Modeling and Programming 2018-2: First Project

Luis Daniel Aragon Bermudez 416041271

May 15th, 2018

Contents

Simple Sandwich Store Mock-Up/Library/Demo that uses the Template, Observer and Decorator patterns

Introduction

This is a Kotlin Mock-Up/Library/Demo that models (somewhat) how a sandwich store would work under certain circumstances.

Background

This demo is based on the following specification:

> **English [en]:** Design the UML diagram and implement a solution for the following problem: We have a tortería that has several locations, each branch keeps an inventory of their ingredients in a different data structure, each branch can know what is missing from their respective inventories. These ingredients are white roll (bolillo), whole roll (bolillo), lettuce, ham, chicken, head cheese, beef milanesa, tomato, white cheese, manchego cheese, mayonnaise, mustard and ketchup. The supervisor of all branches must be able to obtain this information to coordinate the items that must sent to the stores. There is a menu of 10 "tortas" with predefined ingredients, however clients can optionally add ingredients for an extra cost to their "tortas" and can also make their own custom "torta". The "tortas" that are sold are logged in the system as a special product called "simple torta" and "wholegrain torta", depending on what they contain.
>
> Also, the torterias' boss has been so successful that he has bought a sandwich place and wants to be able to manage the branch in the same way. In this sandwichería the ingredients of white bread, wholegrain bread, lettuce, ham, chicken, head cheese, sausage, tomato, white cheese, mayonnaise and ketchup are used. The bread for simple sandwiches has a different cost from that of bolillos', but the boss wants

to avoid modifying the database, so he asks that the sold sandwiches be registered as “simple torta” and “wholegrain torta”, adding an extra ingredient with a negative price to the register called “bread discount”.

The boss needs to have a record of his daily sales with the name of the item sold, each branch’s balance, and in general he/she wants to be able to know how much each branch made.

Español [es]: Diseña el diagrama de clases e implementa una solución para el siguiente problema: Tenemos una tortería que tiene varias sucursales, cada sucursal guarda un inventario de los ingredientes de las tortas en una estructura de datos diferente, cada sucursal puede saber qué le falta de sus respectivos inventarios. Estos ingredientes son bolillo blanco, bolillo integral, lechuga, jamón, pollo, queso de puerco, milanesa de res, jitomate, queso blanco, queso manchego, mayonesa, mostaza y catsup. El jefe de todas las sucursales debe poder obtener esta información para coordinar los artículos que debe enviarles. Existe un menú de 10 tortas con ingredientes predefinidos, sin embargo cuentan con la posibilidad de agregar ingredientes por un costo extra a sus tortas y también pueden hacer su propia torta, las tortas creadas se guarda en el sistema como un producto especial llamado “torta simple” y “torta integral”, dependiendo de lo que contengan.

Además nuestro jefe ha tenido tanto éxito que ha comprado una sandwichería y desea poder manejar la sucursal de la misma manera. En esta sandwichería se utilizan los ingredientes de pan blanco, pan integral, lechuga, jamón, pollo, queso de puerco, salchicha, jitomate, queso blanco, mayonesa y catsup. El pan para sandwich tiene un costo distinto del de un bolillo para torta, pero el jefe desea que la base de datos no se modifique demasiado, por o que pide que los sandwiches vendidos se registren como “torta simple” y “torta integral” y se agregue un ingrediente extra en el registro con precio negativo llamado “descuento de pan”.

El jefe necesita tener un registro de sus ventas diarias con el nombre del objeto vendido, el precio por sucursal y además le gustaría poder ver cuánto ganó en cada sucursal.

It was developed as part of the activities of the 2018 Modeling and Programming course taught by Prof. Rosa Victoria Villa Padilla at the Science Faculty of the National Autonomous University of Mexico.

Demos

The demos are set up with four different Sandwich stores, three of which are torterías:

- TorteriaEightAv.
- TorteriaMainSt.
- TorteriaTeslaBlvd.
- SandwicheriaLimeDrv.

all of which are managed by a single supervisor. Each store also has one clerk that can make and sell sandwiches.

Demo 1

Makes each clerk prepare a random menu item from their store. The supervisor replenishes all the stores’ inventories afterwards.

