Department of Informatics

Professor: R. Ingold
Teaching Assistants: M. Alberti, L. Vögtlin
Assistant: R. Geissbühler

S2019 Object Oriented Programming

Assignment 04

1 Theory

In this exercise you have to answer the following theoretical questions. Keep your answers short and precise.

(a) What is an abstract class? When does it make sense to use one? Additionally, make a concrete example of a situation where you cannot solve the problem without using an abstract class.

(b) What is an abstract methods? Where does it make sense to use one? Additionally, make a concrete example of a situation where you cannot solve the problem without using an abstract method.

(c) What are the conditions a class has to fulfill for having an abstract method?

(d) What is an interface? When does it make sense to use one? Additionally, make a concrete example of a situation where you cannot solve the problem without using an interface.

(e) What are the keywords `extend` and `implement` used for?

(f) How many interfaces can a class implement?

(g) How many abstract classes can a class extend from?

(h) Is it possible to instantiate an abstract class or an interface?

(i) What is the default visibility of fields and methods of an interface? Can you modify their visibility? If yes, how?

(j) What is the default visibility of fields and methods of an abstract class? Can you modify their visibility? If yes, how?

(k) What default properties do fields of an interface have? Can you modify them? If yes, how?

(l) What default properties do fields of an abstract class have? Can you modify them? If yes, how?

2 Implementation

For this exercise you are required to submit four files: Person.java, Employee.java, MotivatedEmployee.java, Worker.java, Manager.java and Main.java, implemented as described below.

(a) Define an abstract class `Person` with protected fields for name and a method `printName()` which just prints the name to console. Define the constructor accordingly.

(b) Define an interface `Employee` which exposes the method `printSalary()`

(c) Define an interface `MotivatedEmployee` which extends `Employee` and exposes the method `printBonus()`

(d) Implement a class `Worker` which extends from `Person` and implements `Employee` with private field for salary. Override the missing methods with a reasonable implementation (the name of the methods is quite self-explanatory). Define constructor and other methods of the class as necessary.

(e) Implement a class `Manager` which extends from `Person` and implements `MotivatedEmployee` with private field for salary and bonus. Override the missing methods with a reasonable implementation (the name of the methods is quite self-explanatory). Define constructor and other methods of the class as necessary.

(f) Implement a `Main` class with a `main()` method which create some `Worker` and some `Manager` objects and adds them to a list. Iterate over such list and print the salaries of each item. *Hint: mind the type of the list!*.

3 Debugging

In this exercise we want to practice the use of the debugger and familiarize with the error messages of the compiler. Refer to the files `Main.java` and `Tuple.java` attached to this series for this exercise. **For this exercise you are NOT allowed to modify the given code in any way**[1].

(a) Report the *exact* output if you were to run `Main()`

(b) Notice how the field `b` of the tuples is unknown (as its not printed). It is not a bug, it is a feature of the code.

(c) Report the *exact* values of the field `b` for all elements in the list. Since it is a private field and you are NOT allowed to modify the code you will need to find another way to get this information: the debugger.

[1] For those who know what reflection is: you are not allowed to use that either.

3.1 The Debugger

A debugger is a program that can examine the state of your program while your program is running. In our case, it will allow us to look into the field **b** of our tuples without making them public or printing it. Get familiar with the debugger of your IDE. I gathered for you some tutorial as a starting point:

- IntelliJ
  http://bit.ly/2oTVnKa
  https://www.jetbrains.com/help/idea/evaluating-expressions.html
- NetBeans
  https://www.youtube.com/watch?v=joWldbcp1So
  http://bit.ly/2I6KHRw
- Eclipse
  http://bit.ly/2qb2K3q
  http://bit.ly/2tkBO3r

You don't have to use this guides, chose your favourite one on the web. The required skills that you have to learn are:

- run a program in debug mode
- set a breakpoint
- go trough the execution step-by-step
- inspect the current status of a variable
- correctly use "watch expressions".

These skills are fundamental for programming and are agnostic not only to the different IDEs but also to different languages. In fact, you will find that all major IDEs offer a good debugger in a very similar way and for multiple languages.

