Exercise 7:
Design Patterns

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Software Engineering Design & Construction
WS 2016/17 - Dr. Michael Eichberg, M.Sc. Matthias Eichholz

Although this exercise is not graded, it is highly recommended to also do them on your own. Just looking at a solution is much easier in comparison to actually coming up with it. Support can be found in the office hours of our tutors and in the forum at `https://www.fachschaft.informatik.tu-darmstadt.de/forum/viewforum.php?f=234`.

Task 1 Scala 2.7 Collections

In the project for the first exercise, you will find the source code for the Scala 2.7 standard library, which contains a collection hierarchy. Note: do not expect to be able to compile it with a current Scala compiler or in Eclipse / IntelliJ. We know that Scala 2.7 is an old version, but newer versions are too complex, please do not try to analyze other Scala versions than 2.7. Analyze how the core of the collections hierarchy makes use of the following design patterns:

- Factory (any variant: abstract factory, factory method)
- Observer
- Strategy
- Template

Look at all of the following classes and traits and their corresponding companion objects:

- Everything in packages `scala`, `scala.collection` and `scala.collection.mutable` that directly or transitively inherits from `scala.Iterable`.
- All base traits and base classes of the above traits and classes *only if they participate in a pattern*.

For each pattern name the participants, which roles they have, how they collaborate and how they might vary from the standard text book examples. Explain the pattern in terms specific to the current pattern instance, e.g., don't just copy text from a design patterns book. Your analysis should be complete and you should mention how often a specific pattern is used in comparison to others. If a pattern is used in a similar way in many different places, give a detailed analysis of a single example and say how other instances of the pattern in the collection hierarchy relate to it.

Task 2 Synchronized Maps

Now have a look at the `SynchronizedMap` trait in the package `scala.collection.mutable`. Which design pattern is used to add synchronization to maps? Does the design pattern that is used resolve the fragile base class problem? Justify your answer. For this task you can use the current version of the Scala standard library.

Task 3 Order Management System

The word has spread about your successful implementation of the order management system (OMS) for Starbuzz Inc. You received an email from the manager of TastyPizza Inc. who heard about your success. He wants you to implement a new OMS for his pizza delivery shop. For the first milestone, he has defined several requirements that need to be fulfilled by your OMS:

- The shop offers pizza, several beverages (lemonade, water and wine) as well as some franchise items (shirts, mugs). The system needs to be able to represent these products.
- All products have a price and pizza and beverages have specific nutrition facts.

- Every pizza contains at least cheese and tomato sauce (Pizza Margherita). It can be combined with additional toppings (ham, onions, etc.), which can be ordered multiple times.
- Pizzas with certain topping combinations are named, e.g., Hawaiian Pizza for the base pizza with ham and pineapple. Those names should be contained in the bill. You do not need to auto-detect names for such combinations if they are not explicitly requested in the order.
- In addition to the normal size, pizza can be ordered in family size.

The manager sent you the menu card of TastyPizza for a better understanding of their available items:

Pizzas	**Calories**	**Price**
Pizza Margherita (tomato, cheese)	1104	4.99
Hawaiian Pizza (tomato, cheese, ham, pineapple)	1024	6.49
Salami Pizza (tomato, cheese, salami)	1160	5.99
Family Size for Pizza	x 1.95	+ 4.15

Toppings	**Calories**	**Price**
Cheese	92	0.69
Ham	35	0.99
Onions	22	0.69
Pineapple	24	0.79
Salami	86	0.99

Drinks	**Calories**	**Price**
Lemonade (0.33l)	128	1.29
Water (0.5l)	0	1.29
Wine (0.75l, 13%)	607	7.49

Franchise	**Price**
TastyPizza Shirt	21.99
TastyPizza Mug	4.99

Your tasks for this exercise are as follows:

- Implement an OMS library in Scala for TastyPizza, Inc that provides the requested features. You do not need to implement a frontend, since your library should be integrated in an existing cashing software. Provide tests to document the usage of your library.
- Your library design should use immutable objects, i.e., orders and pizzas are never changed in-place. For example, adding a topping to a pizza creates a new pizza object.
- Use Scala objects instead of classes where possible.
- Use the Decorator Pattern to implement the different variations of pizza (additional toppings, family sized).
- Create a UML class diagram that illustrates your design.