IS1220 - Object Oriented Software Design

Tutorial 06

Paolo Ballarini

General instructions:

- If you have not done so already, create a workspace for your tutorial `IS1220/TDs`.
- Within the workspace, create a new package for this TD called `fr.ecp.IS1220.TD06/`.
- Carefully document your code (JavaDoc), i.e., explain the general idea of your algorithm, explain the relevant steps in the code implementing the algorithm, document assumptions (if any), corner cases, and error conditions.

Learning outcomes:

- Application of design patterns
- Factory pattern
- Strategy pattern

Exercise 1. Extendable `Shape` classes

Design a class library for representing geometrical shapes. Initially your library should provide classes for representing circles, quadrilateral and triangles objects (choose a proper level of granularity for representing each of the above type of shapes, for example represents a Circle as a the coordinates of its centre and the length of its radium, whereas to represent a polygon you can give the coordinates of its vertices). **Requirement**: the shape Library should be open to further extensions (it should be possible to easily add new shape classes later on).

Q1) Develop a UML class diagram which represents the shape library design.

Q2) Think about the patterns you are going to apply and whether it is useful to add a Factory

Q3) To test you library some Unit Tests. For example one of your test must define 10 shapes objects (at least one of each type, i.e. Triangle, Rectangle, Circle). Write the Unit Tests incrementally: each time you add a new class or functionality.

Exercise 2. Adding different sorting strategy to `Shape` class

Extend the shape library project (developed in Exercise 1) so to equip the `Shape` library with *interchangeable* sorting functionality for shape objects. In particular a client application that uses a collection of shape objects should be capable of sorting them according to the following criteria: *longest perimeter*, *larger area longest side shortest side*. The customer that commissioned you with the realisation of the `Shape` library requires that itshould also be extensible with respect to the sorting criteria, so that in future one can easily add new criteria for sorting the shape objects and the client application can easily swap from one sorting algorithm to another (i.e. what kind of design pattern would fit this kind of problem?).

Q1) extend the UML class diagram which represents the shape library design according to the new requirements.

Q2) modify the `Shape` class so that it realises a Strategy pattern which provides the (*longest perimeter*, *larger area longest side shortest side* sorting functionality (i.e. with respect to the *longest/shortest side* you should consider the "side" of a circle as its diameter, whereas for the *longest perimeter* you should consider the circumference of a circle).

Q3) Test you library and extend the `ShapeClient` application such that it displays also the result of sorting them out with all of the different criteria, hence display the list of shapes ordered with respect to the larger area, the longest perimeter, the longest side, and the shortest side.