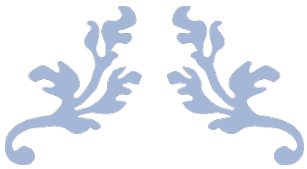

ONLINE GAME SHOP

Project Summary

20 MAY 2018

TEAM MEMBERS:
Pawel Chruscinski, Álvaro López García, Adrian Hamod, Diego Movilla Gayoso, Carlos Bilbao Muñoz, César Carlos Garza Sánchez, Guzmán Garrido Alique

Table of contents

1. Project Architecture

1.1. Multi-tier Architecture

The Online Game Shop is a web application, however, for the Software Engineering project purposes it was designed and partially developed as a desktop application using Java Swing interface and embedded java database – Apache Derby.
The application was designed using well-known architecture model in the industry that consists of multiple layers of abstraction, namely presentation, business logic and integration. The usage of this model provides multiple benefits. It allows modification of components in one layer without having to modify other layers, it encapsulates similar behavior and entities enforcing therefore principles of object-oriented programming such as high cohesion and low coupling. The system with multi-layer architecture is scalable, allows adding new, reusable components as well as is easier to manage and integrate.

1.1.1. General design patterns of the multi-tier architecture used in the project

MVC

The multi-tier architecture of the Online Game Shop is done using the Model-View-Controller design pattern which is one of the best-known pattern used in web application development. It consists of the view that is represented by all the Java Swing classes and is responsible for showing/getting information to/from the user, the model that encapsulates the functionality and data of the application (business logic and integration layers) and the controller that act as an interface between the view and the model. This pattern allows to have different views connected to the same application model and provides means of implementing changes without affecting the rest of the code.

Participating classes:

- All Java Swing classes as view (see presentation layer)
- Controller class as the controller (see presentation layer)
- All classes of the business and integration layer as the model (see business logic and integration layers)

Data Transfer Object

Transfer objects are used to transport data between layers. They encapsulate different attributes into entities that can be easily passed from one layer to another. They are serializable objects with attributes, getters and setters. This generalization of the data representation ensures the format that can be processed by all layers. Future improvements of the application may involve integration with ORM (Object Relational Mapping) system that adds an extra layer between the integration and the application for automatic mapping of the data from database management system into the data transfer objects. Currently, the integration layer imitates this behavior and creates the transfer objects before returning the information to the business logic layer.

Participating classes:

- User class that holds information about the users, including personal details.
- Item class that encapsulates information about the game products such as name, price or description.

1.2. The Three Layers

1.2.1. Presentation

It encapsulates the logic and mechanics of the presentation (the view of the application) that provide services for clients accessing and interacting with the application. It consists of Java Swing classes that build the user interface as well as the controller class that acts as an interface between the Swing classes and the business logic tier. Controller receives calls from Swing interface as well as updates the presentation with information received from the business layer. User experience is a major factor that influences the design choices for this layer.

Classes included in presentation layer:

- All Java Swing classes (MainView, all Frames and Table implementations)
- Controller

1.2.2. Design patterns used in presentation layer

Observer

It is used commonly in user interfaces where multiple components of the interface can "listen" to any changes made to an object. In the case of the Online Game Shop, numerous tables in Swing components are registered as observers of the controller so the controller can update all the views when necessary.

Participating classes:

- ShopObserver interface that is implemented by the CartTable and BrowseTable classes allowing for the controller to notify them about the data change
- Observable interface implemented by the controller adding functionality to register observers and remove observers

1.2.3. Business Logic

This layer provides the application services to the presentation layer. It is build using classes that contain the logic of the application that serve as an intermediate between the user interface and the data storage. The layer gives access to its services using Application Service class that manages the calls to all other business layer classes.

Classes included in business logic layer:

- MyServiceApplication
- All Logic classes (OrderLogic, UserLogic, ReportLogic, etc)
- Business Objects such as Cart or BasicOrder (together with decorator classes)

1.2.4. Design patterns used in business logic layer

Application Service

This design pattern is used in the Online Game Shop to group the functionalities of the business layer and provide single-point access to services that have their logic spread among several business classes and objects. The centralization of the business logic prevents code duplication and provides an interface for the clients which in the case of the Online Game Shop is the controller of presentation layer.

Participating classes:

- ServiceApplicationInterface - which is an application service interface declaring all methods of the business logic services.
- MyServiceApplication – the concrete implementation of the application service that delegates the services to appropriate business logic classes.

Façade

The application service that is used in the Online Game Shop is also an example of a design pattern called Façade. It hides all the components of the subsystems inside the business layer and reduces the number of objects that are created when client accesses the layer. It allows for new implementations of the business logic to be added without the need to expose any changes to the client.

Participating classes:

- ServiceApplicationInterface – the general interface declaring the methods for accessing the subsystems.
- MyService Application – the concrete implementation with methods that allows access to the application subsystems.

Iterator

Iterator pattern is used to be able to traverse through a list of objects without exposing the internal structure of the object. In the Online Game Shop the iterator is used in the Cart class to be able to iterate over the cart without the necessity of accessing the internal list of items.

Participating classes:

- Iterator – generic interface for methods to iterate over the object.
- Iterable – generic interface that allows an object to be iterated over.
- CartIterator – inner class inside the Cart class that implements the Iterator interface and its methods.

- Cart class implements the Iterable interface to allow the iteration over itself and therefore over the items in its internal list.

