Department of Informatics

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

Assignment 03

1 Theory

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

(a) Why is inheritance useful? Make a concrete example if necessary.

(b) Why is it possible to assign an object of type A to a variable of type parent of A, but not the other way around (assign an object of type A to a field of type children of type A) ?

(c) How many classes can a class extend from ? Why ?

(d) Why must `super` be the first operation when used in a constructor?

(e) What are the conditions to meet for a method that want to override a method existing in a super-class ?

(f) What is the keyword `final` doing when applied to attributes, methods respectively classes ? Make an example where this is beneficial for each of the cases.

(g) If a non-primitive attribute is declared as `final` can I make assumptions about its state (count on a given state in a given moment) ? Assume its reference its exposed by a getter method.

(h) Why is it useful to define a parameter of a method as final (in the signature of the method e.g. `public void foo(final int x)`) ?

(i) What is polymorphism ? When is it useful? *Hint: Bike.*

(j) Is a `private` attribute/method visible in a sub-class? If not, what is the maximum level or privacy for which it would be visible in sub-classes?

(k) Why is the usage of `instanceof` often considered as *not* good practice?

(l) What is the purpose of the method `clone()` ? Make an example where if that method is not implemented correctly there could be problems. *Hint: deep copy.*

2 Implementation

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

(a) Implement a class `Person` with protected fields for name and age. Implement the method `toString()` which print the object state as string (see https://docs.oracle.com/javase/6/docs/api/java/lang/Object.html#toString%28%29). Define constructor and other methods of the class as necessary.

(b) Implement a class `Employee` which extends from `Person` with protected fields for salary. Override the `toString()` methods to account for the new field. Define constructor and other methods of the class as necessary.

(c) Implement a class `Student` which extends from `Person` with protected fields for major. Override the `toString()` methods to account for the new field. Define constructor and other methods of the class as necessary.

(d) Implement a `Main` class with a `main()` method which create a `Person`, an `Employee` and `Student` objects and print their state with the implicit call of `toString` in a `System.out.print()` call.

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`, `Car.java` and `ElectricCar.java` attached to this series for this exercise.

(a) Report both *exact* compiler error messages and explain what is the problem if you were to run `Main()`

(b) Modify the code of `Main.java` to fix the problem.

(c) Now that the code runs without error messages, explain why the field `enginePower` is not being printed when printing the object "electricCar". What is the mistake?

(d) Modify the code to fix the problem by overriding the `toString()` method in `ElectricCar`. To do so, use the auto-generation tool provided by you IDE (for NetBeans see https://platform.netbeans.org/tutorials/nbm-code-generator.html and for IntelliJ https://www.jetbrains.com/help/idea/generating-tostring.html). Solutions featuring manual implementation will not be accepted. Make use to include all class and super-class fields.

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

```java
package core;

public class Main {
    public static void main(String[] args) {
        Car car = new Car("MyCar");
        System.out.println(car);

        ElectricCar electricCar = new Car("MyElectricCar", 100);
        System.out.println(electricCar);
    }
}
```

```java
package core;

public class Car {

    protected String model;

    public Car(String model){
        this.model = model;
    }

    @Override
    public String toString() {
        return "Car{" +
                "model='" + model + '\'' +
                '}';
    }
}
```

```java
package core;

public class ElectricCar extends Car {

    private int enginePower;

    public ElectricCar(final String model, final int enginePower){
        super(model);
        this.enginePower = enginePower;
    }

}
```

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 to answer the following theoretical questions. Keep your answers short and precise.

(a) Explain in your own words what is the concept of "autoboxing" in Java (see https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html). Is this just syntactic sugar or it shadows some potential issues when used carelessly ? Motivate your answer.

(b) Explain how can one call alternate constructors in a class i.e. when there are multiple constructors for a single class and you want to call another one.

(c) Can a class be declared as `static` ? If yes, under which circumstances?

(d) What is the major difference between static nested classes and non-static nested classes?

(e) What is the issue that can arise when having a getter for an attribute of a non-primitive type i.e. an custom object?

(f) Why can `Integer` be `null` but `int` can't?

5 Project

For this step of the project, you will have to improve/adapt the class `Sprite` which you wrote last week to take inheritance into account and plan the hierarchy of your game objects.
Space Invaders[1] as a game consists of different game objects like enemies, shots or the player. This game objects share different properties and behaviours e.g. all of this game objects have to be able to be drawn onto a canvas.
To achieve this we will use inheritance.

5.1 Design & Implementation

Draw and describe your game object hierarchy which should contain at least these three different objects (Enemy, Shot, Player). Also think about the behavior and the properties each of the game objects must have (e.g. color or possibility of being hit). You can reuse and modify the class `Sprite` from the previous series. This hierarchy can still be changed in later versions of the project.
When you are pleased with your hierarchy implement it by creating the classes and adding constructors and fields. Methods can be empty as you will implement them later on. Then, answer the following questions:

- Are some fields/methods equivalent, even if they have slightly different names in your implementation of `Enemy` and `Player`?
- Which ones of the fields and methods could be implemented in the `Sprite` class?
- Do some methods have to be present in the `Sprite` class but cannot be implemented in this class?

5.2 Additional Lists

To store all of these new game objects we need lists. Implement for each game object for which it makes sense an own list (it is not allowed to use any implementation of `List` to solve this exercise). You can copy the `SpriteList` from the previous series.
Additionally answer these questions:

- Based on your hierarchy is it possible to store Enemy or/and Shot objects into your `SpriteList`? Why?
- What would be the problem, when you would save Enemy and Shot objects into your `SpriteList`?

[1] https://en.wikipedia.org/wiki/Space_Invaders