Demo 2

It makes the clerks from Main St and Lime Drv prepare the same sandwich. Since the clerk at Mains St works at a tortería he shouldn’t add a discount to the sandwich, but the clerk from Lime Drv should. The supervisor replenishes all the stores’ inventories afterwards.

Design

The following design patterns were used:

- Decorator pattern. It is the base of our `Sandwich` implementation. It allows extremely versatile object creation and modification at runtime.
- Template method pattern. It is used for `SandwichStoreClerk` creation and sandwich creation/selling.
- Builder pattern. `SandwichStore` objects are `SandwichStoreClerk` builders.
- Observer pattern. Relates clerks, stores and supervisors using updates and transactions.

UML diagrams can be seen bellow, furthermore all the classes are thoroughly documented. Reading the docs is highly recommended!

Building and running the program

The program can be built using gradle, the most common tasks are described bellow, for a full list of available tasks use `./gradlew tasks`. If you're on Linux or Mac then running the following command from the project's main directory will be enough to build and run the program: `./gradlew run`. If you're on windows use `gradlew.bat run` from the command prompt instead.

Some of the most common tasks are:

1. `./gradlew build`, compiles and creates the outputs of this project.
2. `./gradlew dokka`, generates the program's documentation and puts it inside `docs`.
3. `./gradlew run -Pdemo=[1-2]`, builds the program and runs the specified demo.
4. `./gradlew clean`, deletes all files and folders generated during the build process (except the .gradle directory).

Further work

Check-out my TornadoFX adaptation of this simple app.

Acknowledgements

For more information on the tools used to build, create and run this program refer to the following links:

- Pandoc is a Haskell library for converting from one markup format to another, and a command-line tool that uses this library. Pandoc was used to keep this README file consistent.
- Gradle is the build tool used for this project.
- Dokka was used to provide beautiful documentation.
- JetBrains' IntelliJ IDEA was used as the primary editor. Its diagramming utility also came in handy to produce the application's class diagram.