For reference (you don't have to rewrite them yourself, we provide the java files attached to the series):

```java
package core;

import java.util.Random;

public class Main {
    public static void main(String[] args) {

        Random random = new Random(42L);

        Tuple[] list = new Tuple[5];

        for (int i = 0; i < list.length; i++) {
            list[i] = new
               Tuple(random.nextInt(10),random.nextInt(10)+10);
        }

        for(Tuple t: list) {
            System.out.println(t.getA());
        }

    }
}
```

```java
package core;

public class Tuple {
    private final int a;
    private final int b;

    public Tuple(int a, int b) {
        this.a = a;
        this.b = b;
    }

    public int getA(){
        return a;
    }
}
```

4 Bonus exercise

This is a bonus exercises. Provided you solve it properly, it can recover previous assignment shall you have failed it. Otherwise it would contribute to have a pass in this assignment.

In this exercise you have the following code:

```
1 public class A {
2     public String message;
3     public A(String m) {
4         message = m;
5     }
6     public int number(int n) {
7         return n;
8     }
9 }
```

```
1 public class B extends A {
2     public B(String m) {
3         super("Message: "+m);
4     }
5 }
```

```
1 public class C extends A {
2     public C(String m) {
3         super(m.toUpperCase());
4     }
5     public int number(int n) {
6         return 2*super.number(n);
7     }
8 }
```

```
1 public class D extends B {
2     public D(String m) {
3         super(m);
4     }
5     public int number(int n) {
6         return 2*super.number(n+1);
7     }
8 }
```

```
1 public class E extends A {
2     public String message = "123";
3     public E(String m) {
4         super(m);
5     }
6 }
```

For each of the following parts indicate the output of the program. If applicable, report the *exact* compiler error message and explain what is the problem. Keep your answers short and precise.

(a)

```
1 A a1 = new A("Hello");
2 System.out.println(a1.message);
3 System.out.println(a1.number(5));
```

(b)

```
1 B b1 = new B("Bye");
2 System.out.println(b1.message);
3 System.out.println(b1.number(7));
```

(c)

```
1 A b2 = new B("Robot");
2 System.out.println(b2.message);
```

(d)

```
1 C c1 = new C("Java");
2 System.out.println(c1.message);
3 System.out.println(c1.number(3));
```

(e)

```
1 B b1 = new B("Robot");
2 A a1 = b1;
3 b1 = a1;
4 System.out.println(b1.message);
```

(f)

```
1 D d1 = new D("Bird");
2 System.out.println(d1.message);
3 System.out.println(d1.number(2));
```

(g)

```
1 E e1 = new E("Tree");
2 System.out.println(e1.message);
3 System.out.println(e1.number(2));
```

(h)

```
A e2 = new E("Cup");
System.out.println(e2.message);
System.out.println(e2.number(2));
```

(i)

```
D d1 = new D("Bird");
B b1 = d1;
System.out.println(b1.message);
System.out.println(b1.number(2));
```

(j)

```
Object o = new C("Java");
System.out.println(o.message);
System.out.println(o.number(3));
```

(k)

```
A c2 = new C("Car");
System.out.println(c2.message);
System.out.println(c2.number(4));
```

5 Project

This week we want to improve the hierarchy of our Game Objects with the help of interfaces and abstract classes. Also we want to find/create nice images to represents our Game Objects (sprites) and let them move around.

5.1 Improve Hierarchy

As a first thing you should take the hierarchy from last week and think again about the general structure, shared behaviours and values.
Now improve your hierarchy with the help of abstract classes and interfaces. You have to introduce at least two abstract classes and two interfaces. There is also the possibility to change a concrete class (e.g. Shot) into a abstract class.
When you are pleased with your updated hierarchy implement it as last week. So you do not have to implement the methods, just add the signature of it. The hierarchy should also be represented by the structure of your project.

5.2 Implement Move

As a next step we need our Game Objects to move. Implement for each Game Object for which it is useful a `move()` method with a proper signature.
Hint: Take a look at `exampleA` from week 1.

5.3 Create Sprites

To finish your hierarchy each Game Object needs a proper representation (sprite). Search in the internet or draw by yourself an sprite for each Game Object. If you have scaling issues you can use the method `getScaledInstance()`[2] when you create your representation object.

5.4 Tester

To check for proper operation and functionality create a `Tester.java` class and let your Game Objects moving around. If you have problems with this step check previous example and exercise solutions.

[2] https://docs.oracle.com/javase/8/docs/api/java/awt/Image.html#getScaledInstance-int-int-int-