Moștenirea

Moștenirea este un concept fundamental al programării orientate pe obiecte (OOP) care permite unei clase să moștenească atributele și metodele unei alte clase. Clasa care moștenește se numește subclasă (sau clasă derivată), iar clasa din care se moștenește se numește superclasă (sau clasă de bază).

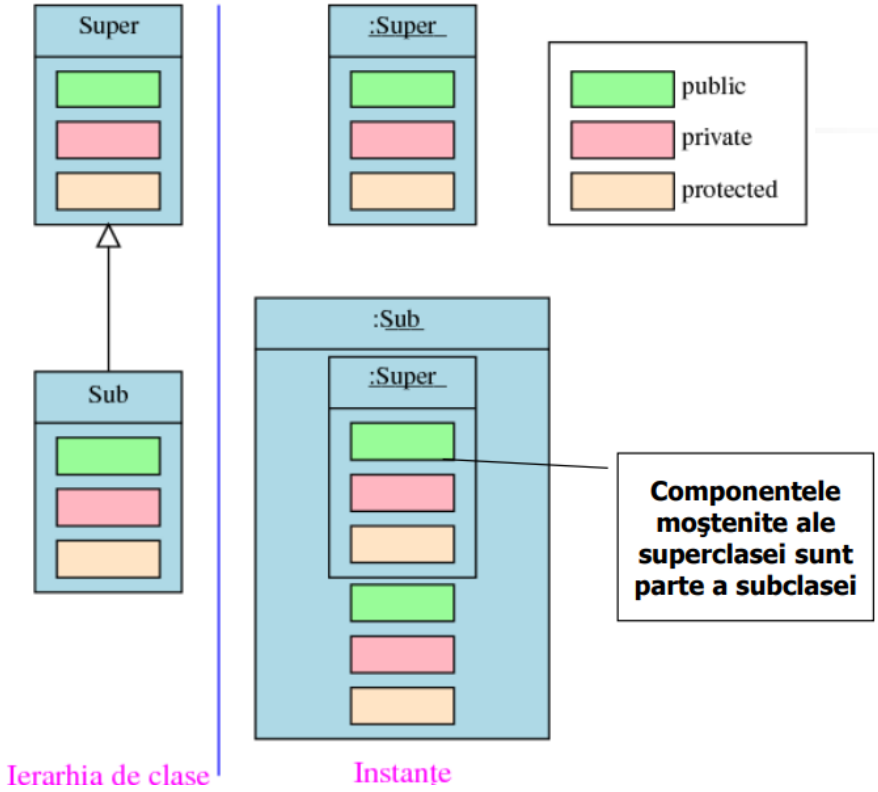

Figura 1. Ierarhia și instanțele

Avantajele moștenirii includ:

1. *Reutilizarea codului*: Subclasele pot reutiliza codul din superclasă, evitând duplicarea și promovând modularitatea.

2. *Ierarhia de clase*: Moștenirea permite crearea unei ierarhii de clase, unde subclasele pot moșteni și extinde funcționalitatea superclaselor.

3. *Polimorfismul*: Moștenirea facilitează polimorfismul, permițând obiectelor de subclase să fie tratate ca obiecte ale superclaselor.

În Java, o subclasă moștenește de la o singură superclasă, ceea ce se numește moștenire simplă. Moștenirea multiplă, în care o clasă moștenește de la mai multe superclase, nu este permisă în Java.

Superclasa

Superclasa (sau clasa de bază) este clasa din care alte clase moștenesc atributele și metodele. Superclasa definește caracteristicile comune pe care subclasele le vor moșteni și le pot extinde sau suprascrie.

În Java, o clasă devine superclasă atunci când este moștenită de o altă clasă. Superclasa poate conține atribute și metode care sunt accesibile subclaselor prin moștenire.

Cuvântul cheie `extends`

Pentru a moșteni dintr-o superclasă în Java, se folosește cuvântul cheie `extends`. Sintaxa este următoarea:

```
class SubClasa extends SuperClasa {
    // atribute și metode specifice subclasă
}
```

Figura 2. Exemplu de utilizare al “extends”

Constructorul din superclasă

Atunci când se creează un obiect al unei subclase, constructorul superclasă este apelat implicit pentru a initializa atributele moștenite. Dacă superclasa nu are un constructor fără parametri, atunci subclasa trebuie să apeleze explicit un constructor al superclasă folosind cuvântul cheie `super()` în propriul constructor.

```java
class Animal {
    private String name;

    public Animal(String name) {
        this.name = name;
        System.out.println("Constructorul Animal: " + name);
    }
}

class Dog extends Animal {
    private String breed;

    public Dog(String name, String breed) {
        super(name); // Apelăm constructorul clasei părinte (Animal)
        this.breed = breed;
        System.out.println("Constructorul Dog: " + name + ", rasa: " + breed);
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog("Bujie", "Labrador");
    }
}
```

Figura 3. Exemplu de utilizare a super()

Suprascriere (`@Override`)

Subclasele pot suprascrie (override) metodele moștenite din superclasă pentru a furniza o implementare specifică. Acest lucru se realizează prin definirea unei metode cu aceeași semnătură (nume, parametri și tip de returnare) în subclasă.

Anotația `@Override` este opțională, dar recomandată, deoarece ajută la detectarea erorilor de suprascriere accidentală și îmbunătățește lizibilitatea codului.

```java
class Animal {
    public void makeSound() {
        System.out.println("Animalul face un sunet.");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Cățelul lătră.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.makeSound(); // Afișează "Animalul face un sunet."

        Animal dog = new Dog();
        dog.makeSound(); // Afișează "Cățelul lătră."
    }
}
```

Figura 4. Notația @Override

Apelarea metodelor din superclasă

Într-o metodă suprascrisa, subclasa poate apela implementarea metodei din superclasă folosind cuvântul cheie `super` urmat de numele metodei și parametrii corespunzători.

Aceste concepte formează baza moștenirii și a relației dintre superclase și subclase în Java. Ele permit reutilizarea codului, ierarhia de clase și polimorfismul, care sunt principii cheie ale programării orientate pe obiecte.

1. Să se definească o superclasă Forma care conține o metodă arie() ce returnează 0 și o metodă afisare() care afișează „Formă geometrică generică.”

Să se creeze două clase derivate:

- Patrat, cu atributul latura și metoda arie() care returnează latura * latura.
- Cerc, cu atributul raza și metoda arie() care returnează 3.14 * raza * raza.

Ambele clase vor suprascrie metoda afisare() pentru a afișa tipul de formă. În main, se vor crea obiecte pentru fiecare formă, se vor afișa detaliile și ariile.

```java
package lab5;
class Forma {
    public double arie() {
        return 0;
    }

    public void afisare() {
        System.out.println("Formă geometrică generică.");
    }
}

class Patrat extends Forma {
    private double latura;

    public Patrat(double latura) {
        this.latura = latura;
    }

    @Override
    public double arie() {
        return latura * latura;
    }

    @Override
    public void afisare() {
        System.out.println("Formă: Pătrat, latura = " + latura);
    }
}

```

```java
public class MainForma {
    public static void main(String[] args) {
        Forma f1 = new Forma();
        Forma f2 = new Patrat(4);
        Forma f3 = new Cerc(3);

        f1.afisare();
        System.out.println("Arie = " + f1.arie());
        System.out.println();

        f2.afisare();
        System.out.println("Arie = " + f2.arie());
        System.out.println();

        f3.afisare();
        System.out.println("Arie = " + f3.arie());
    }
}
```