Figures

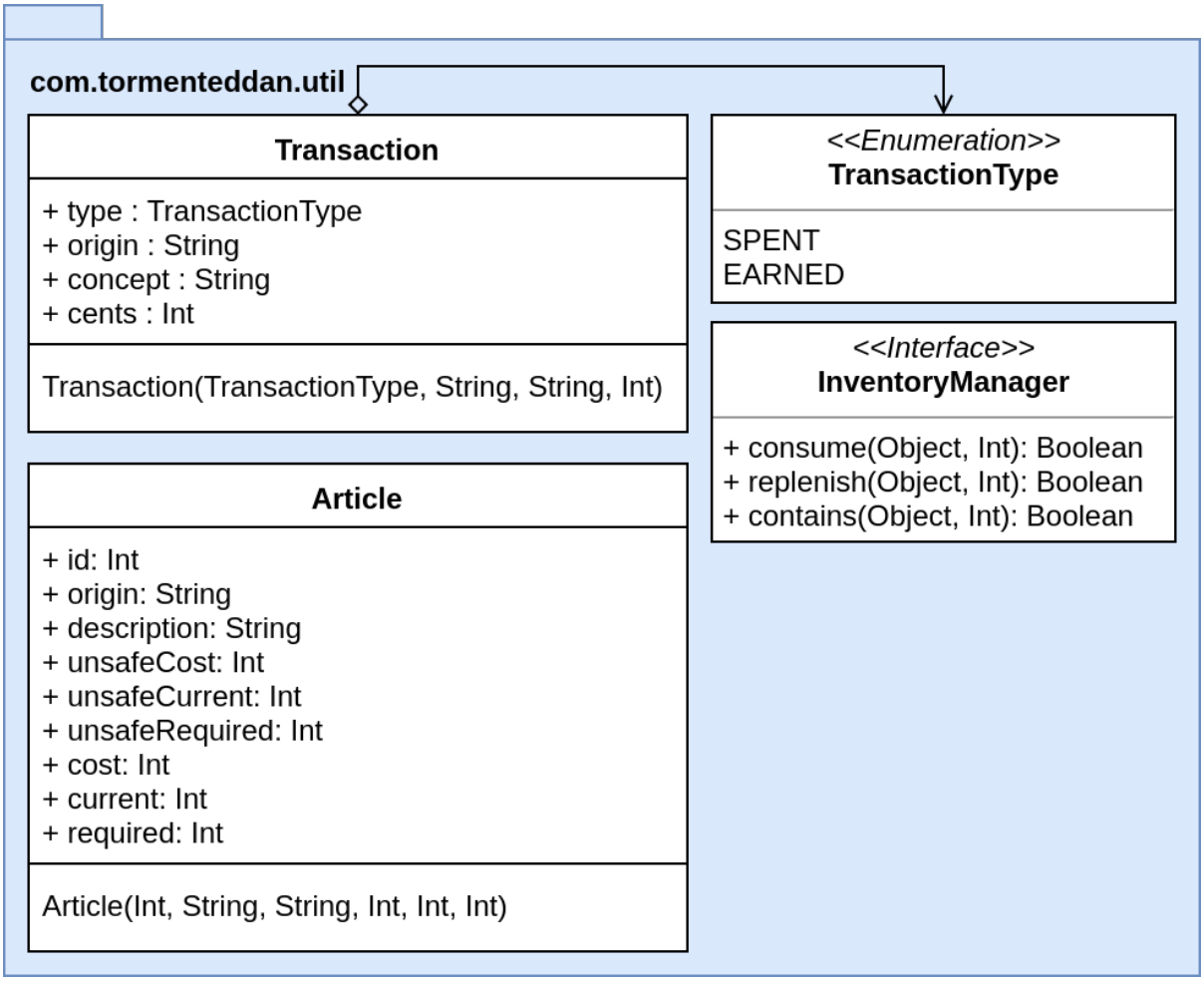

Figure 1: Util module UML class diagram. Library with useful interfaces for storing and working with inventories.

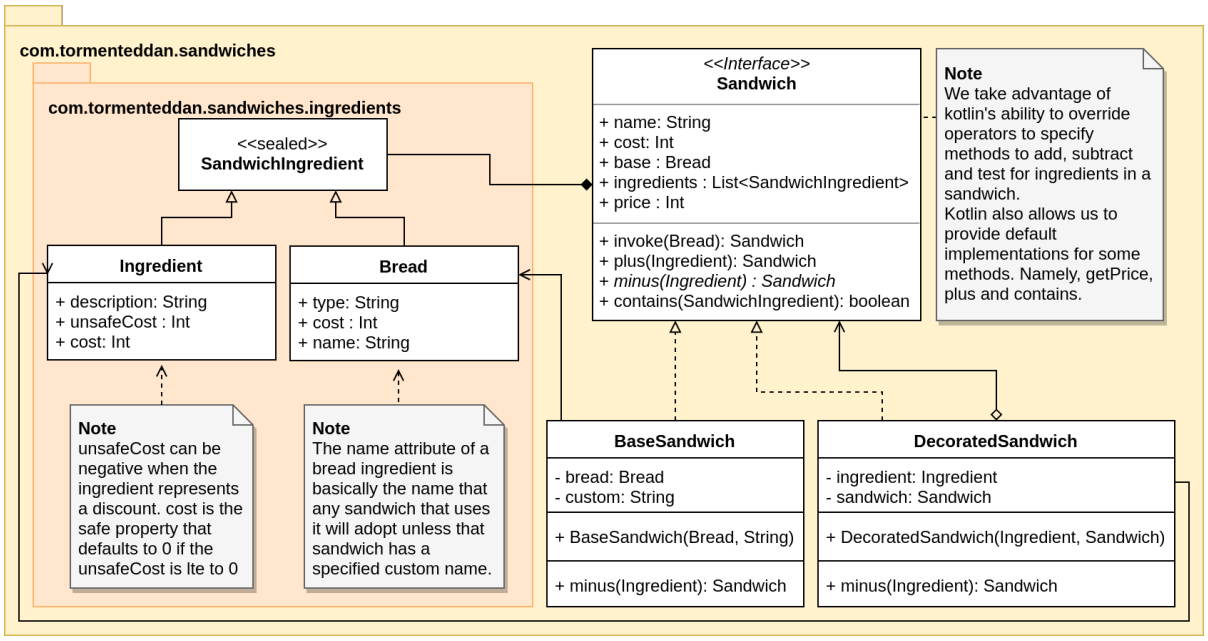

Figure 2: Sandwiches module UML class diagram. Library with a Sandwich implementation that uses the decorator pattern.

