Contents

Case Study

The main objective of this course project is to design and implement a simple ordering system utilizing some of the programming concepts and paradigms in F# and Python programming. This is an example case study where we consider VIA Canteen that sells some products such as food, fruits, and drinks. For now, orders are taken in-house by order entry personnel. However, some of its customers come from outside (like SOSU or other buyers using the train station). On a closer look, VIA canteen realizes that it cannot be effectively run with its current setup and therefore needs a new online order management system (with a new name Goodies To Go - G2G).

Currently, G2G offers drinks, food, and fruits. Further, it is considering adding to its product line in-demand take-away goodies, room delivery on the campus and other products that are not just food, fruit, and drinks. G2G is very ambitious; it hopes to open a more order-processing location on other VIA Campuses within the next one year.

Our goal is to design and implement a system that not only meets G2G's immediate needs but also is flexible enough to support other types of products in the future. The solution will allow customers (for instance, VIACustomer, SOSUCustomer) to order goodies through their computer (or mobile phone). Below is a simple conceptual domain model showing some of the objects.

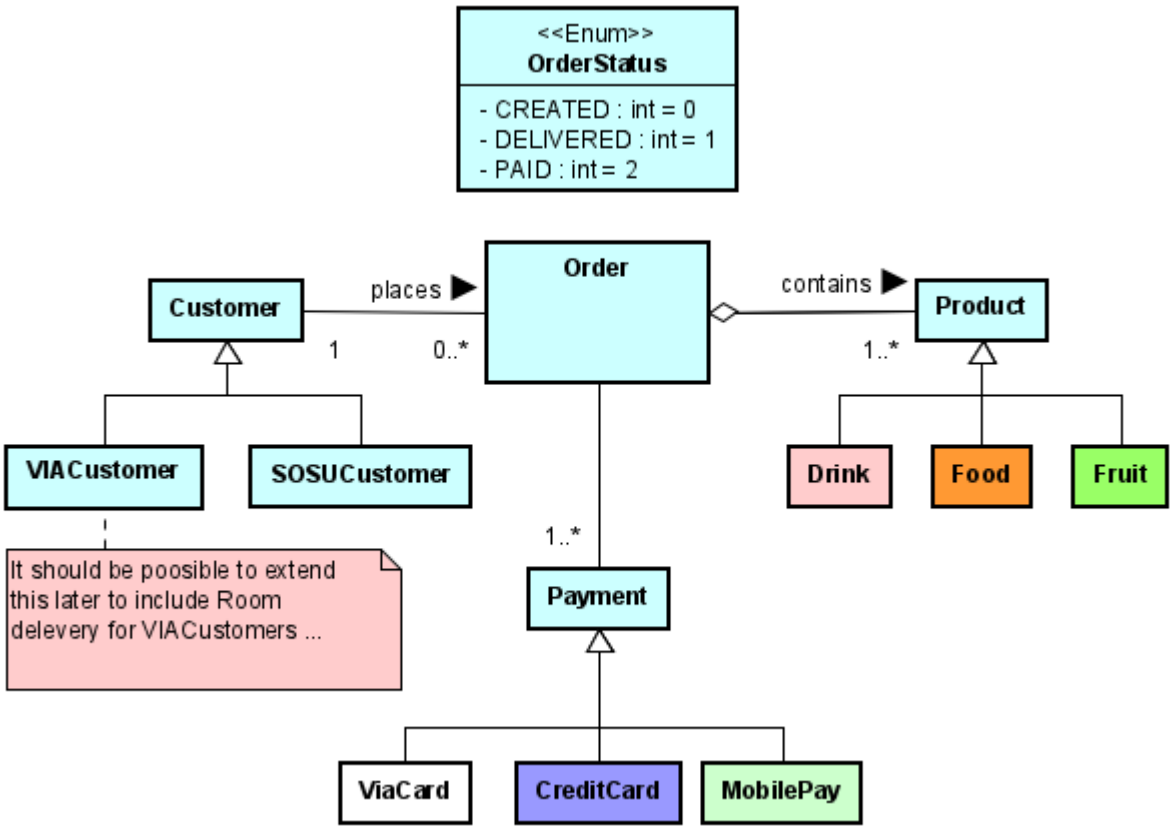

Figure 1: Conceptual domain model.

To develop this application, a simple domain-specific language for describing the objects, products and all the different categories of drinks, food, and fruits as well as computing the prices need to be developed.

In this "first sprint" of the project, you will define data types to model the domain of the application as shown in the conceptual domain in model above. The task is to define the different types of drinks, food, and fruits in the café (canteen) and implement the price computing function. You need to decide for yourself the prices for the different drinks. For instance, the **drink**s can be different types of coffee, tea, juice, soda, milk, together with their corresponding **size**s (Small, Medium, and Large) and should have different **price**s.

The types should have at least:

- 3 different types of coffee drink
- 3 different types of tea drink
- 3 different types of juice drink

You will be computing the different prices for the above drinks depending on the type and size of the drink. The same will be repeated for the food and drinks.

Sprint 1 - F# Domain Driven Development

You are allowed to develop your own strategy for describing the domain objects and the different drinks together with their sizes as well as functions that compute the prices for a given type of drink, food, or fruit.

As a minimum, you should define a simple data type for describing the different drinks, food, and fruits and then implement a function to compute the price.

Hints: You will need to define the data type for drinks (using **union**) and the three chosen categories, for instance, coffee or tea or juice together with the size (using **record**). You will also need to define the price computation function that uses **pattern matching**.

Sprint 2 - Actor Model/MailboxProcessor

In this second "sprint", you are required to:

1. define a simple data type for describing the customer and payment types as well as the order.
2. declare a function ***gtgVAT: int -> float*** such that the **value** ***gtgVAT n x*** is obtained by increasing **x** by **n** percent. Use any percent value for your VAT.
3. implement a function to order a product (drink) item or leave a comment, in a concurrent way using ***gtgVAT*** function to charge VAT when the drink is of type coffee.

Hints: The ***OrderDrinkMsg*** (or ***OrderProductMsg***) message processing should use the ***Price*** calculation function defined in Sprint 1 (which returns the price for the specified product (eg. drink) multiplied by the given quantity.

F# has both shared-memory concurrency and message-passing concurrency. As discussed during the lessons, F# has a built-in ***mailbox processor*** concept that is popular in **Erlang** language. This built-in mailbox processor is defined in the F# library as a type called **MailboxProcessor** and usually referred to as an **Agent** or **Actor**.

For instance, to order a drink:

```
type OrderDrinkMsg = | OrderDrink of Drink * int // item and qty
                     | LeaveComment of string
```

Or a message to order Products (if you modelled all the domain objects):

```
type OrderProductMsg = | OrderDrink of Drink * qty:int // item and qty
                       | OrderFood of Food * qty:int
                       | OrderFruit of Fruit * qty:int
                       | LeaveComment of string
```

Implement an agent (call it ***gtgAgent*** or ***gtgActor*** if you prefer) that receives an OrderDrinkMsg declared above and prints a message with the price of the drink to the sender (for OrderDrink or OrderProduct). Further, a second message LeaveAComment of

string (ie. “Comment”) to leave a string comment. Acknowledge the comments with your own ideas. For instance, depending on your implementation:

```
> gtgAgent.Post(OrderDrink(Coffee{type=Latte; Size=Small}, 2));;
```

Should give for instance: `> Please pay DKK34.00 for your 2 Latte coffee drinks. Thanks!`