**Workshop**

Topics:

1. OOP
2. Immutable Objects
3. Exceptions
4. Extra Project

Prerequisites:

1. IntelliJ IDEA
2. Java SDK 7/8

OOP Exercises:

Create a new java project with IDEA and JDK 7.

1. In the new project create a ro.teamnet.zerotohero.oop.graphicshape **package**.
2. Create a new class named **Shape** with a public ***area()*** method, that returns a **double** value.As fields you should have one primitive data field of type **int**, named ***color***, and one primitive of type **float**, named ***saturation***. Give the ***area()*** method a default implementation. For ***color*** and ***saturation*** create setter methods.
3. Place the Shape class, in ro.teamnet.zerotohero.oop.graphicshape package.
4. Make the ***color*** and ***saturation*** fields, of the class **Shape**, **protected** (we will test the protected access modifiers property).
5. Create a new class **Circle**, in ro.teamnet.zerotohero.oop.graphicshape package. In this class create three **private** data fields of type **int**: ***xPos***, ***yPos*** and ***radius***, and a constructor with no parameters in which the three primitives are initializes with values (of you choice).
6. Create a new interface, named **ShapeBehaviour**, in ro.teamnet.zerotohero.oop.graphicshape package, with a ***area()*** method that returns a **double** value.
7. Now let’s use the interface. Make the class **Shape** implement **ShapeBehaviour**.
8. Create a abstract class **AbstractShape**, in ro.teamnet.zerotohero.oop.graphicshape package, with a ***area()*** method that returns a **double** value.
9. As with interfaces we can use abstract classes by extending them and inheriting their properties and functionalities. Make the **Shape** class extend **AbstractShape**.
10. Inheritance can be used by extending a concrete class as well. Make **Circle** class extend **Shape** class and override the **area()** method so that it calculates the circle area. Import java.lang.Math.PI with static import for that calculation.
11. Create a new class **Square** that extends the **Shape** class with a **private** **int** primitive named ***side*** and a constructor that takes one parameter which initializes the ***side*** variable. Overwrite the ***area()*** method so that it calculates the square area.
12. Create a new class named **Circles** with a method ***getAreaPub()*** with **double** return type. In this method create a new **Circle** object and return its area (call to public method ***area()***, within package).
13. Create a runapp package in ro.teamnet.zerotohero.oop. In this package create a new class named **RunApp** with a **main(String[] args)** method. In this method create a new **Circles** object and print the default circle area by caling ***getAreaPub()*** of the Circles object (The message printed should be something like ‘The default circle area is 2827.4333882308138’).
14. Overload the **Circle** class constructor by creating three more constructors with one, two, and three parameters.
15. Every class extends the **Object** class which has a default ***toString()*** method. Overwrite the ***toString()*** method in the **Cirlce** class in order to return the ***xPos***, ***yPos*** and ***radius*** in the following format: “center = (x,y) and radius = z”. The **toString()** method prototype is: public String toString();
16. In the **Circle** class create an overloaded method named ***fillColour()****.* One method should have no parameters, one with an int parameter and one with a float parameter. The ***fillColour*** method with no parameters should print the super class ***color*** primitive variable. The ***fillColour*** method with an int parameter should set the super class ***color*** primitive variable and print a message after. The message should be like "The circle color is now 2". The ***fillColour*** method with a float parameter should set the super class ***saturation*** parameter.
17. In class **Circles** create a method named ***getAreaDef()*** with void return type. In this method create a new **Circle** object and call the three ***fillColour*** methods.
18. To use the ***getAreaDef()*** call it in the main method of the **RunApp** class.
19. Create a canvas package in in ro.teamnet.zerotohero and in it create a **Canvas** class. In this class create a method ***paint ()*** that should print the following message: “I should be an artist”. The ***paint()*** method should have the default access modifier. From the main method of the **RunApp** class create a new **Canvas** object and try to call the ***paint()*** method. Observe that the compiler won’t let you do so.
20. Now lets test the runtime polymorphism. In the ***main*** method of the **RunApp** class create a new **Shape** object and initialize it with a new **Circle(10)** object. Print the new **Shape** object area. Create a **ShapeBehaviour** object by instantiating it with a new **Square**(10) object. Print the new **Square** object area.
21. Let’s dive deeper into overwriting. Create a new **Point** class in

ro.teamnet.zerotohero.oop.graphicshape package. This method should have two primitives **int** variables ***xPos*** and ***yPos***. Create a constructor, for the **Point** class, with two parameters, that initializes the two data variables. Another method inherited from the **Object** class is ***equals***(). Let’s overwrite it by creating a new method equals like so :

// override the equals method to perform "deep" comparison of two Point objects

@Override

public boolean equals(Object other) {

if(other == null)

return false;

// check if the dynamic type of 'other' is Point

// if 'other' is of any other type than 'Point', the two objects cannot be

// equal if 'other' is of type Point (or one of its derived classes), then

// downcast the object to Point type and then compare members for equality

if(other instanceof Point) {

Point anotherPoint = (Point) other;

