TEMA:MOSTENIREA.OVERRIDE.OVERLOAD.POLIMORFISMUL

**20.05.2021**

* **MOSTENIREA**

În Java, este posibil să moștenim atribute și metode de la o clasă la alta. Grupăm „conceptul de moștenire” în două categorii:

* subclasă (copil) - clasa care moștenește de la o altă clasă
* superclasă (părinte) - clasa din care este moștenită

Pentru a moșteni dintr-o clasă, utilizați cuvântul cheie „**extends”**

**SINTAXA:**

*Class* subclass-name *extends* superclass-name{

}

**Example**

class Vehicle {

protected String brand = "Ford"; // Vehicle attribute

public void honk() { // Vehicle method

System.out.println("Tuut, tuut!");

}

}

class Car extends Vehicle {

private String modelName = "Mustang"; // Car attribute

public static void main(String[] args) {

// Create a myCar object

Car myCar = new Car();

// Call the honk() method (from the Vehicle class) on the myCar object

myCar.honk();

// Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from the Car class

System.out.println(myCar.brand + " " + myCar.modelName);

}

}

Cuvântul cheie **„final”**

Dacă nu doriți ca alte clase să moștenească dintr-o clasă, utilizați cuvântul cheie **final**:

* **OVERRIDE.**

Override(suprascrierea) este o caracteristică care permite unei subclase sau unei clase copil să ofere o implementare specifică a unei metode care este deja furnizată de una dintre super-clase sau clase părinte

**Exemplu**:

**class** Parent {

    void show()

    {

        System.out.println("Parent's show()");

    }

}

// Inherited class

**class** Child *extends* Parent {

**// This method overrides show() of Parent**

    void show()

    {

        System.out.println("Child's show()");

    }

}

  class Main {

    public static void main(String[] args)

    {

        // If a Parent type reference refers

        // to a Parent object, then Parent's

        // show is called

        Parent obj1 = new Parent();

**obj1.show(); // OUTPUT:** “Parent's show()”

        // If a Parent type reference refers

        // to a Child object Child's show()

        // is called. This is called RUN TIME

        // POLYMORPHISM.

        Parent obj2 = new Child();

**obj2.show(); // OUTPUT:** “Child's show()”

    }

}

* **OVERRIDE VS OVERLOAD**

1. Ce este OVERLOAD și OVERRIDE?

Când două sau mai multe metode din aceeași clasă au același nume, dar parametri diferiți(dupa numar sau tip), se numește **Overloading.**

Când semnătura metodei (numele și parametrii) sunt aceleași în clasa superclass și în clasa copil, se numește **Overriding.**

[@Override](https://docs.oracle.com/javase/8/docs/api/java/lang/Override.html)  adnotarea informează compilatorul că elementul este menit să suprascrie un element declarat într-o superclasă

* Here is an example of overloading and overriding in a Java program.

package com.journaldev.examples;

import java.util.Arrays;

public class Processor {

public void process(int i, int j) {

System.out.printf("Processing two integers:%d, %d", i, j);

}

// **overloading method process**

public void process(int[] ints) {

System.out.println("Adding integer array:" + Arrays.toString(ints));

}

// **overloading method process**

public void process(Object[] objs) {

System.out.println("Adding integer array:" + Arrays.toString(objs));

}

}

class MathProcessor extends Processor {

@**Override**

public void process(int i, int j) {

System.out.println("Sum of integers is " + (i + j));

}

@**Override**

public void process(int[] ints) {

int sum = 0;

for (int i : ints) {

sum += i;

}

System.out.println("Sum of integer array elements is " + sum);

}

}

The process() method is overloaded in the Processor class. Then, they are overridden in the child class MathProcessor.

* **POLIMORFISMUL**

Aceiasi metoda(cu acelasi nume/parametri) executa actiuni diferite in dependent de clasa obiectului

Polimorfismul folosește aceste metode pentru a îndeplini diferite sarcini. Acest lucru ne permite să efectuăm o singură acțiune în moduri diferite.

**!!! Mereu cand rescriem metoda scrim deasupra metodei: *@Override***

Example

class Animal {

public void animalSound() {

System.out.println("The animal makes a sound");

}

}

class Pig extends Animal {

**@Override**

public void animalSound() {

System.out.println("The pig says: wee wee");

}

}

class Dog extends Animal {

public void animalSound() {

System.out.println("The dog says: bow wow");

}

}