Programming Techniques, 2022-2023

Zhou Nianmei, Ludo Buynseels, Stijn Langendries, Kymeng Tang, Jeroen Wauters, Koen Pelsmaekers.

Lab session 1-2: Introduction to inheritance

Goal

The goal of these lab sessions is to introduce inheritance in object-oriented programming, by means of two small exercises.

Exercise 1

Start by designing an UML class diagram (Visual Paradigm), holding 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, diverse types of sensors are installed in a building, all of which are connected to a centralized control center. Each sensor holds information about its type and location (for instance "smoke sensor" in the "kitchen"), the name of the manufacturer and whether 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.

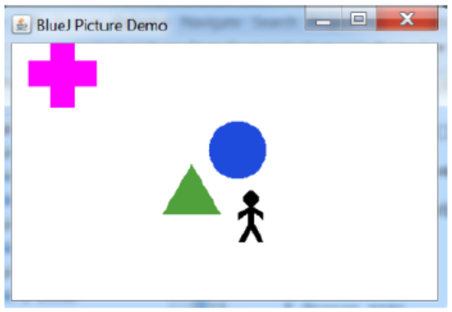
Each sensor type provides some specific configuration options. Motion sensors can be configured with the distance at which their alarm will trigger; smoke sensors can detect both smoke and heat or only smoke; CO sensors have a configurable minimal concentration of CO which triggers the alarm.

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.

Exercise 3

At one point in the course Object-Oriented Software Development (OOS), you created a weather station with two measuring devices: a thermometer and a barometer. The assignment for that exercise is below.

> *A weather station contains 2 measuring devices: a thermometer and a barometer. Temperature is recorded in °C and atmospheric pressure in hectoPascal. The thermometer and barometer not only record the present value, but also remember the maximum and minimum of the recorded values. Define the class(es) and object diagram of your solution.*
>
> *When you create a measurement device (thermometer/barometer), all values are initialized to meaningful values. Provide the weather station with a method to record new measurements (one method that records temperature and pressure at the same time). In reality this value is given by the sensor, we will simulate this by providing the values. It is also possible to "reset" the weather station. At that moment all extremes are forgotten, so after providing a first measurement, both values, min and max should be the same.*
>
> *Also provide a method to get an overview of the extremes.*
>
> *This example on object communication is a situation where the "system" (i.e. the weather station) is responsible for creating all the parts. So the constructor of the "system" creates instances of the individual elements.*

When you made this exercise, you did not know about inheritance yet, so the typical OOS-student takes one of two approaches: easy logic too much

- Three classes: `WeatherStation`, `Thermometer` and `Barometer`, with the weather station containing an attribute of each of the other two classes. reuse things
- Two classes: `WeatherStation` and `MeasuringInstrument`, with the weather station having two attributes of type `MeasuringInstrument` (one representing the thermometer, and one representing the barometer. think more

Which one did you implement? Why? Think about at least one positive and one negative aspect for each of these solutions.

a. A better solution

As you now (hopefully) realize, there is a third approach that combines the upsides of each solution with none of the downsides: the use of inheritance. Reimplement a solution for the weather station exercise, but this time create a parent class `MeasuringInstrument` with two children, `Thermometer` and `Barometer`. These two different instruments both contain all functionality described above, but now we also want to add some specific functionality for each:

- A thermometer contains an attribute (type char) indicating whether it measures temperature in Fahrenheit ('F') or Celsius ('C'). If any other value is passed to the constructor, 'C' is taken as the default value. The method to display the current value should also display "°C" or "°F".
- A barometer has a method `divergenceFromNorm`, which returns how far the current value diverges from the average atmospheric pressure of 1013 millibar (negative value if lower, positive value if higher).

b. Updating the weather station
Next, we will update the existing weather station.

- Make sure the weather station can contain any number of measuring instruments. At construction time, the weather station doesn't contain any instruments yet, so you should provide a method to add new instruments to the station.
- Each of the station's methods (new measurement, reset and print) should still function. This means that the new measurement method requires a random number of parameters. You can achieve this with the triple dot notation: `float… values`. The actual type of values is `float[]`, so you can iterate over it and access elements like any other array.
- To test your approach, add an extra thermometer to your weather station, so it now contains three measurement instruments in total.
- Finally, add a new measurement device: a pluviometer (= rain meter). This should be able to perform all operations that a barometer and thermometer have, and also store the total amount of rain measured over its entire lifetime. Override the method `newMeasurement` so it updates this value accordingly, and call the parent's implementation using the super keyword, rather than copy-pasting. Your overridden version of `newMeasurement` should only be two lines long!

c. Unit testing
Just like in BlueJ, you can run unit tests in IntelliJ as well. Download the file WeatherStationTest.java from Toledo and add it to your project.

You will most likely have used different method names than those used in the test class. Start by refactoring your code to correspond to the naming conventions dictated by the test class, so your project compiles.

Once it does, make sure every test succeeds. Note that the assignment is vague or incomplete in places, and you will have to deduce some nuances of the weather station's behavior from the test code. This is intentional. Carefully check the test methods, and don't forget to read the comments.

Exercise 4

Let's take the previous idea one step further. Your assignment is to write code for a store management system, but this time you don't get any explanation. All you have to work from is the test file StoreManagementTest.java on Toledo. Create a new project and import the test file. Then, one test at a time, analyze the test code, read the comments it contains, and figure out which classes and methods you need and how they should function.