**Programare Avansata pe Obiecte  
Laborator 5**

**Pahontu Bogdan-Ionut**

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

Contents

[1. Interfaces (Part 1) 3](#_Toc4451271)

[1.1. What about interfaces? 3](#_Toc4451272)

[1.2. Defining an interface 3](#_Toc4451273)

[1.3. Implementing interfaces 4](#_Toc4451274)

[1.4. Inheriting an interface / Multiple inheritance 4](#_Toc4451275)

[1.5. Interface variables 5](#_Toc4451276)

[1.6. Default methods 5](#_Toc4451277)

[1.7. Static interface methods 7](#_Toc4451278)

[2. Tasks 7](#_Toc4451279)

# Interfaces (Part 1)

## What about interfaces?

* Although Java doesn’t allow multiple inheritance, it does allow classes to implement any number of interfaces. An interface is an abstract data type that defines a list of abstract public methods that any class implementing the interface must provide;
* An interface can also include a list of constant variables and default methods;
* In Java, an interface is defined with the **interface** keyword, analogous to the class keyword used when defining a class. A class invokes the interface by using the **implements** keyword in its class definition;

## Defining an interface

* **Abstract** and **public** are assumed. In other words, whether or not you provide them the compiler will automatically insert them as part of the method definition;

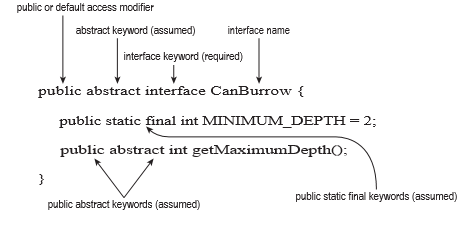


Figure 1 – Defining an interface

* Rules for creating an interface:
  + Interfaces cannot be instantiated directly;
  + An interface is not required to have any methods;
  + An interface may not be marked as final;
  + All top-level interfaces are assumed to have public or default access. Therefore, marking an interface as private, protected, or final will trigger a compiler error, since this is incompatible with these assumptions;
  + All non-default methods in an interface are assumed to have the modifiers abstract and public in their definition. Therefore, marking a method as private, protected, or final will trigger compiler errors as these are incompatible with the abstract and public keywords;

## Implementing interfaces

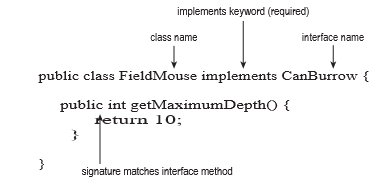


Figure 2 – Implementing an interface

## Inheriting an interface / Multiple inheritance

* A class can implement an interface, a class cannot extend an interface.
* An interface can extend another interface.
* Please see the example in lab5.examples.InheritingEx package:
  + In this example we have three interfaces and a class Furniture that implements all three interfaces;
  + What about method “getPropertyName” that is defined in each interface. Why is working ok?
  + Remember that interface methods in this example are abstract and define the “behavior” that the class implementing the interface must have. If two abstract interface methods have identical behaviors—or in this case the same method signature— creating a class that implements one of the two methods automatically implements the second method.

* What if the method will have different signatures? Please see
  + lab5.examples.InheritDifSign
  + lab5.examples.InheritWrongDifSign
    - Reason: It is not possible in Java to define two methods in a class with the same name and input parameters but different return types.

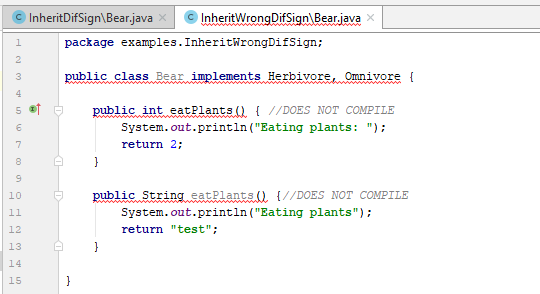


Figure 3 – Implementing multiple interfaces – same method signature

## Interface variables

* Like interface methods, interface variables are assumed to be public.
* Unlike interface methods, though, interface variables are also assumed to be static and final.
* Rules:
  + Interface variables are assumed to be public, static, and final;
  + The value of an interface variable must be set when it is declared since it is marked as final;

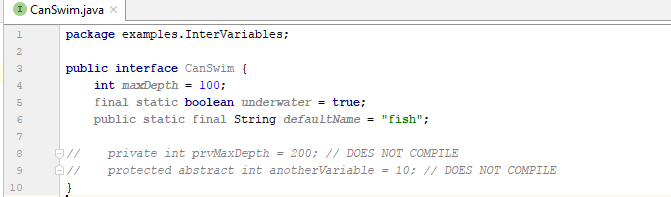
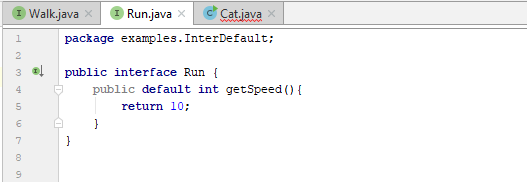
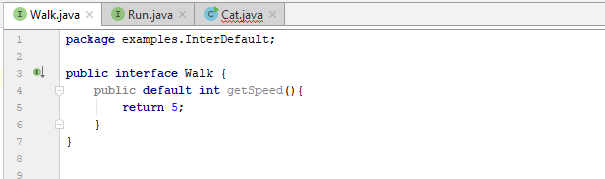


