University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Object-Oriented Programming
Inheritance

10 avril 2024

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester : S4

Plan

Inheritance

A mechanism where a class acquires properties and behaviors from another class

- A new class of objects can be created conveniently by inheritance
- the new class (called the ***subclass***) starts with the characteristics of an existing class (called the ***superclass***), possibly customizing them and adding unique characteristics of its own.

Benefits of inheritance

- Code Reusability
- Improved Code Organization
- Flexibility and Polymorphism
- Reduced Development and Maintenance Costs
- Promotes Extensibility

Superclass

Superclass (parent class / base class) :

- The original class from which a subclass inherits.
- Defines common attributes and methods that can be used by subclasses.
- Serves as a foundation for building more specialized classes

Subclass

Subclass (child class) :

- A new class that inherits from a superclass
- Inherits all public and protected members (attributes and methods) from the superclass
- Can add its own attributes and methods to specialize its behavior
- Can override inherited methods to provide different implementations

Extends : The keyword used to declare that a subclass inherits from a superclass

Syntax of Inheritance

```
class Superclass {
  // Superclass body
}

class Subclass extends Superclass {
  // Subclass body
}
```

Example - Superclass

```java
public class Person {
  private  String firstName;
  private String lastName;
  private int age;

  public Person(String firstName, String lastName, int
        age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
  }
   public  String getFirstName() {
    return firstName;
  }
  // Methods ...
}
```

Example - Subclass

```
public class Student extends Person {
  protected int id;
  private String speciality;

  public Student(String firstName, String lastName, int
        age, int id, String speciality) {
    super(firstName, lastName, age);
    this.id = id;
    this.speciality = speciality;
  }
  public void setSpeciality(String speciality) {
    this.speciality = speciality;
  }
    // Methods...
}
```

Things to Consider

- **Visibility** : Subclasses only inherit members declared as public, or protected in the superclass (*and* ***default*** *in the same package*). Private members are not accessible.
- **Overriding** : Subclasses can override inherited methods to provide different behavior. This is useful for customization.
- **Final Keyword** : You can prevent a class from being inherited using the ***final*** keyword.

Constructors and Inheritance

- **Constructors are not inherited** : Unlike fields and methods, constructors are not directly inherited by subclasses in Java. This is because constructors are specific to the object creation process of a particular class and its needs.
- **Calling the superclass constructor** : Subclasses can explicitly call the constructor of their superclass using the "**super**" keyword..
- **Default constructor** : If a subclass doesn't have an explicitly defined constructor, the Java compiler will implicitly call the superclass's no-argument (default) constructor, if it exists. Otherwise, a compilation error will occur :

  (If the superclass doesn't have a no-argument (*default*) constructor and only has parameterized constructors, the subclass cannot have a default constructor)

The super Keyword

- Used with inheritance to interact with the superclass (parent class) from a subclass
- Access and interact with **fields** (variables) and **methods** defined in the superclass, even if they are *hidden* or *overridden* in the subclass
- Used to explicitly call the constructor of the superclass

super is a keyword that provides a way to access members of the superclass in the context of a subclass

Example - super

```
class Person {

  private String name;

  public Person(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }
}
```

```java
class Student extends Person {
  private int studentId;

  public Student(String name, int studentId) {
    super(name);
    this.studentId = studentId;
  }

  public void introduce() {
    System.out.println("Hello, my name is " + super.getName()
          + " and my student ID is " + studentId + ".");
  }
}

public class Main {
  public static void main(String[] args) {
    Student student = new Student("Alice", 12345);
    student.introduce();
  }
}
```

Types of Inheritance

- **Single inheritance** – one superclass, one subclass
- **Multilevel inheritance** – chained subclasses and superclasses
- **Hierarchical inheritance** – multiple subclasses extend one superclass
- **Multiple inheritance** : not supported in Java
- **Hybrid Inheritance** : a mix of two or more of the above types of inheritance It can involve any combination of single, multilevel, and hierarchical Inheritance

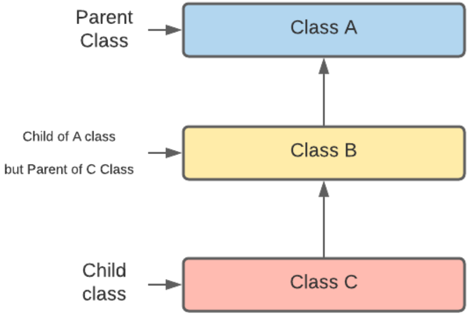
Single inheritance

A class can have only one direct parent class :

```
class Parent {

  // Parent class methods and fields
}

class Child extends Parent {

  // Child class can access methods and fields of
        Parent
}
```

Multilevel Inheritance

A class is derived from a class which is also derived from another class :

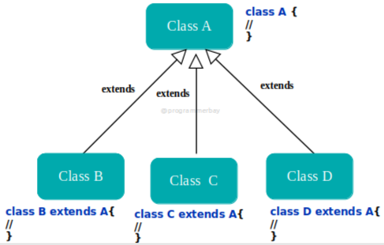


```
class Grandparent {
  // Grandparent class methods and fields
}
class Parent extends Grandparent {
  // Inherits from Grandparent
}
class Child extends Parent {
  // Inherits from Parent (and indirectly from Grandparent)
}
```

```java
public class Person {
  String name;  int age;
  public Person(String name, int age)
  {  this.name = name;    this.age = age;  }
}
public class Employee extends Person {
  String employeeID;
  public Employee(String name, int age, String employeeID) {
    super(name, age);
    this.employeeID = employeeID;
  }
}
public class Professor extends Employee {
  String department;
  public Professor(String name, int age, String employeeID,
         String department) {
    super(name, age, employeeID);
    this.department = department;
  }
}
```

Hierarchical Inheritance

multiple classes inherit from a single parent class. This means a single superclass can have multiple subclasses :



```
class Parent {
  // Parent class methods and fields
}
class Child1 extends Parent {
  // Inherits from Parent
}
class Child2 extends Parent {
  // Also inherits from Parent
}
```

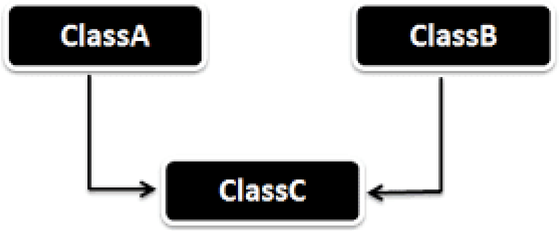
```java
public abstract class Shape {
  public abstract double area();
  public abstract double perimeter();
}

public class Circle extends Shape {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  @Override
  public double area() {
    return Math.PI * radius * radius;
  }

  @Override
  public double perimeter() {
    return 2 * Math.PI * radius;
  }
```

```java
public class Rectangle extends Shape {
  private double width;
  private double height;

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

  @Override
  public double area() {
    return width * height;
  }

  @Override
  public double perimeter() {
    return 2 * (width + height);
  }
}
```

```java
public class Main {
  public static void main(String[] args) {
  Shape circle = new Circle(5.0);
  Shape rectangle = new Rectangle(4.0, 6.0);

  System.out.println("Circle Area: " + circle.area());
  System.out.println("Circle Perimeter: " +
        circle.perimeter());

  System.out.println("Rectangle Area: " + rectangle.area());
  System.out.println("Rectangle Perimeter: " +
        rectangle.perimeter());
}
}
```

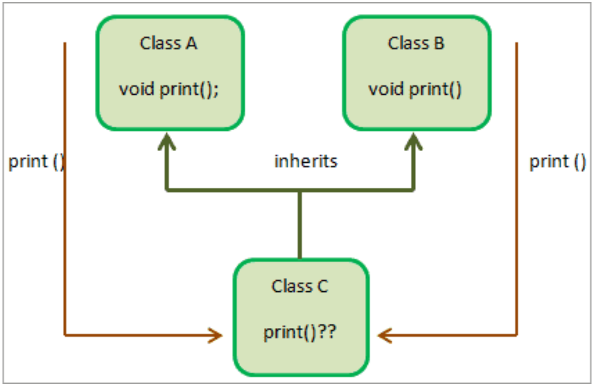
Multiple inheritance

A class inherits behaviours and attributes from more than one parent class



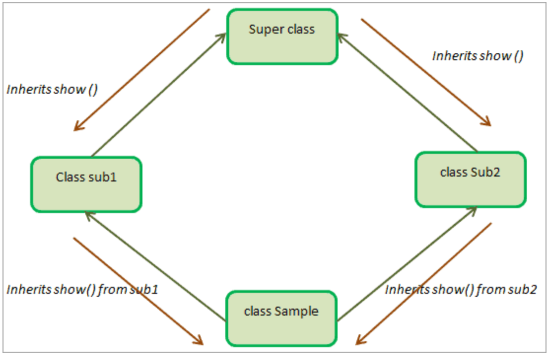
Multiple inheritance : Ambiguity

- **Complexity and Ambiguity** : when the parent classes have methods or attributes with the same names but different implementations. This ambiguity can make the code harder to read, maintain, and debug.



Multiple inheritance : Diamond Problem

- This problem arises when a class inherits from two parent classes, which themselves inherit from a common ancestor, forming a ***diamond*** shape in the inheritance hierarchy.
- The subclass inherits two copies of the methods and fields from the common ancestor, leading to ambiguity about which implementation to use



Exercice 01

Create a set of Java classes representing individuals within an academic institution, utilizing inheritance :

- **Person** : This base class should hold basic information common to all individuals (e.g., name, age, contact information).
- **Employee** : This class should inherit from Person and include additional attributes and methods specific to employees (e.g., job title, salary).
- **Student** : This class should inherit from Person and include attributes and methods related to students (e.g., student ID, Speciality).
- **Professor** : This class should inherit from Employee and include attributes and methods specific to professors (grade, specialization, departement).

`https://github.com/hamdani2023/javaPOO_ISIL_S04`

Declaring a class as final (1)

- This prevents other classes from inheriting from the final class.
- This is useful when you want to ensure a class cannot be extended and its behavior remains consistent.
- For example, a MathUtils class containing static utility methods might be declared final to prevent subclasses from overriding these methods and potentially introducing unexpected behavior.

Declaring a class as final (2)

When a class is declared with the final keyword, it cannot be subclassed.

```
public final class FinalClass {

  // class body
}

// This would cause a compile-time error
  class ExtendedClass extends FinalClass {

  }
```

Declaring a method as final (1)

- This prevents subclasses from overriding the method.
- This is useful when you want to ensure the specific implementation of a method remains unchanged in subclasses.
- For example, a calculateArea() method in a Shape class might be declared *final* to ensure all shapes (e.g., Circle, Rectangle) use the same logic for calculating area..

Declaring a ***field*** as ***final*** simply means its value cannot be changed after initialization, but it does not prevent inheritance $-\!>$ **Constant**

Declaring a method as final (2)

Declaring a method as final means it cannot be overridden by sub-classes

```
public class SuperClass {
  public final void showFinalMethod() {
    System.out.println("This method is final and cannot be
           overridden.");
  }
}

public class SubClass extends SuperClass {
  // This would cause a compile-time error
   @Override
   public void showFinalMethod() {
         System.out.println("Attempting to override a final
                method.");
     }
}
```

Questions ?