



Hotel Management System

DESIGN PATTERNS (SE-408)

Abdul Haseeb(SE-16050)

Bilal Ahmad (SE-16048)

Hafiz Ahmed Abdullah (SE-16067)

Syed Raffay Hussain (SE-16301)

INTRODUCTION

OBJECTIVE:

A hotel management system that will facilitate the users and customers to easily book different types of rooms according to their needs. With a very flexible design to easily incorporate new rooms and other features

PROJECT SCOPE:

The scope of this project is to provide users and hotels with a simple yet flexible hotel management system. That a customer can use to reserve a room with ease and comfort.

PROJECT PURPOSE:

Use of different design patterns to build a real-world project application to light up the importance of Design patterns.

SALIENT FEATURES

- Allow users to reserve rooms through ease and comfort.
- Select a different kind of rooms according to your choice.
- Choose the different kinds of services and features that you need in your booked room.
- Payment right on the moment hiding all the complexity of payment features and giving users a very simple and clean interface.
- Allows Hotel Managers to easily add as many kinds of rooms as they want.
- Allows managers to add ad many different services in a specific room without breaking it.

TECHNOLOGIES USED

Operating System:	Windows 10
Language:	JAVA
IDE:	IntelliJ Idea

DESIGN PATTERNS IMPLEMENTED

Factory Pattern:

The factory Design pattern is being used in the project to create different kinds of rooms including Single bed or Double Bed. Moreover, you can easily add other different kinds of rooms in the future easily without breaking the code.

Facade Pattern:

The facade design pattern is being used to handle all the payment options. It provides a very simple User interface to the user with simple options including cash withdraw, balance check, etc. and hiding all the complexity behind the scenes.

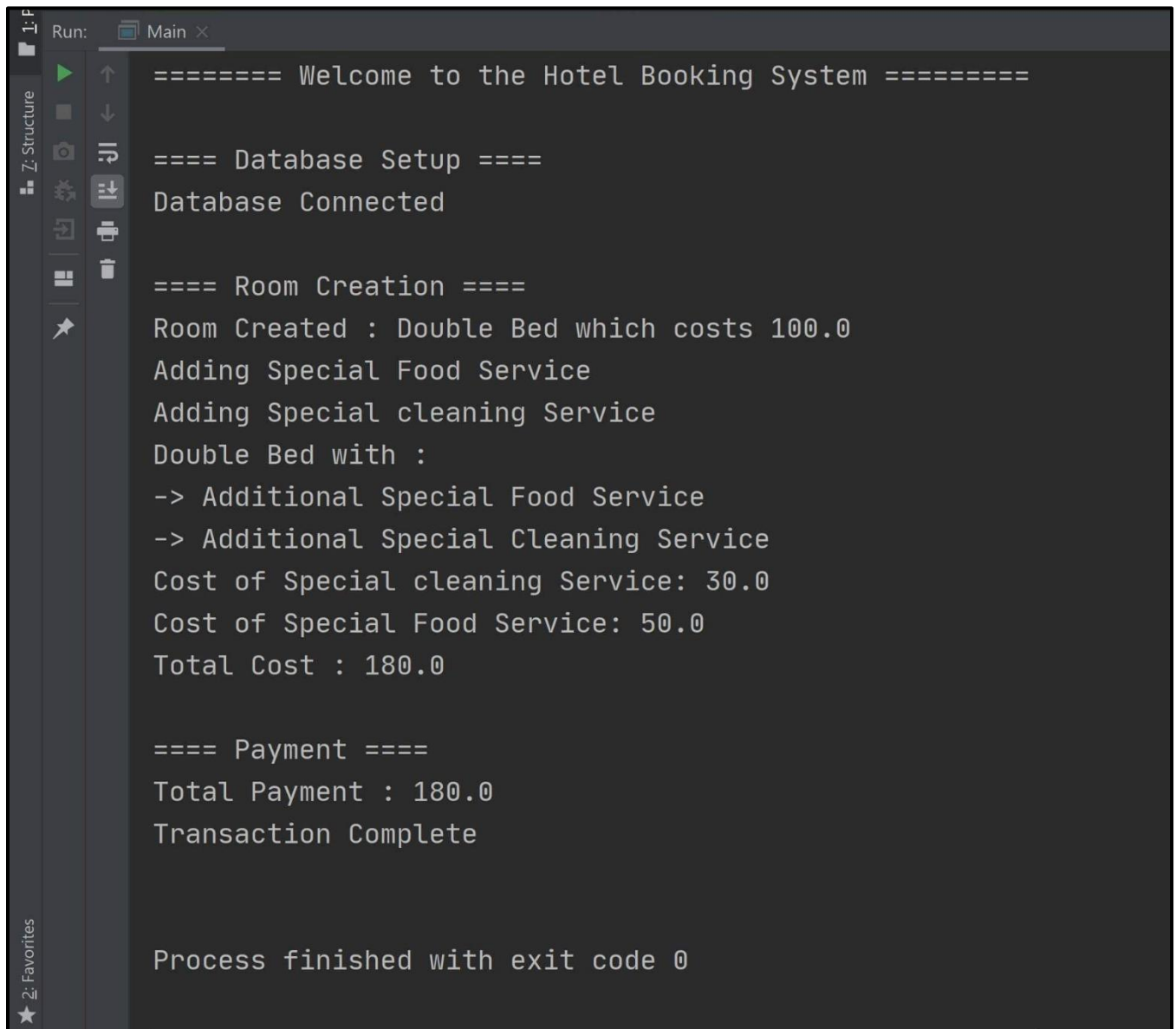
Decorator Pattern:

