Software Systems Analysis & Design
Fall 2021
Assignment 3
Team 32

Vladimir Makharev, Artem Batalov, Kirill Kuznetsov
Murat Shaikhutdinov, Pavel Baharuev
B20-01, B20-01, B20-04, B20-05, B20-01
Innopolis University

November 28, 2021

Contents

1 Report — Supermarket Billing Software

1.1 Requirements

Description

Most of the supermarkets use software for preparing the final bill before the goods are packed. This software itself can very easily be designed by someone starting in Java (Note that most of the actual software used for billing in supermarkets use Java itself). This billing software sums the amount for individual items and then adds them up to get the final sum that the customer has to pay. The number of individual items, the price of each item should be editable for the user. Also, there should be an option to remove the item from the list altogether.

Design patterns

- Chain of Responsibility
- Command
- **Iterator – chosen**
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

Technical requirements

Programming language: *Java*, as usual :)

1.2 Solution Description

Source code: GitHub Repository (follows the Google Java Style Convention)

Project Context

We need to implement the supermarket billing software. A program should provide the interface to work with the so-called cash register. There are possibilities to add, remove, edit, and manage the shopping items in the customer's cart. Finally, the program should make a bill for the customer.

Why do we use the *Iterator* design pattern?

We have chosen the Iterator as its purpose is to allow the user to process the container's elements while isolating them from the underlying structure of the container. In our case, it means that the user does not care about the shopping cart structure while working (deleting, managing, *etc.*) with the cart's product positions. The Iterator also helps us to be prepared for further extensions of the program in the context of the shopping cart and product classes.

The job we have done

First, in the UML class diagram, we have added the new Iterator interface and its concrete implementations. New classes have an aggregation relationship with the shopping cart class. Second, in the source code we have provided the `DeleteIterator` interface and its implementation – `ProductPositionDeleteIterator` class. The latter one uses the `ShoppingCart` as an iterable object and has special function `getNextAndDelete()` that allows to delete `ProductPosition`'s objects while iterating over shopping list.
The pattern in the code can be found in the `iterators` package and in the `ShoppingCart` class.

Project structure

Our code imitates the work of a supermarket cash register billing. We have classes for the customers, product cart, cash register, products, and decorators. For each customer, we have one shopping cart. In this cart, we can add and remove products. We use the cart by operating it within the customer (only the customer sends the request to the shopping cart). When the order is ready, we push it to the cash register. The cashier can edit the parameters of the product list (i.e., customer's shopping cart) using a key. We check the key in the product class. Customers can interact with products only by cloning them to the cart (Prototype design pattern from assignment 1). If we need to apply the evening discount for the group of products or decrease the price for rotten products, we use the decorators (Decorator design pattern from assignment 2). When we iterate through the list of the products to see the unacceptable (e.g., we have added the product that is already out of stock) ones and then remove them, we use the iterator pattern (assignment 3).

Application of the Iterator pattern

Our implementation of the Iterator design pattern can work with the lists and derived classes. It provides the opportunity to delete an object without the termination of the iteration. Also, it solves the ConcurrentModificationException problem much better than crosschecking everything in every step of the iteration.