Decorator

Decorator pattern allows a dynamic addition of behavior into the objects. It is more flexible then using multiple static inheritance. In the Online Game Shop, the basic order that consist of the total price of the order and the textual summary, can be further "decorated" with additional functionalities that includes charity donation, fast delivery, pack as gift and discount and the functionality of the textual description and price calculation changes depending on the actual decorators applied.

Participating classes:

- Order – interface of the Order object to which additional functionalities can be added
- BasicOrder – concrete order object to which additional functionalities can be added
- OrderDecorator – class that has a reference to Order object and act as an interface by forwarding the requests to Order object.
- Conrete decorator classes (OrderDiscount, OrderDonation, OrderFastDelivery, OrderAsGift) add functionalities to the Order object

1.2.5. Integration

The layer is responsible for the transfer of data between the application and the database systems. It allows for any application to connect to database as well as provides an interface to database systems. The Online Game Shop uses an embedded Apache Derby database, however the architecture of the integration layer allows for easy integration of the application with other database management systems, such as Oracle or MySQL.

Classes included in integration layer:

- DAO factories (DerbyDAOFactory and abstract DAOFactory)
- Concrete DAO objects (DerbyUserDAO, DerbyItemDAO, DerbyOrderDAO)

1.2.6. Design patterns used in integration layer

DAO

The DAO design pattern (Data Access Object) is used in the application to separate the data access in database from the data process logic located in business layer. It provides an interface to data access in such way that the application does not depend on the actual database management system and allows for many types of databases to be connected in the future. Currently the Apache Derby embedded database is used in the application, however other systems can be integrated with ease, for example Oracle, MySQL or even non-relational database systems.

The DAO pattern uses set of interfaces that can be then implemented by concrete database access implementation.

Participating classes:

- UserDAO, ItemDAO, OrderDAO interfaces and all concrete implementations which currently are DerbyUserDAO, DerbyItemDAO and DerbyOrderDAO.

Abstract Factory

The Data Access Object design pattern is enhanced even further by taking advantage of the abstract factory pattern. It provides an interface to create families of objects. In the Online Game Shop application, the abstract DAO factory manages creation of the concrete DAO factory depending on the database implementation. Moreover, each concrete factory manages the creation of the concrete DAO objects that are used to access the data.

Participating classes:

- DAOFactory – abstract factory class that creates the family of objects, in this case the family of DAO objects. At the moment the only family in the application is the DerbyDAO objects that relate to data access through the Apache Derby database, but others can be easily added in the future such as Oracle or MySQL.
- DerbyDAOFactory – factory that manages the creation of the concrete DAO objects, such as DerbyUserDAO, DerbyItemDAO, DerbyOrderDAO.

2. Analysis and Design

2.1. Sequence Diagrams

UC1: Administrator adds new stock item
UC2: Application automatically notifies customers about due rentings by e-mails
UC3: Administrator downloads sales/rents reports for finance purposes
UC4: Administrator displays pending customer order
UC5: Administrator processes returns
UC6: Administrator deletes scrapped stock
UC7: Customer adds an item to the cart
UC8: No longer valid
UC9: Customer reviews and rates a product
UC10: Customer searches for a specific product
UC11: Customer browses the catalogue of products
UC12: Administrator registers to the system
UC13: Customer registers to the system
UC14: Customer or administrator logs into the system
UC15: Customer checks out
UC16: Customer updates his account details
UC17: Administrator updates his account details
UC18: Customer accesses history of items bought/rented (and returned)

UC19: Customer accesses a list of currently rented products
UC20: Administrator downloads stock report
UC21: Customer adds to cart an item already rented
UC22: Administrator updates the number of copies of an item in stock
UC23: Administrator accepts a customer order
UC24: Administrator cancels a customer order
UC25: Administrator downloads customized sales/rents reports for financial purposes
UC26: Customer renews rental of a game

2.2. Activity Diagrams

UC12:
An existing administrator logs in the system after the admin homepage is displayed. Only if a customer or administrator logs, the 'register as an admin' button can be pressed. After all the information for the new administrator is completed you can press the create account button. An activation email is sent to the new administrator and after it presses the link to activate the account it will have to change the password and will be able to log into the system as an administrator.

UC15:
Customer chooses if either he wants to continue shopping or if he wants to proceed to the checkout. If the latter option is chosen, then the user is asked to fill the payment information or use the already saved information from previous times. He can as well choose if he wants to save said info for future uses. Finally, after the payment is accepted/rejected the user is notified and an email is sent with the order information.

2.3. Use Case Diagrams

Customer:
The customer has to be previously logged into the system (UC14) to do any of the actions. If the user is not registered in order to log into the system, it will have to register (UC13). It can access the history of the items bought or rented (UC18) and review and rate any of those products (UC9). They can be reviewed and rated after the customer accesses the list of currently rented products (UC19) and can renew the rental of any game (UC26). The customer can browse the catalogue of products (UC11) and add any of those items to the cart (UC7). They can get to those items by searching a specific product (UC10). and add to the cart an item already rented (UC21). After all this the customer can check out (UC15). It will be able to update its account at any time (UC16).

Server:
The server automatically notifies the customers about due renting via email (UC2).

Admin:
The administrator can display all pending customer orders (UC4) and accept (UC23) of cancel (UC24) any of those orders. The admin can also download the customized sales

and rents reports for finance purposes (UC25), update its account (UC17), process returns (UC5), add new stock item (UC1), download the sales and rents reports for finance purposes (UC3), update the number of copies of an item in stock (UC22), delete scrapped stock (UC6) and download the stock report (UC20). The administrator has to be previously registered in the system (UC12) and log into it (UC14).