The Decorator pattern allows adding additional optional features and services to the room whether its a single bed or double bed. You can add additional food service or cleaning service if you want. Moreover, in the future, you can add other services too without breaking the code.

Singleton Pattern:

The Singleton pattern is used to keep a single instance of the database no matter from where ever you create database object it will be returning the same instance to ensure users only work with a single instance of the database.

Project Output

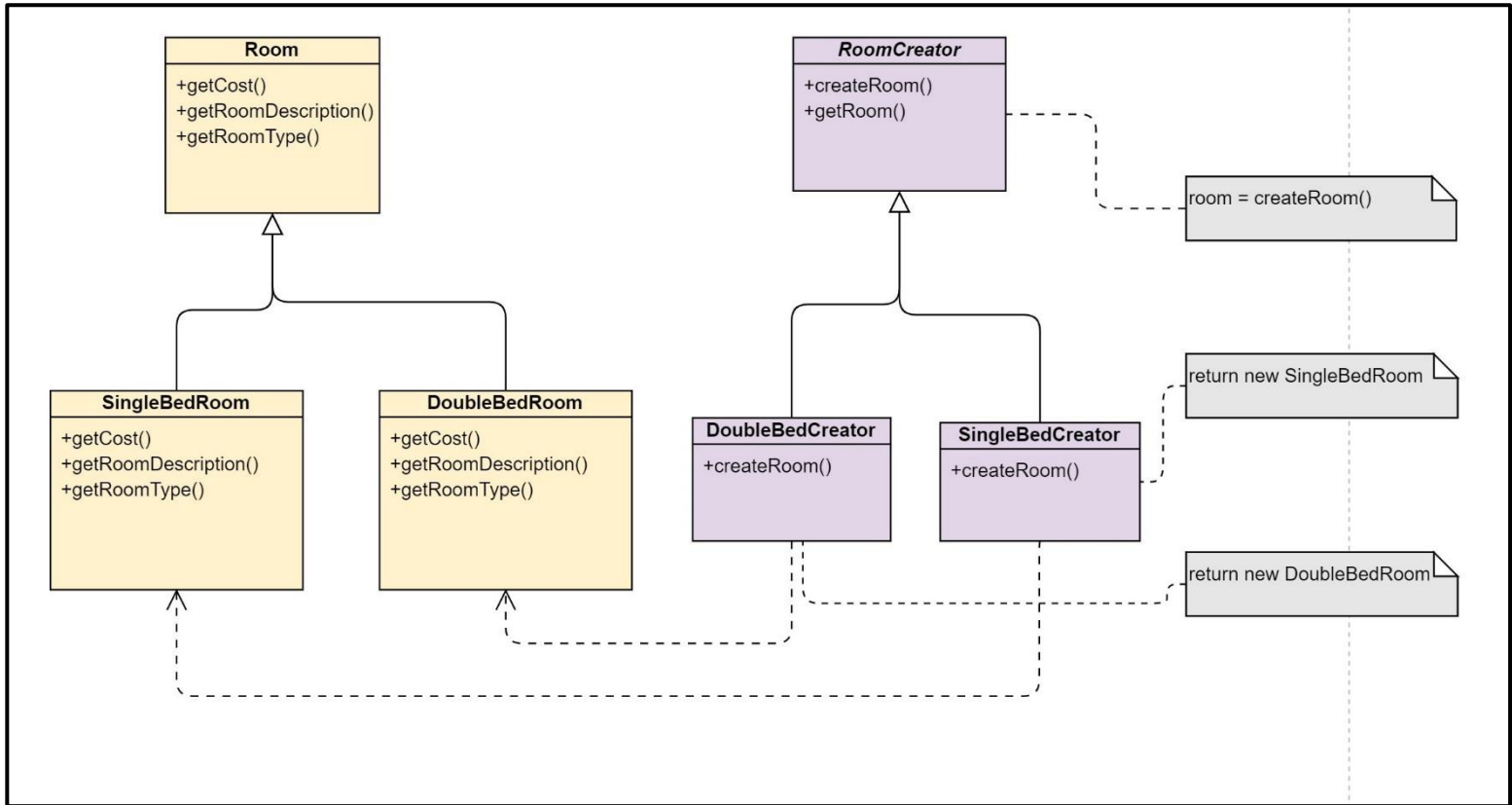


The screenshot shows a Java IDE with a dark theme. On the left, there is a sidebar with icons for 'Run', 'Structure', and 