1.3 UML Class Diagram

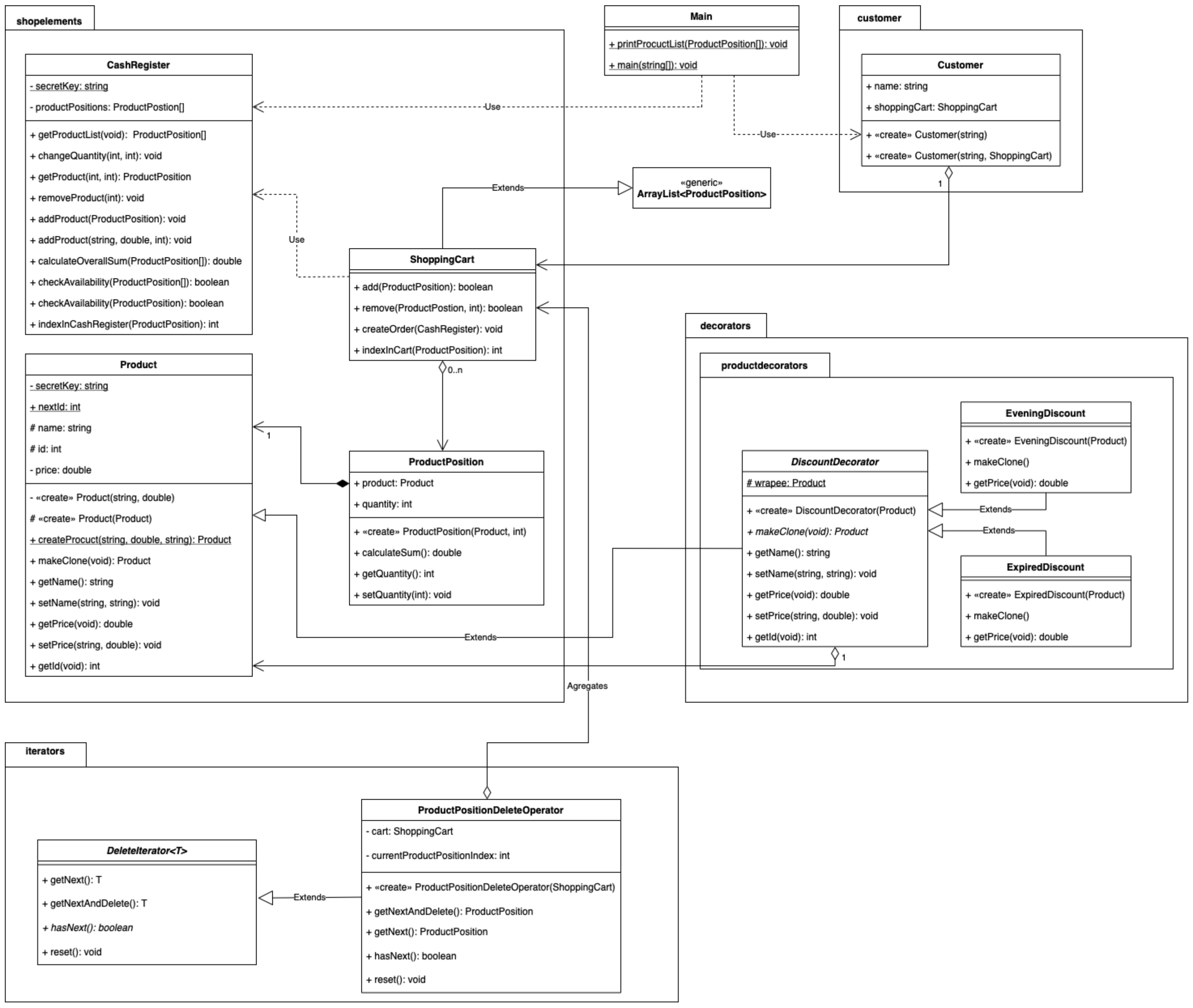
shopelements
CashRegister
- secretKey: string
- productPositions: ProductPostion[]
+ getProductList(void): ProductPosition[]
+ changeQuantity(int, int): void
+ getProduct(int, int): ProductPosition
+ removeProduct(int): void
+ addProduct(ProductPosition): void
+ addProduct(string, double, int): void
+ calculateOverallSum(ProductPosition[]): double
+ checkAvailability(ProductPosition[]): boolean
+ checkAvailability(ProductPosition): boolean
+ indexInCashRegister(ProductPosition): int
Product
- secretKey: string
+ nextId: int
# name: string
# id: int
- price: double
- «create» Product(string, double)
# «create» Product(Product)
+ createProcuct(string, double, string): Product
+ makeClone(void): Product
+ getName(): string
+ setName(string, string): void
+ getPrice(void): double
+ setPrice(string, double): void
+ getId(void): int
ShoppingCart
+ add(ProductPosition): boolean
+ remove(ProductPostion, int): boolean
+ createOrder(CashRegister): void
+ indexInCart(ProductPosition): int
ProductPosition
+ product: Product
+ quantity: int
+ «create» ProductPosition(Product, int)
+ calculateSum(): double
+ getQuantity(): int
+ setQuantity(int): void
Main
+ printProcuctList(ProductPosition[]): void
+ main(string[]): void
customer
Customer
+ name: string
+ shoppingCart: ShoppingCart
+ «create» Customer(string)
+ «create» Customer(string, ShoppingCart)
«generic»
ArrayList<ProductPosition>
Use
Use
Use
Extends
1
0..n
1
decorators
productdecorators
DiscountDecorator
# wrapee: Product
+ «create» DiscountDecorator(Product)
+ makeClone(void): Product
+ getName(): string
+ setName(string, string): void
+ getPrice(void): double
+ setPrice(string, double): void
+ getId(void): int
EveningDiscount
+ «create» EveningDiscount(Product)
+ makeClone()
+ getPrice(void): double
ExpiredDiscount
+ «create» ExpiredDiscount(Product)
+ makeClone()
+ getPrice(void): double
Extends
Extends
Extends
1
Agregates
iterators
DeleteIterator<T>
+ getNext(): T
+ getNextAndDelete(): T
+ hasNext(): boolean
+ reset(): void
ProductPositionDeleteOperator
- cart: ShoppingCart
- currentProductPositionIndex: int
+ «create» ProductPositionDeleteOperator(ShoppingCart)
+ getNextAndDelete(): ProductPosition
+ getNext(): ProductPosition
+ hasNext(): boolean
+ reset(): void
Extends

Figure 1: The UML class diagram of Supermarket Billing Software.