Table of Contents

[1. Abstractizare 2](#_Toc3076137)

[1.1. Interfață 2](#_Toc3076138)

[1.2. Clasa abstractă 2](#_Toc3076139)

[1.3. Clasă 2](#_Toc3076140)

[2. Încapsulare și moștenire 3](#_Toc3076141)

[3. Polimorfism 3](#_Toc3076142)

[4. Clasa Object 4](#_Toc3076143)

[5. Asociere, Agregare, Compoziție 4](#_Toc3076144)

[6. Singleton 5](#_Toc3076145)

[1. Referințe 6](#_Toc3076146)

# Abstractizare

Abstractizarea se realizează prin folosirea

* Interfeței
* Clasei abstractă
* Clasei concretă

## Interfață

„In the Java programming language, an interface is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces.”

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

O subcategorie a interfețelor sunt interfețele marker. Exemple de interfețe marker

Cloneable, Serializable.

## Clasa abstractă

„An abstract class is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be sub-classed.”

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

## Clasă

“In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an instance of the class of objects known as bicycles. *A class is the blueprint from which individual objects are created.*”

(<https://docs.oracle.com/javase/tutorial/java/concepts/class.html> )

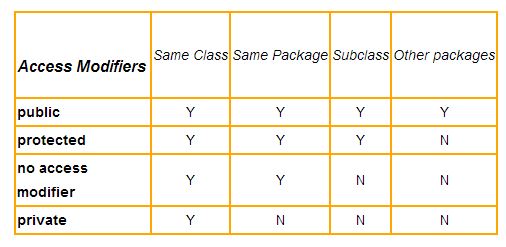
În câteva cuvinte:

* interfață descrie capabilitățile pe care o clasă ar trebui să le implementeze.
* clasă poate extinde (moșteni) o altă clasă (chiar și abstractă) și poate să implementeze una sau mai multe interfețe.
* interfață poate sa extindă o altă interfață.
* clasa abstractă poate avea metode concrete dar și metode abstracte (keyword-ul abstract este obligatoriu acum față de interfețe).

# Încapsulare și moștenire

Sunt patru categorii de modificatori:

* Default
* Public
* Private
* Protected



Extends and implements

# Polimorfism

“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>)

În cadrul proiectului nostru clasa Product și FoodProduct.

*// in general nu folositi nume de variabila de acest fel, este urat si contraindicat*Product foodProduct\_2 = **new** FoodProduct();  
foodProduct\_2.setName(**"foodProduct\_2"**);

Reguli pentru overriding:

* 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
* Only inherited methods can be overwritten
* It needs to throw the same exception type

Overloading

* 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

# Clasa Object

Clasa Object stă la baza oricărei clase din java. Este o clasă ce vine cu un număr minimal de funcționalități precum:

clone

hashCode

equals

finalize

Relația dintre equals și hashCode

a.equals(b) => a.hashCode() == b.hashCode()

# Asociere, Agregare, Compoziție

Compoziția – ex Product și name

Agregare – ex Product și Category

Amândouă răspund la întrebarea “has”

Asocierea este cea mai slabă dintre relații, ex. UserService folosește User, dar nu depinde existențial de el.

Moștenirea răspunde la întrebarea “is a”

Product

FoodProduct

# Singleton

O clasă ce are o singura instanță la un moment dat în cadrul mașini virtuale.

*Singleton implementat static*

Cel mai simplu singleton

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

*Bill Pugh Singleton*

Rezolvă problema accesării în mediu multi-threading chiar și pe mașini virtuale mai vechi jdk < 1.5.

**public class** BillPughSingleton {  
  
 **private** BillPughSingleton(){}  
  
 **private static class** SingletonHelper{  
 **private static final** BillPughSingleton ***INSTANCE*** = **new** BillPughSingleton();  
 }  
  
 **public static** BillPughSingleton getInstance(){  
 **return** SingletonHelper.***INSTANCE***;  
 }  
}

# MVC

Processing

Model

(User, Product …)

Controller

(Main)

View

(nothing for now,

or maybe the console)

Response

Result

# Referințe

1. Polimorfism, <https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html>
2. Class Object, <https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>
3. Thinking in Java, 4th Edition, Bruce Eckel (pdf free pe google)
4. Singleton, <https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples>