**Programare Avansata pe Obiecte  
Laborator 3**

**Pahontu Bogdan-Ionut**

**E-mail:** [**pahontubogdan@gmail.com**](mailto:pahontubogdan@gmail.com)

Contents

[1. Abstracting 3](#_Toc3244383)

[2. Class Inheritance 3](#_Toc3244384)

[2.1. General aspects 3](#_Toc3244385)

[2.2. Extending a class 4](#_Toc3244386)

[2.3. Constructors 4](#_Toc3244387)

[2.4. super vs super() 5](#_Toc3244388)

[3. Overriding a method 5](#_Toc3244389)

[4. Overloading a method 6](#_Toc3244390)

[5. Association, Aggregation, Composition 6](#_Toc3244391)

[5.1. Association 6](#_Toc3244392)

[5.2. Composition 7](#_Toc3244393)

[5.3. Aggregation 7](#_Toc3244394)

[6. Polymorphism 7](#_Toc3244395)

[7. Object class 8](#_Toc3244396)

[8. Singleton 9](#_Toc3244397)

[9. Tasks 9](#_Toc3244398)

# Abstracting

* Interfaces;
  + The interface describes the capabilities that a class should implement.
  + An interface extends another interface;
* Abstract classes;
  + An abstract class can have both concrete methods and abstract methods
* Concrete classes;
  + A class can extend (inherit) another class (even abstract) and can implement one or more interfaces.

# Class Inheritance

## General aspects

* Java supports **single inheritance**, by which a class may inherit from only one direct parent class.
* Java also supports multiple levels of inheritance, by which one class may extend another class, which in turn extends another class. You can extend a class any number of times, allowing each descendent to gain access to its ancestor’s members. Together they are called the **members of the class.**
* It is possible in Java to prevent a class from being extended by marking the class with the **final** modifier.

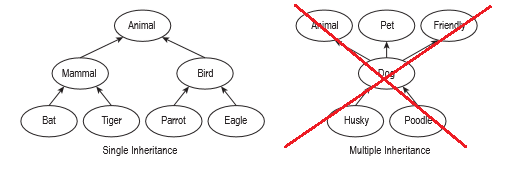


Figure 1 – Inheritance example

## Extending a class

* In Java, you can extend a class by adding the parent class name in the definition using the extends keyword. The syntax of defining and extending a class is shown below:

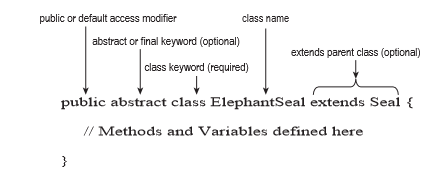


Figure 2 – Extending a class

## Constructors

* Every class has at least one constructor
* In the case that no constructor is declared the compiler will automatically insert a default no-argument constructor.
* The first statement of every constructor is either a call to another constructor within the class, using this(), or to a constructor in the direct parent class, using super().
* What happens if the parent class doesn’t have a no-argument constructor?

public class Mammal { public Mammal(int age) { } }

public class Elephant extends Mammal { // DOES NOT COMPILE }

## super vs super()

* super() is a statement that explicitly calls a parent constructor and may be only used in the first line of a constructor of a child class
* super is a keyword used to reference a member defined in a parent class and may be used throughout the child class.

**public class** Pet **extends** Animal{  
 **public** Pet() {  
 **super**();  
 System.***out***.println(**"This is the default Pet constructor"**);  
 }  
  
 **public void** showMessage(){  
 System.***out***.println(**"This is a message from Pet class"**);  
 }  
  
 **public void** displayAMessage(){  
 **this**.showMessage();  
 **super**.showMessage();  
 }  
}

# Overriding a method

* The same name for the method
* The same signature, type and number of parameters
* Return parameter the same class of a subclass
* The access modifiers, need to be the same or less restrictive
* Not final method
* It needs to throw the same exception type

# Overloading a method

* Provide functionality to reuse method name with different args
* Must have different arguments list
* May have different return types
* May have different access modifiers
* May throw different exceptions

# Association, Aggregation, Composition

## Association

* In object-oriented design, we often want to test weather an object contains a particular property or value.

