Software Development, 2019-2020

Tim Stas, Kymeng Tang, Arnor Van Leemputten, Gil Vranken, Jeroen Wauters, Koen Pelsmaekers.

Lab session 1-2: Introduction to inheritance

Goal

The goal of this lab sessions is to provide an introduction to inheritance in object-oriented programming, by means of two small exercises.

Exercise 1

Start by designing an UML class diagram (Visual Paradigm), containing all the necessary classes, methods and relationships. Mark all methods that override another method. Once your lab teacher has approved your design, implement it in Java (IntelliJ IDEA).

You are hired by a company to develop a flexible software system for automatic security. To this end, different types of sensors are installed in a building, all of which are connected to a centralized control center. Each sensor contains information about its type and location (f.i. “smoke sensor” in the “kitchen”), the name of the manufacturer and whether or not it is active (when creating a sensor, it is inactive by default). Some possible sensor types:

- Smoke sensor
- Motion sensor
- CO sensor

It should be simple to add extra types of sensors (to your code).

Every type of sensor executes a specific alarm procedure when it is activated. A CO sensor will for instance open the windows; a smoke sensor will close them to prevent a potential fire from spreading; a motion sensor will automatically contact the police.

For this exercise you can simulate these actions with a simple print operation, which shows the information of the activated sensor along with the action being executed. For example:

```
Alarm in smoke sensor kitchen (sensorCompany)
Windows are being closed and siren is sounding
```

When adding a sensor to the control center, a check is performed to make sure that the sensor is not already present in the system. Sensors are considered identical if they have the same type, the same manufacturer and the same location.

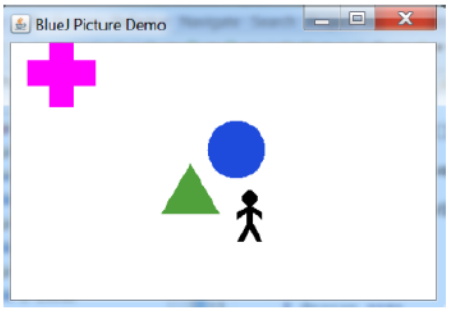
Each sensor type provides some specific configuration options. Motion sensors can be configured with the distance at which their alarm will trigger; smoke sensors can detect both smoke and heat or only smoke; CO sensors have a configurable minimal concentration of CO which triggers the alarm.

From the control center one should be able to perform a test for either one specific sensor or all sensors at the same time. When a test procedure is called, active sensors respond in the same way as when an actual alarm would be triggered (inactive sensors do not respond at all). When you call the method `testSensor(Sensor s)` it will return the index at which the sensor was found in the collection (or -1 if the sensor was not found). `testAllSensors()` returns the amount of active sensors.

Every control center has a name (the building which it monitors) and one of the following categories: small, medium or large. Use an enum to implement this. The center should be able to provide an overview of all the sensors and their status. Override the method `toString()` in all sensor classes to achieve this.

```
Overview of all sensors @ Campus Groep T (medium)
Info of active sensor (type = smoke sensor), from sensorCompany located at kitchen
smokeOnly = true
Info of inactive sensor (type = motion sensor), from sensorCompany located at garage
with an active radius of 0.3m
Info of active sensor (type = motion sensor), from sensorCompany located at garden
with an active radius of 5.4m
```

Exercise 2



Create a new project in IntelliJ IDEA and import the classes from shapes.zip on Toledo (you do this by copying the files to the src folder in your project). Your assignment is to refactor this code: use inheritance to remove as much duplicate code as possible from the project. Start by examining the code for all the shape classes an create a super class `MyShape`, which contains all of their shared functionality and properties. Note the use of `private/protected/public`.

Subsequently add a class `SimpleDrawingProgram`, containing a polymorphic collection of `MyShape` objects. Implement the functionality to add a new `MyShape` object and to draw all of the shapes in the collection. You can test this class by attempting to draw your car from the first lab of the course of Object-oriented Programming and Databases. When this works correctly, add a new shape: a rectangle (hint: check the class `Square`). Test the functionality of this new class by adding a few rectangles of different colors to your drawing.