class**.
- **How It Works**:
  - Identify common attributes and methods in multiple classes.
  - Create a superclass that contains these common attributes and methods .
  - Use inheritance to allow subclasses to inherit from the superclass.

3. Fundamental Concepts of Object-Oriented Programming (OOP)

Inheritance enables **generalization** and **specialization**

- **2. Specialization**
- Specialization is the process of creating new classes (subclasses) that inherit from a more general class (superclass) and add or modify specific attributes and methods.
- **How It Works**:
  - Create a subclass that inherits from a superclass.
  - Add new attributes or methods specific to the subclass.
  - Override methods from the superclass to provide specialized behavior.

E. Encapsulation

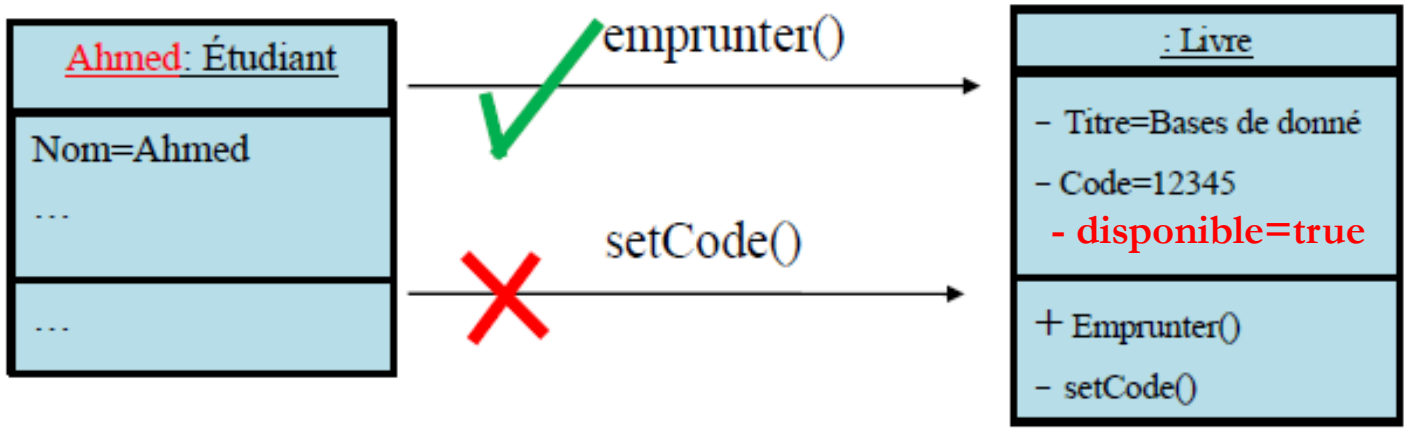
- Encapsulation consists of hiding part of attributes and methods and only allowing access through controlled methods.
- Hidden attributes are accessible by other objects through **services** (visible methods).
- An object's **services** can be invoked through **messages**.
- The list of **messages** to which an object is capable of responding constitutes its **interface** (its external view).

- **Purpose**:
  - Provides a **clear** and **consistent interface** for interacting with the object.
  - **Maintains the integrity** of the data : it allows to prohibit direct access to the attributes of the objects (use of accessors).
  - **Improved Maintainability**: Allows to change the internal implementation of a class without affecting external code.
-

3. Fundamental Concepts of Object-Oriented Programming (OOP)

Encapsulation allows to define visibility levels for attributes and methods. There are three levels of visibility:

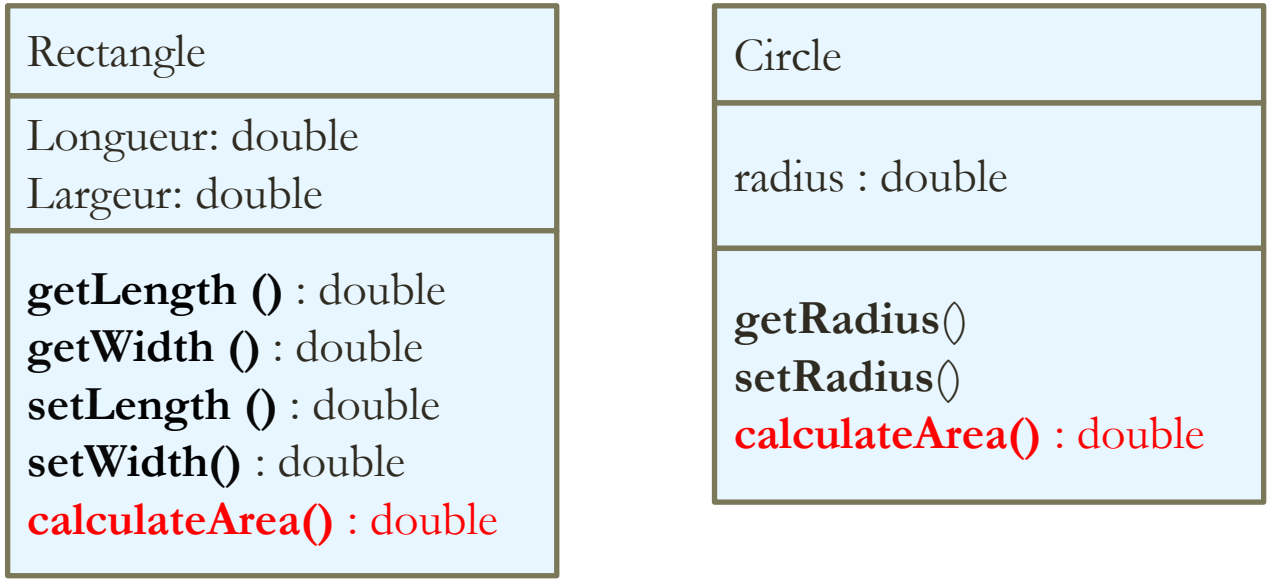
- **Public(+):** All objects can access an object's attributes or methods defined with the public visibility level. This is the lowest level of protection.
- **Protected(#):** Access is restricted to derived objects.
- **Private(-)**: Access is restricted to methods of the same class. This is the highest level of data protection



3. Fundamental Concepts of Object-Oriented Programming (OOP)

F. Le polymorphisme

- Polymorphism allows **one interface to have multiple implementations**, making the code more flexible and scalable.
- Polymorphism is the ability of a single interface to represent multiple forms of behavior.
- It allows the same method name to be used for different types of objects, where each object can provide its own specific implementation of the method.
- **Exemple**



When the message " **calculateArea**()" is executed, it does not have the same effect on an object class Rectangle and an of class Circle. The method "calculateSurface()" is polymorphic.

Course outline

1. Programming Paradigms
2. From procedural programming to object-oriented programming
3. Fundamental Concepts of Object-Oriented Programming (OOP)
4. **Overview of the Java Programming Language**

4. Overview of the Java Programming Language

Présentation de Java

- Java is an object-oriented programming language.
- It was developed in 1991 by **Sun Microsystems** (purchased by Oracle Corporation in 2009).
- In 2010, **Oracle** Corporation acquired **Sun Microsystems**, making Java an Oracle product.
- Oracle continues to maintain and update Java, introducing new features and performance enhancements.

4. Overview of the Java Programming Language

- **Quelques chiffres à propos de Java (2011):**
  - **Enterprise Adoption:** Java remains widely used, with over **90%** of **Fortune 500 companies** relying on it for critical applications.
  - **Number of Developers**: Estimated 18.7 million Java developers worldwide in 2024.
  - **Language Popularity**: Ranked 3rd in the TIOBE Index (Jan 2025), with 10.15% market share (https://www.tiobe.com/tiobe-index/).
  - **Mobile Devices**: Java remains key for Android development, powering billions of devices.
  - **Smart Cards**: Over 1.4 billion Java-based smart cards are produced annually in