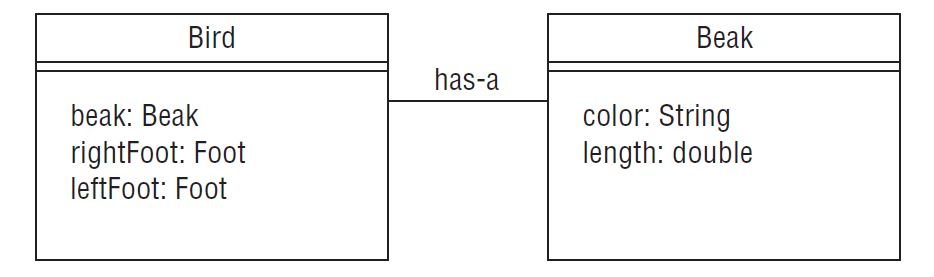
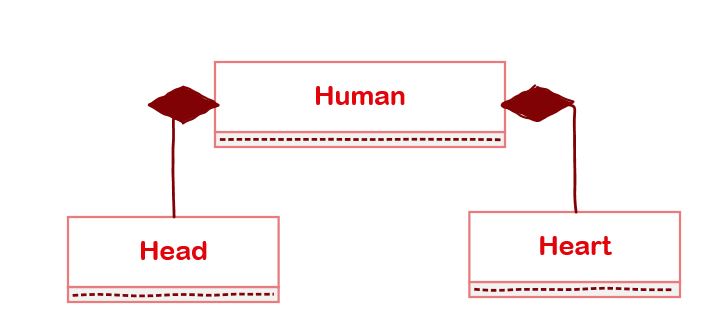


Figure 3 – Has a relationship

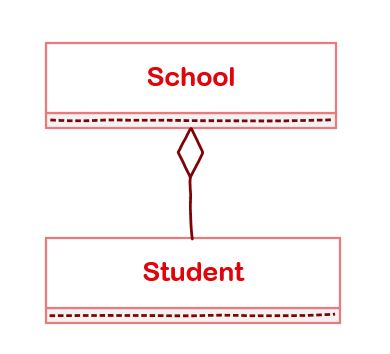
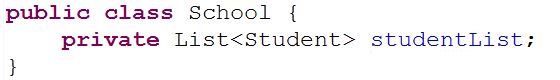
## Composition

* Association between two objects is a Composition, when one class owns the other class and the other class cannot meaningfully exist, when it’s owner is destroyed.
* You can use the final keyword to represent Composition.



## Aggregation

* Association between two objects is known as Aggregation when one class uses the other class and can still exist if one of the two is destroyed.

# Polymorphism

“The dictionary definition of polymorphism refers to a principle in biology in which an organism or species can have many different forms or stages. This principle can also be applied to object-oriented programming and languages like the Java language. Subclasses of a class can define their own unique behaviors and yet share some of the same functionality of the parent class.”

(Polimorfism, <https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html>)

**public class** Main {  
 **public static void** main(String[] args){  
 Animal animal = **new** Animal();  
 animal.makeSound();  
  
 Animal dog = **new** Dog();  
 dog.makeSound();  
  
 Animal cat = **new** Cat();  
 cat.makeSound();  
 }  
}

# Object class

* **Object** class is present in **java.lang** package. Every class in Java is directly or indirectly derived from the **Object** class. If a Class does not extend any other class then it is direct child class of **Object** and if extends other class then it is an indirectly derived.
* Methods:
  + toString()
  + hashCode()
  + equals(Object obj)
  + getClass()
  + clone()

# Singleton

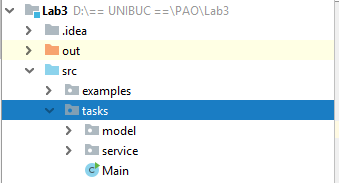
* A class has a single instance at a moment on the VM level.