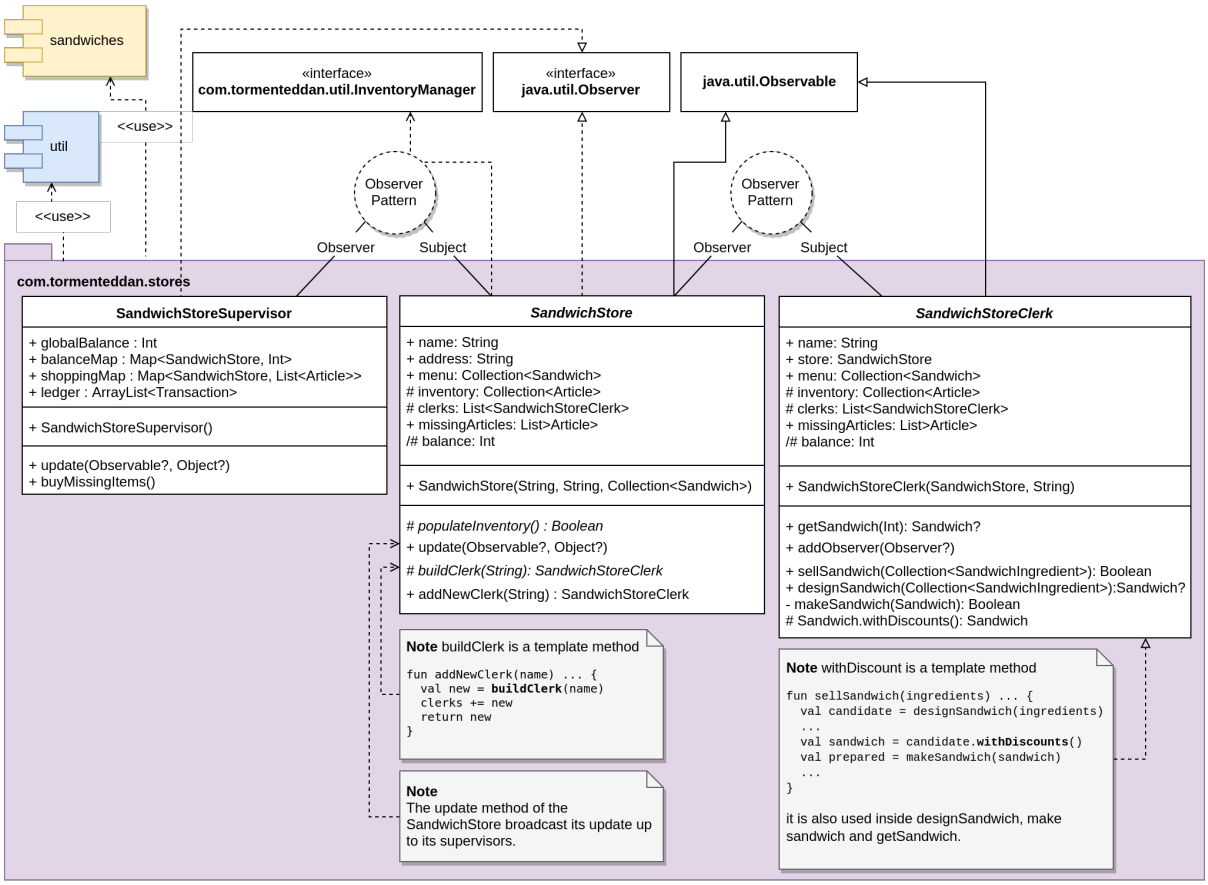

Figure 3: Store module UML class diagram. This package contains the `SandwichStore` class, which is both observable and an observer. In practice a sandwich store would observe its clerks and notify its supervisors. Both `SandwichStoreClerk` and `SandwichStoreSupervisor` classes are included in this package as well.