Figure 4 – Interface variables

## Default methods

* In Java 8 authors have introduced a new type of method to an interface, referred to as a default method;
* A **default method** is a method defined within an interface with the **default** keyword in which a method body is provided;
* A default method within an interface defines an abstract method with a default implementation.
* Classes have the opinion to override the default method if it`s needed, but not required to do so.
* Rules to define default interface methods:
  + A **default method** may only be declared within an **interface** and not within a class or abstract class;
  + A default method must be marked with the **default** keyword. If a method is marked as default, it **must provide a method body;**
  + A default method is **not assumed to be static, final, or abstract**, as it may be used or overridden by a class that implements the interface;
  + Like all methods in an interface, a default method is **assumed to be public** and will not compile if marked as private or protected;
  + When an interface extends another interface that contains a default method:
    - it may choose to ignore the default method.
    - the interface may override the definition of the default method;
* The following example illustrates interfaces with default methods and multiple implementation





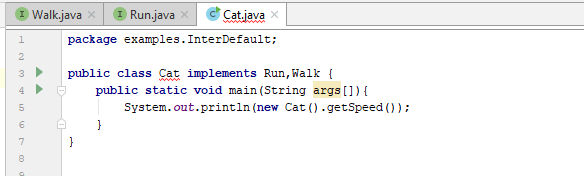
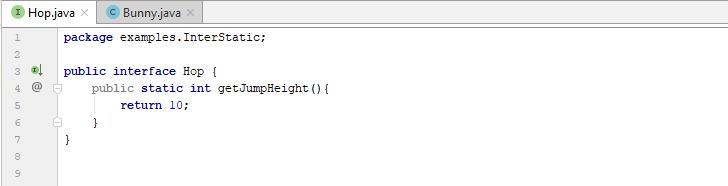


Figure 5 – Class implementing two interfaces

## Static interface methods

* Java 8 also now includes support for static methods within interfaces;
* Rules:
  + Like all methods in an interface, a static method is assumed to be public and will not compile if marked as private or protected;
  + To reference the static method, a reference to the name of the interface must be used;



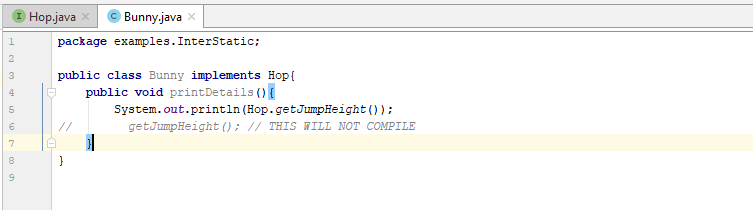


Figure 6 – Static interface methods

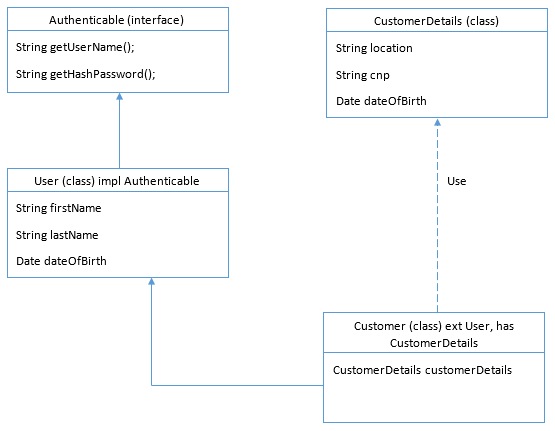
# Tasks

**Task 1:**

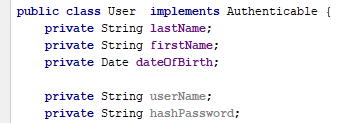
1. Create packages lab5 🡪 tasks
2. Create a package lab5 🡪 tasks🡪 shapeDemo;
3. Create an interface NamedObject with the following methods;
   1. getName();
   2. getDescription();
4. Create an interface named Shape with two methods:
   1. getArea();
   2. getPerimeter();
5. Create a class Rectangle that implements Shape interface;
6. Class rectangle will have the following:
   1. Double sideA;
   2. Double sideB;
   3. Constructor with two parameters;
   4. Methods that needs to be implemented from Shape interface;
7. Create a class Square that extends class Rectangle;
8. Square class will have a constructor with one single parameter;
9. Create a class named Triangle that implements Shape interface;
   1. Double sideA; Double sideB; Double sidec;
   2. Constructor;
   3. Methods that needs to be implemented from Shape interface;
10. Implement a class CustomTriangle that extends Triangle and implements NamedObject with the following;
    1. String name;
    2. String description;
    3. Constructor;
    4. Methods that needs to be implemented from NamedObject interface;
11. Create a main class that uses the previous classes/interfaces;

**Task 2:**

1. Create the following structure:



1. The classes should be distributed into packages like below:
   1. Authenticable will be in auth package;
   2. CustomerDetails and Customer will be placed in customer package
   3. User will be placed in model package
2. For Authenticable interfaca add one field public static final String DEFAULT\_TOKEN and a method getToken();
3. For Customer implement setter and getter for customerDetails property
4. For User class implement all the necessary methods , setters, getters, etc..



1. Add a package named services;
2. Inside this package add a new class UserService with method checkDefToken(Authenticable auth) that checks that returns true if the default token is “TEST”;
3. Create a main class that uses the previous classes/interfaces;
4. Choose one of the interfaces implemented earlier and implement static methods and default methods that will be relevant for the entire app structure.

**Task 3:**

interface Enclosure {

public double perimeter();

public double area();

}

class Circle {

double radius;

Circle(double radius) {

this.radius = radius;

}

}

class Square {

double width;

Square(double width) {

this.width = width;

}

}

1. Starting from the previous structure, modify the Circle and Square class in order that both implement Enclosure interface;
2. Create a list with circles and squares and display the perimeter/area for each one;