'Favorites'. The main area displays the output of a program named 'Main'. The output text is as follows:

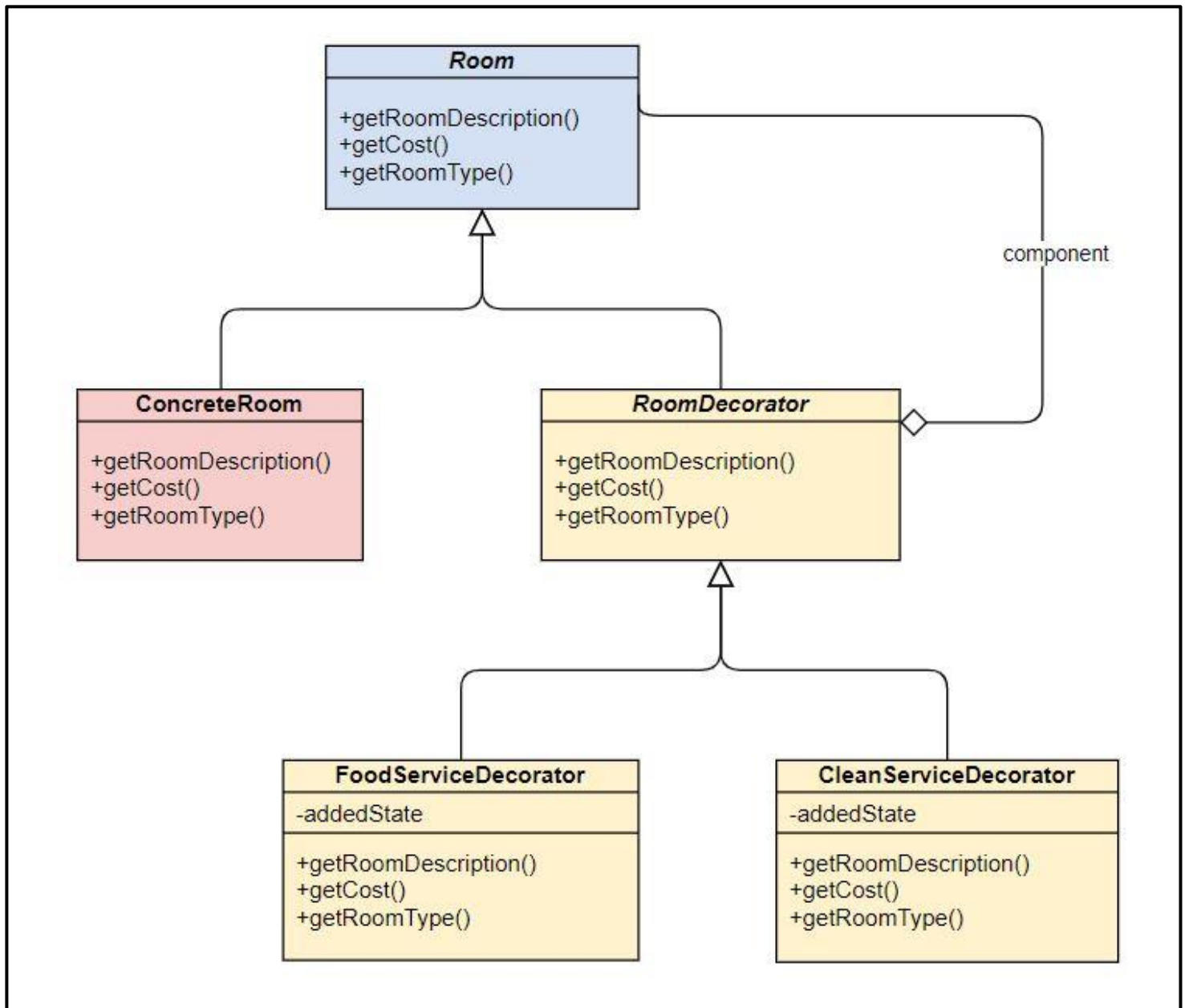
```
===== Welcome to the Hotel Booking System =====  
  
==== Database Setup ====  
Database Connected  
  
==== Room Creation ====  
Room Created : Double Bed which costs 100.0  
Adding Special Food Service  
Adding Special cleaning Service  
Double Bed with :  
-> Additional Special Food Service  
-> Additional Special Cleaning Service  
Cost of Special cleaning Service: 30.0  
Cost of Special Food Service: 50.0  
Total Cost : 180.0  
  
==== Payment ====  
Total Payment : 180.0  
Transaction Complete  
  
Process finished with exit code 0
```

CLASS DIAGRAMS

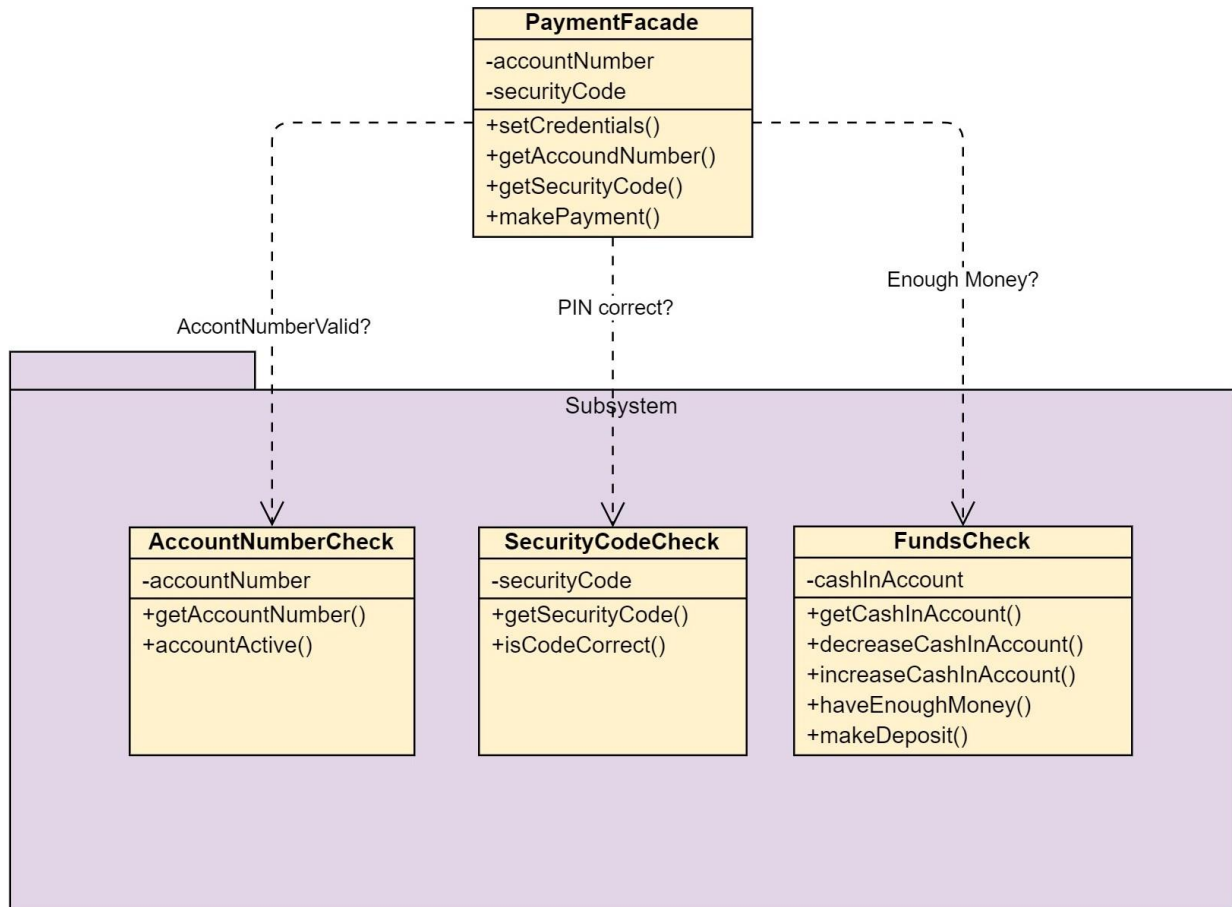
Factory Pattern:



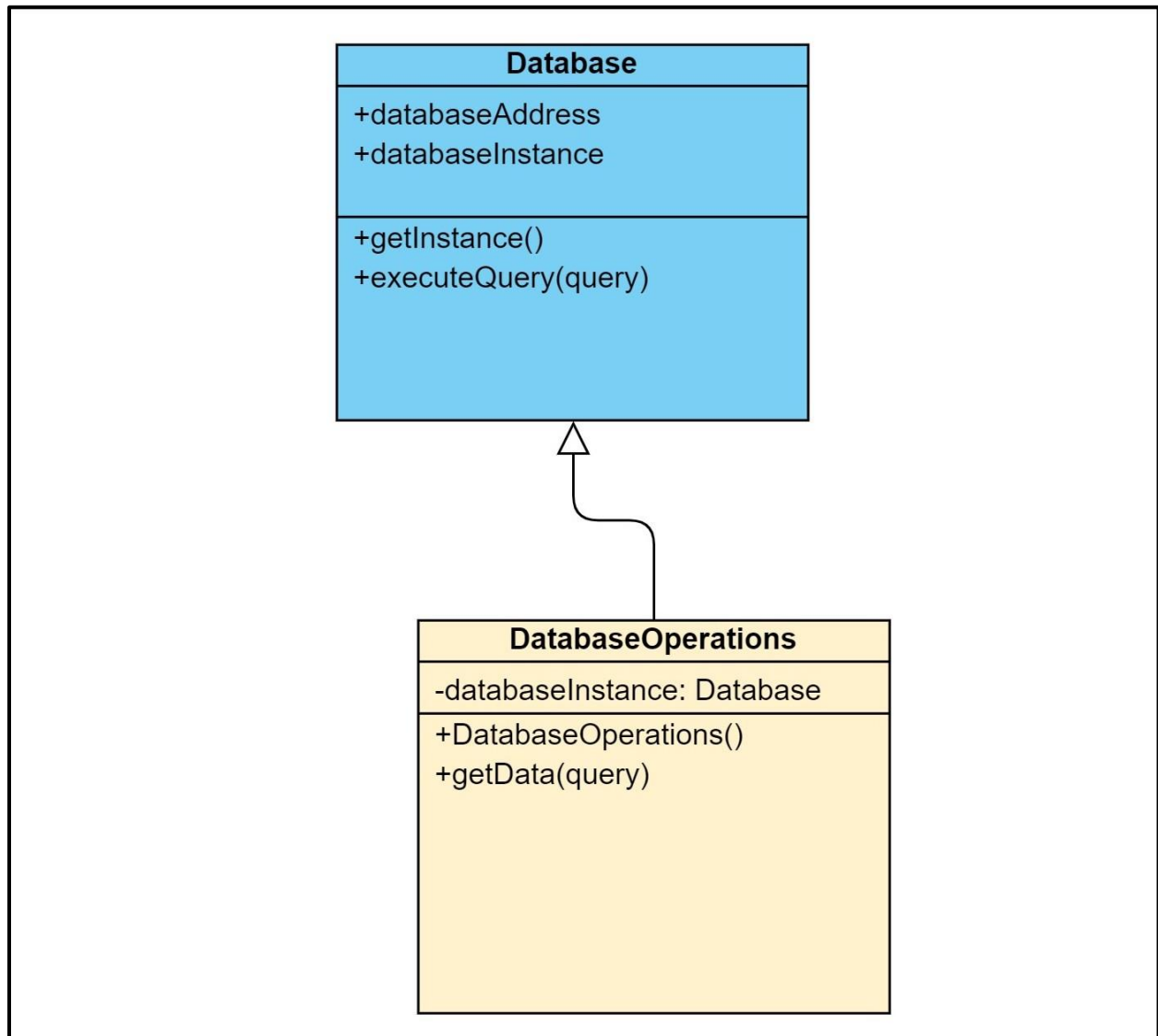
Decorator Pattern:



Facade Pattern:



Singleton Pattern:



CODE

Main.java

```
package com.company;

import com.company.Decorator.CleanServiceDecorator;
import com.company.Decorator.FoodServiceDecorator;
import com.company.Facade.PaymentFacade;
import com.company.Factory.RoomFactory.DoubleBedCreator;
import com.company.Factory.RoomFactory.RoomCreator;
import com.company.Factory.RoomFactory.SingleBedCreator;
import com.company.Room.Room;
import com.company.Singleton.Database;
import com.company.Singleton.DatabaseOperations;

public class Main {

    private static RoomCreator roomCreator;
    public static double totalAmountToBePaid = 0;
    private static Database databaseInstance;
    private static DatabaseOperations databaseOperations;

    public static void main(String[] args) {
        System.out.println("\n\n===== Welcome to the Hotel Booking System =====\n\n");

        handleDatabaseConnection();
        handleRoomCreation(1); // 0 means singleBed, 1 means doubleBed
        handlePayment();
    }

    public static void handleRoomCreation(int type) {
        System.out.println("\n==== Room Creation ====");
        if (type == 0) {
            roomCreator = new SingleBedCreator();
        } else {
            roomCreator = new DoubleBedCreator();
        }

        System.out.println("Room Created : " + roomCreator.getRoom().getRoomType() +
            " which costs "
            + roomCreator.getRoom().getCost());
    }
}
```

```
        Room decoratedRoom = new CleanServiceDecorator(new FoodServiceDecorator(roomC
reator.getRoom()));

        System.out.println(decoratedRoom.getRoomDescription());
        totalAmountToBePaid = decoratedRoom.getCost();
        System.out.println("Total Cost : " + totalAmountToBePaid);

    }

    public static void handlePayment() {
        System.out.println("\n==== Payment ====");

        PaymentFacade payment = new PaymentFacade();
        payment.setCredentials(12345678, 1234);
        payment.makePayment(totalAmountToBePaid);
    }

    public static void handleDatabaseConnection() {
        System.out.println("==== Database Setup ====");
        databaseInstance = Database.getInstance("Database_name");
        databaseOperations = new DatabaseOperations(databaseInstance);

        if (databaseInstance != null) {
            System.out.println("Database Connected");
        } else {
            System.out.println("Database Connection Failed");
        }
    }
}
```

Database.java

```
Database.java
com > company > Singleton > Database.java
1  package com.company.Singleton;
2
3  public class Database {
4      private static volatile Database databaseInstance;
5      public String databaseAddress;
6
7      private Database(String databaseAddress) {
8          this.databaseAddress = databaseAddress;
9      }
10
11     public static Database getInstance(String databaseAddress) {
12         if (databaseInstance == null) {
13             synchronized (Database.class) {
14                 if (databaseInstance == null) {
15                     databaseInstance = new Database(databaseAddress);
16                 }
17             }
18             return databaseInstance;
19         }
20         return databaseInstance;
21     }
22
23     public String executeQuery(String query) {
24         // pass query to the database using the connection made
25         return "Data Fetched";
26     }
27 }
```

DatabaseOperations.java

```
DatabaseOperations.java X
com > company > Singleton > DatabaseOperations.java
1  package com.company.Singleton;
2
3  public class DatabaseOperations {
4      Database databaseInstance;
5
6      public DatabaseOperations(Database instance) {
7          databaseInstance = instance;
8      }
9
10     public void getData(String query) {
11         databaseInstance.executeQuery(query);
12     }
13 }
```

Room.java

```
Room.java X
com > company > Room > Room.java
1  package com.company.Room;
2
3  public interface Room {
4      public double getCost();
5
6      public String getRoomDescription();
7
8      public String getRoomType();
9
10 }
```

SingleBedRoom.java

```
SingleBedRoom.java X
com > company > Room > SingleBedRoom.java
1  package com.company.Room;
2
3  public class SingleBedRoom implements Room {
4      @Override
5      public double getCost() {
6          return 50.00;
7      }
8      @Override
9      public String getRoomDescription() {
10         return "Single Bed with ";
11     }
12     @Override
13     public String getRoomType() {
14         return "Single Bed";
15     }
16 }
```

DoubleBedRoom.java

```
DoubleBedRoom.java X
com > company > Room > DoubleBedRoom.java
1  package com.company.Room;
2
3  public class DoubleBedRoom implements Room {
4      @Override
5      public double getCost() {
6          return 100.00;
7      }
8      @Override
9      public String getRoomDescription() {
10         return "Double Bed with : ";
11     }
12     @Override
13     public String getRoomType() {
14         return "Double Bed";
15     }
16 }
```

RoomCreator.java

```
RoomCreator.java X
com > company > Factory > RoomFactory > RoomCreator.java
1  package com.company.Factory.RoomFactory;
2  import com.company.Room.*;
3
4  public abstract class RoomCreator {
5      public Room getRoom() {
6          Room room = createRoom();
7          return room;
8      }
9      public abstract Room createRoom();
10 }
```

SingleBedCreator.java

```
SingleBedCreator.java X
com > company > Factory > RoomFactory > SingleBedCreator.java
1 package com.company.Factory.RoomFactory;
2 import com.company.Room.*;
3
4 public class SingleBedCreator extends RoomCreator {
5     @Override
6     public Room createRoom() {
7         return new SingleBedRoom();
8     }
9 }
```

DoubleBedCreator.java

```
DoubleBedCreator.java X
com > company > Factory > RoomFactory > DoubleBedCreator.java
1 package com.company.Factory.RoomFactory;
2 import com.company.Room.*;
3
4 public class DoubleBedCreator extends RoomCreator {
5     @Override
6     public Room createRoom() {
7         return new DoubleBedRoom();
8     }
9 }
```

WelcomeToBank.java

```
WelcomeToBank.java X
com > company > Facade > WelcomeToBank.java
1 package com.company.Facade;
2
3 public class WelcomeToBank {
4     public WelcomeToBank() {
5         System.out.println("Welcome to ABC Bank");
6         System.out.println("We are happy to give you your money if we can find it\n");
7     }
8 }
9
```

SecurityCodeCheck.java

☞ SecurityCodeCheck.java X

com > company > Facade > ☞ SecurityCodeCheck.java

```
1  package com.company.Facade;
2
3  public class SecurityCodeCheck {
4      private int securityCode = 1234;
5
6      public int getSecurityCode() {
7          return securityCode;
8      }
9
10     public boolean isCodeCorrect(int secCodeToCheck) {
11         if (secCodeToCheck == getSecurityCode()) {
12             return true;
13         } else {
14             return false;
15         }
16     }
17 }
```

PaymentFacade.java

```
package com.company.Facade;

public class PaymentFacade {
    private int accountNumber;
    private int securityCode;
    AccountNumberCheck acctChecker;
    SecurityCodeCheck codeChecker;
    FundsCheck fundChecker;

    public PaymentFacade() {
        acctChecker = new AccountNumberCheck();
        codeChecker = new SecurityCodeCheck();
        fundChecker = new FundsCheck();
    }

    public void setCredentials(int newAcctNum, int newSecCode) {
        accountNumber = newAcctNum;
        securityCode = newSecCode;
    }
}
```

```

    public int getAccountNumber() {
        return accountNumber;
    }

    public int getSecurityCode() {
        return securityCode;
    }

    public void withdrawCash(double cashToGet) {
        if (acctChecker.accountActive(getAccountNumber()) && codeChecker.isCodeCorrect(getSecurityCode())
            && fundChecker.haveEnoughMoney(cashToGet)) {
            System.out.println("Transaction Complete\n");
        } else {
            System.out.println("Transaction Failed\n");
        }
    }

    public void makePayment(double payment) {
        if (acctChecker.accountActive(getAccountNumber()) && codeChecker.isCodeCorrect(getSecurityCode())) {
            System.out.println("Total Payment : " + payment);
            fundChecker.makeDeposit(payment);
            System.out.println("Transaction Complete\n");
        } else {
            System.out.println("Transaction Failed\n");
        }
    }

    public void depositCash(double cashToDeposit) {
        if (acctChecker.accountActive(getAccountNumber()) && codeChecker.isCodeCorrect(getSecurityCode())) {
            fundChecker.makeDeposit(cashToDeposit);
            System.out.println("Transaction Complete\n");
        } else {
            System.out.println("Transaction Failed\n");
        }
    }
}