**public class** MyStaticSingleton {  
  
 **private static** MyStaticSingleton *instance* = **new** MyStaticSingleton();  
  
 **private** MyStaticSingleton() {  
 *// private constructor* }  
  
 **public static** MyStaticSingleton getInstance() {  
 **return** *instance*;  
 }  
}

# Tasks

**Task 1:**

1. Create packages lab3 -> tasks
2. Create two packages:
   1. lab3 -> tasks -> model
   2. lab3 -> tasks -> service.

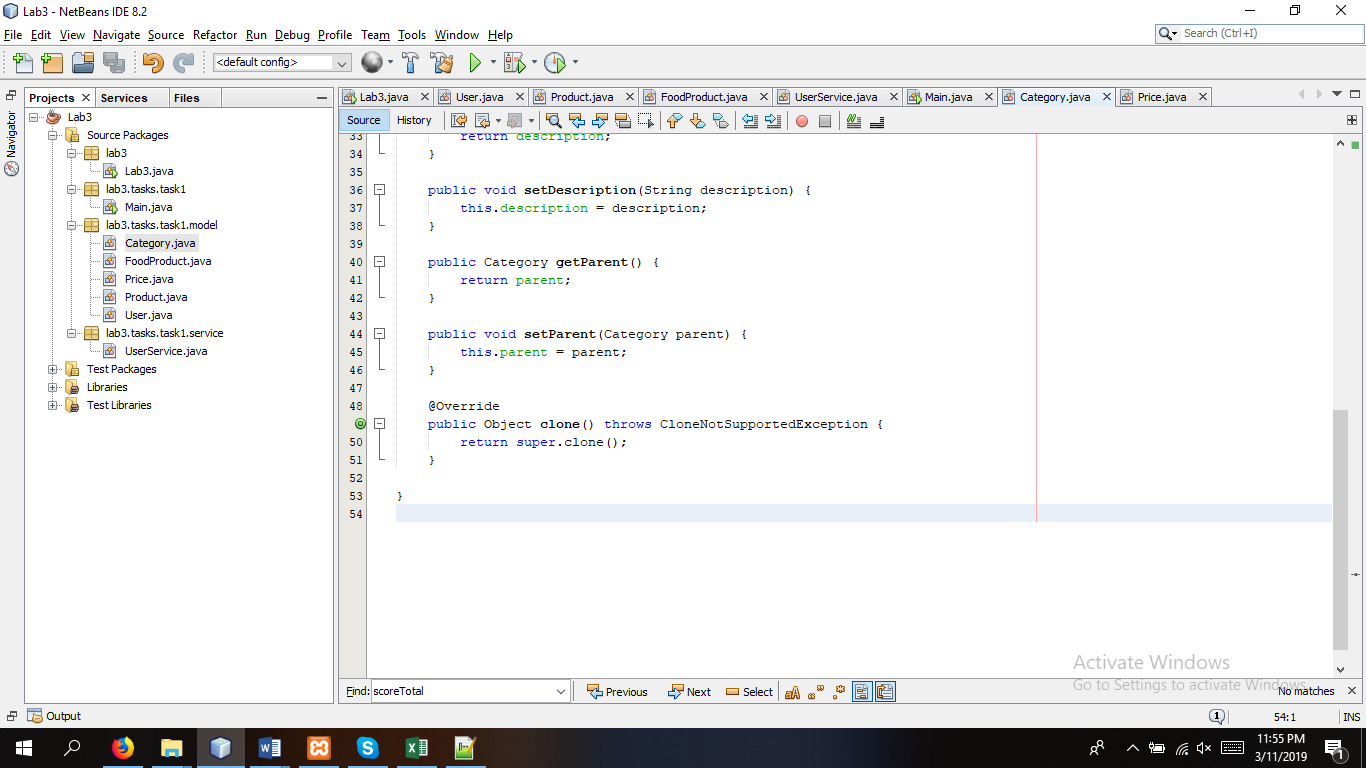


1. In package model create the following classes:
   1. **Class Category** with private fields:
      1. String **name**;  
         String **description**;  
         Category **parent**;
   2. **Class Price** with private fields:
      1. Double **price**;  
         Date **startFrom**;  
         Date **endFrom**;
   3. **Class Product** with private fields:
      1. Category **category**;  
         Price **price**;  
         String **name**;  
         String **smallDescription**;
   4. **Class User** with private fields:
      1. Integer **id**;

String **username**;  
String **password**;

Implement getter and setter for each of the above classes.

1. Create class FoodProduct that inherits class Product and overwritten the getName method.
2. In package service create a singleton class named UserService.
3. For each class from model package implement a clone method in parent class Object.



1. Classes from package model should implement the interface: Cloneable.

**TASK 2:**

1. Test in class Main a Product object.

**Product product = new Product();  
product.setName("Product one");**

1. Create an instance of FoodProduct Object.

**FoodProduct foodProduct = new FoodProduct();  
foodProduct.setName("Product two");**

1. Call the getName methods for the two objects and compare the results.
2. Create an instance of Product class with a FoodProduct class.

**Product foodProduct = new FoodProduct();**

**foodProduct.setName(„foodProduct”);**

**TASK 3:**

In class UserService from the package service try to delete th empty constructor and add the following code:

private static User[] *listOfUsers* = new User[4];  
private UserService()

{  
 *listOfUsers*[0] = new User(1, "Ioana", "password");  
 *listOfUsers*[1] = new User(2, "Adelin", "password");  
 *listOfUsers*[2] = new User(3, "Marcu", "password");  
 *listOfUsers*[3] = new User(4, "Doru", "password");

}

* ! Do not forget to add in class User a constructor with parameters.

public User getOne(Integer id)

{

//TODO implement - to find user by id

}  
  
public User getOne(String username)

{  
 // strings are using equals for comparing

//TODO implement - to find user by username  
}

**TASK 4:**

Similarly, create a ProductService class in the package serivce.