Jan 2025	Jan 2024	Change	Programming Language	Ratings	Change
1	1		Python	23.28%	+9.32%
2	3	↑	C++	10.29%	+0.33%
3	4	↑	Java	10.15%	+2.28%
4	2	↓	C	8.86%	-2.59%
5	5		C#	4.45%	-2.71%

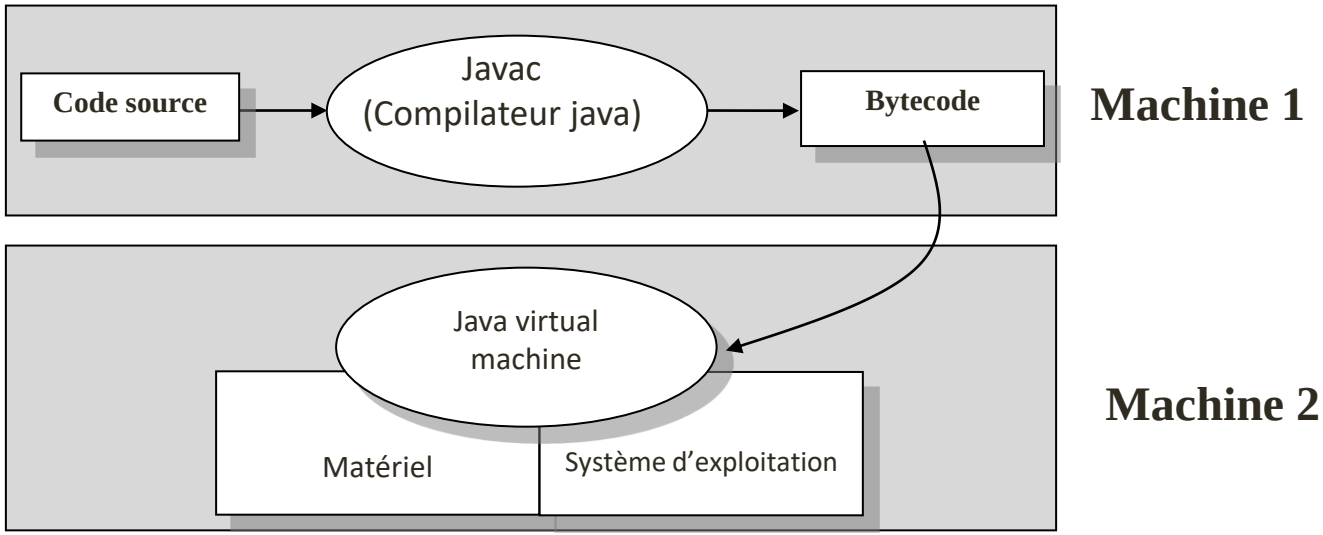
Key Features of Java

- Java has several key features that have contributed to its widespread success. Here are its most important characteristics

1. Java is Interpreted and Compiled

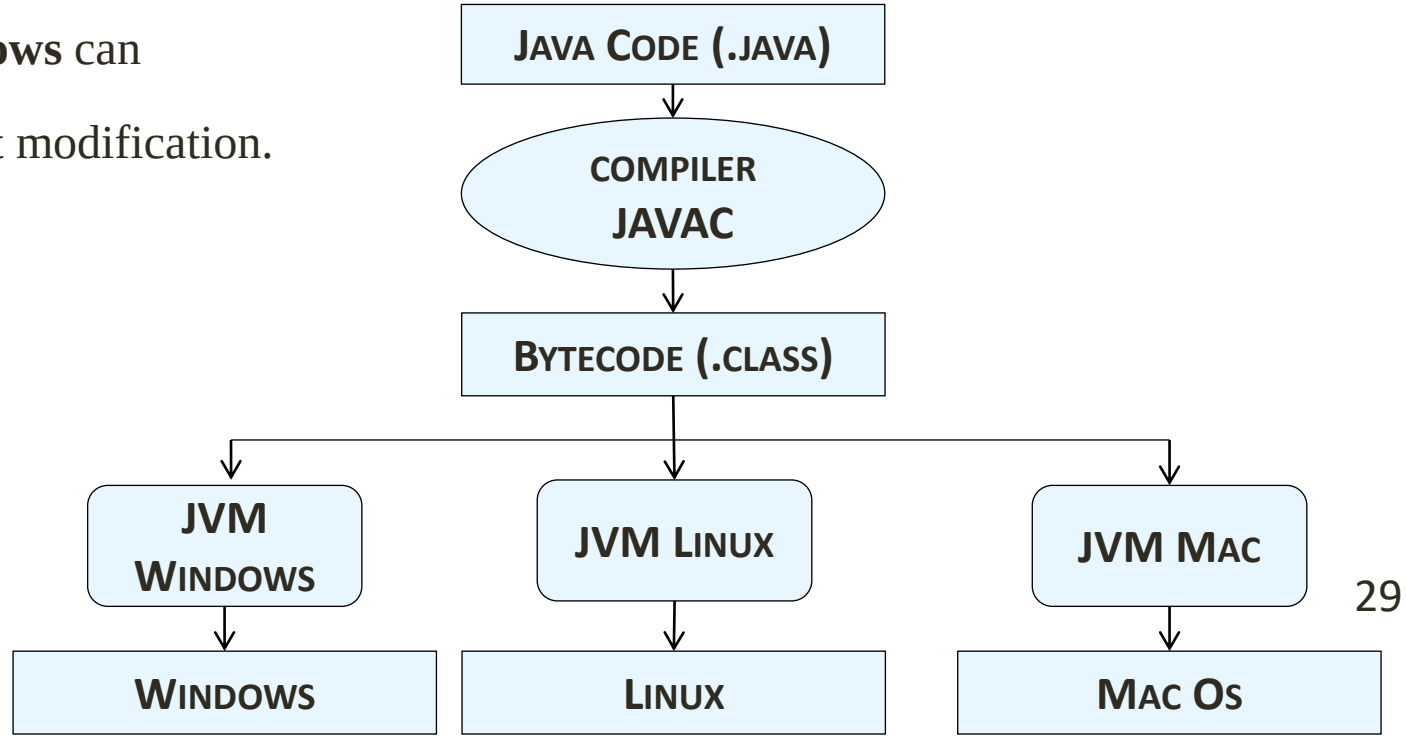
- Java uses a **hybrid approach** of **compilation and interpretation**:
  - The **source code** is compiled into **bytecode** by the **Java Compiler (javac)**.
  - **Bytecode** is an **intermediate representation** that is **not directly executed** by the CPU.
  - Instead, the **Java Virtual Machine (JVM)** interprets and executes the bytecode.

This process allows Java programs to be **portable and platform-independent**.



Key Features of Java

- **2. Java is Platform-Independent (Portable)**
- Java follows the **"Write Once, Run Anywhere" (WORA)** principle.
- Compiled **bytecode** can run on **any device** that has a **JVM**, regardless of the operating system.
- The JVM is platform-specific, but Java code itself does **not** depend on the underlying system.
- **Example:** A Java program compiled on **Windows** can run on **Linux, macOS, or any other OS** without modification.



4. Overview of the Java Programming Language

Key Features of Java

3. Java is fully Object-Oriented, meaning everything is based on **classes and objects**.

4. Java is Simple and Developer-Friendly : java is **inspired by C and C++**, but **removes** complex features like: **Manual memory management** (Java has automatic garbage collection), **Pointers** (avoiding security risks and errors), **Multiple inheritance** (Java uses interfaces instead).

5. Java is Strongly Typed :

- Java enforces **strict type-checking** at **compile-time**.
- Implicit type conversions that may **lead to data loss** are **not allowed**

- **Example:**
- This is **valid in C** but **invalid in Java**:

```
double a = 5.5;
int y = a; // Allowed in C (but may lose precision)
```

- In Java, an **explicit cast** is required:

```
double a = 5.5;
int y = (int) a; // Explicit type conversion
```