```


FundsCheck.java

FundsCheck.java X

com > company > Facade > FundsCheck.java

```
1  package com.company.Facade;
2
3  public class FundsCheck {
4      private double cashInAccount = 1000.00;
5
6      public double getCashInAccount() {
7          return cashInAccount;
8      }
9
10     public void decreaseCashInAccount(double cashWithdrawn) {
11         cashInAccount -= cashWithdrawn;
12     }
13
14     public void increaseCashInAccount(double cashDeposited) {
15         cashInAccount += cashDeposited;
16     }
17
18     public boolean haveEnoughMoney(double cashToWithdrawal) {
19         if (cashToWithdrawal > getCashInAccount()) {
20             System.out.println("Error: You don't have enough money");
21             System.out.println("Current Balance: " + getCashInAccount());
22             return false;
23         } else {
24             decreaseCashInAccount(cashToWithdrawal);
25             System.out.println("Withdrawal Complete: Current Balance is " + getCashInAccount());
26             return true;
27         }
28     }
29     public void makeDeposit(double cashToDeposit) {
30         increaseCashInAccount(cashToDeposit);
31     }
32 }
```

AccountNumberCheck.java

```
AccountNumberCheck.java X
com > company > Facade > AccountNumberCheck.java
1  package com.company.Facade;
2
3  public class AccountNumberCheck {
4      private int accountNumber = 12345678;
5
6      public int getAccountNumber() {
7          return accountNumber;
8      }
9
10     public boolean accountActive(int acctNumToCheck) {
11         if (acctNumToCheck == getAccountNumber()) {
12             return true;
13         } else {
14             return false;
15         }
16     }
17 }
18
```

RoomDecorator.java

```
RoomDecorator.java X
com > company > Decorator > RoomDecorator.java
1  package com.company.Decorator;
2  import com.company.Room.*;
3
4  abstract class RoomDecorator implements Room {
5      protected Room tempRoom;
6
7      public RoomDecorator(Room newRoom) {
8          tempRoom = newRoom;
9      }
10
11     public String getRoomDescription() {
12         return tempRoom.getRoomDescription();
13     }
14
15     public double getCost() {
16         return tempRoom.getCost();
17     }
18
19     public String getRoomType() {
20         return tempRoom.getRoomType();
21     }
22 }
```

ConcreteRom.java

```
ConcreteRoom.java X
com > company > Decorator > ConcreteRoom.java
1 package com.company.Decorator;
2 import com.company.Room.*;
3
4 public class ConcreteRoom implements Room {
5     public double getCost() {
6         System.out.println("Cost of Concrete Room: " + 150.0);
7         return 150.0;
8     }
9     public String getRoomDescription() {
10        return "A comfortable Single Bed room, with sunshine view and suiteable temperature";
11    }
12    public String getRoomType() {
13        return "Single Bed";
14    }
15 }
```

FoodServiceDecorator.java

```
FoodServiceDecorator.java X
com > company > Decorator > FoodServiceDecorator.java
1 package com.company.Decorator;
2 import com.company.Room.*;
3
4 public class FoodServiceDecorator extends RoomDecorator {
5     public FoodServiceDecorator(Room newRoom) {
6         super(newRoom);
7         System.out.println("Adding Special Food Service");
8     }
9     public String getRoomDescription() {
10        return tempRoom.getRoomDescription() + "\n-> Additional Special Food Service";
11    }
12    public double getCost() {
13        System.out.println("Cost of Special Food Service: " + 50.0);
14        return tempRoom.getCost() + 50.0;
15    }
16    public String getRoomType() {
17        return tempRoom.getRoomType();
18    }
19 }
```

CleanServiceDecorator.java

```
CleanServiceDecorator.java X
com > company > Decorator > CleanServiceDecorator.java
1 package com.company.Decorator;
2 import com.company.Room.*;
3
4 public class CleanServiceDecorator extends RoomDecorator {
5     public CleanServiceDecorator(Room newRoom) {
6         super(newRoom);
7         System.out.println("Adding Special cleaning Service");
8     }
9
10    public String getRoomDescription() {
11        return tempRoom.getRoomDescription() + "\n-> Additional Special Cleaning Service";
12    }
13
14    public double getCost() {
15        System.out.println("Cost of Special cleaning Service: " + 30.0);
16        return tempRoom.getCost() + 30.0;
17    }
18
19    public String getRoomType() {
20        return tempRoom.getRoomType();
21    }
22
23 }
24
```