Week 5 Bootcamp

Design Goals

The project is a vehicle booking system, allowing users to perform various actions to book any vehicle among the list of available vehicles. The following design has the goal of adding a new payment functionality for Google Pay and Apple Pay payment methods with the purpose of adding some balance to the User's balance account. Apple Pay should also adopt a payment limitation of $1000. This payment system should be integrated while maintaining SOLID principles.

Design Concept

The presented design for the payment function has a parent abstract class called "Payment" that provides a framework for the payment types to extend from. The two payment methods, GooglePay and ApplePay, inherit from this Payment class as concrete child classes. To enable the menu to display the different payment options, the AddBalanceAction has a new constructor that takes in a Payment type parameter and processes the payment if the payment option is viable, where no limit is exceeded.

Design Rationale

The decision to create Payment as an abstract class rather than a concrete class stems from Payment itself not needing to be instantiated since it is a general object.

Alternatively, Payment could be implemented as an interface, and the benefit is that multiple inheritance can be used for future extension of features such as processing refunds via these payment methods, where Refund could be an additional interface that ApplePay and GooglePay implements. However, Payment in the context of the booking system is more of an entity of the system that represents the payment options and gives permission for a payment to be processed, rather than being a capability of the booking system to process the payment by updating the balance, which is the AddBalanceAction's job. Therefore, Payment was chosen to be an abstract class over an interface.

Since Google Pay and Apple Pay are closely related entities that share the same functionality of adding an amount to the balance, it is appropriate for them to abstract these identities to avoid repetitions (DRY). This parent-child class structure adheres to the Open/Closed Principle (OCP), as the booking system could be extended to offer a new payment option, such as PayPal, by simply adding a new PayPal subclass that inherits the Payment class. An alternative approach could have been to modify AddBalanceAction to print the different payment methods, have switch statements that instantiate the user's chosen payment method as an attribute of the class, followed by an if-else statement that checks whether the payment can proceed with the selected payment method using "instanceof". In this design, adding a PayPal method would require modifications to both the switch statement, as well as the if-else instanceof statements, breaking OCP.

The alternative design mentioned previously also fails Single Responsibility Principle (SRP) due to the AddBalanceAction being responsible for too many things including displaying the payment options, instantiating the selected option, and checking if that option is processable. With the proposed design, SRP is achieved, since the payment methods tell the AddBalanceAction whether or not the payment can be made, leaving AddBalanceAction’s execute() being responsible only for adding to the balance.