// two points are equal only if their x and y positions are equal

if( (xPos == anotherPoint.xPos) && (yPos == anotherPoint.yPos) )

return true;

}

return false;

}

Create, in the main method of the **RunApp** class, three objects and instantiate them with **Point** values :

Object p1 = new Point(10, 20);

Object p2 = new Point(50, 100);

Object p3 = new Point(10, 20);

Now if you type:

System.out.println("p1 equals p2 is " + p1.equals(p2));

System.out.println("p1 equals p3 is " + p1.equals(p3));

This code will print :

p1 equals p2 is false

p1 equals p3 is true

1. In inheritance the subclass can call the super classes constructor by using the super keyword. Create a new class, in ro.teamnet.zerotohero.oop.graphicshape, named **Point3D**, that extends **Point** class, with a private **int** ***zPos*** and a constructor with three parameters. The first two parameters will be used in the call of the super classes constructor like so ***super(x, y)*** and the third will be used to set the ***zPos***;

Immutable Object Exercises:

1. Create a new immutable class and test its immutability.

Exception handling:

1. Define a new exception.
2. Create a new example of throwing an exception by using your newly defined exception.
3. Create an example of exception propagation.
4. Create an example of nested exceptions in which the newly thrown exception.
5. Create an example in which you make use of try/catch.
6. Create an example in which you make use of try/catch with resources.
7. Create an example in which you make use of try/catch/finally.
8. Create an example in which you make use of try/finally.
9. Create an example in which you make use of multi-catch.
10. Create an example in which you make use of catch with multiple exceptions.

System.***out***.println("Text that will be printed in console");

Project:

Pentru proiect clasele se vor pune in pachete cu denumiri sugestive.

For the project, all the classes should be created in packages with specific and sugestive names.

Sa se creeze doua exceptii : AnimalManancaOmException care este o exceptie de tip unchecked si AnimalPeCaleDeDisparitieException care este o exceptie de tip checked.

Create two exceptions: AnimalEatingManException which is an unchecked exception and AnimalExctinctException which is checked exception.

Sa se creeze interfata AngajatZoo cu metodele:

* void: lucreaza(Animal animal);

Create the ZooEmployee interface with the following methods:

* void:work(Animal animal);

Create the abstract class Animal with the following methods:

* abstract method **eat**, which has one parameter of type **Object**
* abstract method **play**, no parameters;
* abstract method **takeBath**, no parameters;
* **sleep** method , which is not an abstract and will print a text in console “Animal sleeps”

Sa se creeze clasa abstracta Animal cu metodele:

* Metoda abstracta ***mananca***, care are un singur parametru de tip **Object**
* Metoda abstracta ***seJoaca***, fara parametrii;
* Metoda abstracta ***faceBaie***, fara parametrii;
* Metoda ***doarme*** , care nu este abastracta si va afisa text in consola **“Animalul doarme”.**

In the Animal class constructor print in console “New Animal!”.

In cadrul constructorului Animal sa se scrie text in consola “Animal nou”.

Create the RareZooAnimal class that extends the Animal class. Keep the encapsulation principle for the following fields of RareZooAnimal, **name** and **countryOfOriginName**. Implement all the constructors for the RareZooAnimal.

In **takeBath** method implementation print the following text in console “RareAnimal taking bath”.In **play** method implementation print the following text in console “RareAnimal plays” and then call by using the super the method **sleep** from the parent.In the **eat** method in case the object recieved as a parameter is a ZooEmployee throw a new AnimalEatingManException, if not print in the console “RareZooAnimal is eating!”.

Sa se creeze clasa AnimalZooRar care extinde clasa Animal. Sa se pastreze principiul encapsularii datelor pentru aceasta clasa, pentru a tine informatii despre ***numele si numeleTariiDeOrigine*** despre AnimalZooRar.Sa se implementeze toti constructorii clasei.

In implementarea metodei ***faceBaie*** sa se afiseze text in consola “AnimalulZooRar face baie”. In cadrul clasei AnimalZooRar, in implementarea metodei ***seJoaca*** sa se sa se afiseze text in consola “AnimalulZooRar se joaca” si apoi sa se apeleze din parinte metoda ***doarme***. In cadrul metodei ***mananca,***in cazul in care obiectul primit ca parametru este un AngajatZoo sa se arunce exceptia *AnimalManancaOmException* daca nu, sa se afiseze mesajul in consola “AnimalZooRar mananca”.

Create the FerociousZooAnimal that extends the Animal class. Print in console sugestive text for all the method implementations that will be overriden from the parent. Override the **sleep** method from the parent where you will use both the super implementation for this method but also will print a new text in console “The ferocious animal hunts”. It should be taken care of that an FerociousZooAnimal should not eat a ZooEmployee, the same as in the previous class.

Sa se creeze o clasa AnimalZooFeroce care extinde clasa Animal. Sa se afiseze text in consola in implementarile metodelor mostenite de la parinte. Se va suprascrie metoda ***doarme*** din parinte unde se va folosi atat implementarea din clasa Animal (super) a acestei metode dar se va afisa si un nou text in consola , “Animalul feroce vaneaza”.Se va avea grija ca AnimalulZooFeroce sa nu manance un AngajatZoo, ca in exemplul precedent.

Create the ZooKeeper class that implements the ZooEmployee interface. In the implementation of the **work** method you will print the following text “The keeper enters the animal cage”. Overload the **work** method so that it can recieve another parameter, named food of Object type. In the implementation of work call the **work** method from the curent object using this and call all the methods from the Animal recieved as a parameter.In the case that the Animal recieved as a parameter is null the **ExtinctAnimalException** will be thrown and will be added to the method signature.

Sa se creeze clasa IngrijitorZoo care implementeaza interfata AngajatZoo. In implementarea metodei ***lucreaza*** se va afisa text in consola “Ingrijitorul intra in cusca aninalului”.

Sa se supraincarce metoda ***lucreaza*** astfel incat sa primeasca inca un parametru, ***mancare*** de tip Object, iar in implementarea acestei metode sa se apeleze ***lucreaza***din obiectul curent, precum si toate metodele din clasa Animal,metodei ***mananca*** i se va pasa parametru de tip Object.

In cazul in care Animalul primit ca parametru este de tip AnimalZooRar si obiectul mancare primit ca parametru este null se va arunca exceptia AnimalPeCaleDeDisparitieException, exceptia va fi adaugata la semnatura metodei.

Sa se creeze clasa VeterinarZoo care implementeaza interfata AngajatZoo. In metoda ***lucreaza*** se afiseaza in consola “Veterinarul are grija de animal” iar in cazul in care Animalul primit ca parametru la metoda ***lucreaza*** este un AnimalZooFeroce, atunci se va apela metoda ***faceBaie***.

Sa se creeze o clasa GradinaZooMain unde se va implementa metoda

**public static void main(String[] args){}**

In aceasta metoda se scrie urmatoarea secventa de cod.

AnimalZooRar animal1 = new AnimalZooRar(“Pinguin”);

AnimalZooRar animal2 = new AnimalZooRar(“Elefant”,”Africa”);

AnimalZooRar animal3 = new AnimalZooRar();

AnimalZooFeroce animalFeroce = new AnimalZooFeroce();

AngajatZoo angajat1 = new IngrijitorZoo();

IngrijitorZoo angajat2 = new IngrijitorZoo();

AngajatZoo angajat3 = new VeterinarZoo();

VeterinarZoo angajat4 = new VeterinarZoo();

//aici se vor afisa informatiile despre animal1 si animal2 (nume si tara)

//System.out.println(animal1.getNume());

//apelul metodelor

angajat3.lucreaza(animal1);

angajat3.lucreaza(animal2);

angajat3.lucreaza(animal3);

angajat4.lucreaza(animal1);

angajat4.lucreaza(animal2);

angajat4.lucreaza(animal3);

angajat1.lucreaza(animal1);

angajat1.lucreaza(animal2);

angajat1.lucreaza(animal3);

angajat2.lucreaza(animal1);

angajat2.lucreaza(animal2);

angajat2.lucreaza(animal3);

angajat2.lucreaza(animal1,null);

angajat2.lucreaza(animal1,angajat1);

angajat2.lucreaza(animal1,new String(“Mancare”));

angajat2.lucreaza(animalFeroce);

angajat2.lucreaza(animalFeroce,null);

angajat2.lucreaza(animalFeroce, new String(“Mancare”));

System.out.println(“Finish!”);

Sa se modifice codul prind adaugare fie de structuri try/catch/finally fie de alte metode din interiorul claselor pentru a ajunge la punctul “Finish” fara erori de compilare sau fara erori de runtime.

Sa se adauge un camp de tip Integer in interfata AngajatZoo , **valoareBonusPerAnimal** care va fi egal cu 50, sa se adauge o noua metoda in interfata AngajatZoo, ***calculeazaBonusSalarial***. Sa se creeze o logica in clasele VeterinarZoo si IngrijitorZoo pentru metoda ***calculeazaBonusSalarial*** astfel incat dupa realizarea cu succes a actiunii de ***lucreaza*** cu un animal, sa se calculeze un bonus salarial care sa fie salvat pentru fiecare angajat si care se poate sa fie afisat in consola.Pentru IngrijitorZoo se va calcula 3 \* bonus pentru fiecare animal iar pentru VeterinarZoo 2 \* bonus pentru fiecare animal.

Sa se modifice metoda mananca astfel incat animalele sa nu manance alte animale. (asemenea angajatilor)

(Se pot adauga campuri noi, metode noi, interfete noi,exceptii noi, clase abstracte noi, totul tine de imaginatia si modul in care vedeti ca se poate implementa acest punct doar ca sa se respecte conceptele OOP de mosteniri, DRY si encapsulare)

Sa se creeze clasa GradinaZoo , cu urmatoarele campuri:

* String : denumireGradinaZoo;
* Date: dataDeschideriiGradinii;
* AnimalZooRar : animalRar;
* IngrijitorAnimale: angajatulLunii;

Sa se particularizeze clasa GradinaZoo pentru a fi immutabila. ATENTIE la campurile care sunt mutabile.