Hotel Management Software
Group 4

Table Of Contents:

<<Section 1 - Software Specifics>>

1.1 <<Title>>

Hotel Management Software (HMS)
Prepared by: Queens College Software Engineers (QCSE)
Prepared for: Hotel Inc. (HI)

1.2 <<Group Members>>

Group 4

- Samuel Mirakov (Captain) - Singleton Pattern
- Uneeb Siddiqi - Memento Pattern
- Redwanul Haque - Builder Pattern
- Ke Dong - Strategy Pattern
- Xiaoqian He - Observer Pattern
- Mahfuz Uddin - Decorator Pattern

1.3 <<Project Description>>

This is a Requirements Specification document for Hotel Inc, HI. HI operates a chain of hotels throughout the United States. Hotel Management Software, HMS, allows customers to book hotels online, see/edit their current reservation, cancel a reservation, and much more. Staff and customers will access HMS through a web page. Staff members update the billings for each purchase made by the customer and the accumulating bill is given at the end of the customer's visit.

1.4 <<Purpose (why the customer requires this project)>>

HMS allows customers to book online. Customers no longer need to walk into a hotel to book or call the hotel, they can just book online or through the HMS application. This saves customers an enormous amount of time. The convenience of allowing customers to book a room(s) will save hotel staff time thus requiring fewer receptionists and hence reducing operational costs. This will also bring an increase in customers into hotels associated with this application software, due to its usability.

1.5 <<Business Drivers/Business Model>>

HMS wants Hotel Inc. to maximize profits and compete with the biggest chains in the industry. By automating this booking process, we can reduce operation costs by as much as 25% and increase profits with a margin of at least 30% within the first year of launching. We will provide updates to this software annually to provide a better experience for customers, hence, increasing profits and decreasing operational costs even more [for hotel management]. This software will be simple to use, as a result, long hours of training will not be required.

1.6 <<Scope>>

HMS will include the design of a website as well as smartphone applications for both Apple and Android users, available in the Apple Store and Google Play Store. HMS will not include any hosting or database service, however, we will set up the database and hosting for the parent company, Hotel Inc. HI will be required to identify where they want to host and store their database. Choices will include either Azure or AWS. Other Hotel companies will be able to associate themselves with the use of this application but will have to agree to HMS Terms of Service and Privacy Policy.

1.7 Definitions and/or Acronyms>>

1. Azure or AWS provides cloud computing services as well as hundreds of other services via the internet such as hosting, warehousing, computing, or analyzing.
2. A web server stores and delivers the content for a website, such as videos, images, text, and application data.

<<Section 2 - Diagrams and Graphs>>

2.1 <<Use Case>>

- **A customer decides to book a reservation Online on any given day**
- **Customer enters first and last name**
- **Customer enters 8-digit account number**
- **Customer enters hotel location**
- **Customer enters the amount of days they will stay at the hotel**
- **Customer enters the amount of rooms they want**
- **Customer enters the amount of beds they require per room with a limit of 2 per room**
- **Customer decides which method of payment they will use (after being shown total price)**

Use Case 1	Make reservation
Actor	Customers
Use-Case Overview	Customers need a place to stay and create an online reservation for a hotel. Customers fill out their personal information, as well as preferences of hotel locations, to secure a place to stay at. Customers will be presented with their accumulative bill and will choose their preference of payment.
Precondition	Customers need a place to stay.
Trigger	Customer decides to create an online reservation instead of a walk-in.

Basic Flow: Create Online Reservation

Description	**This scenario describes the situation where creating an online reservation and the number of rooms needed is available for the customer. This is the main success scenario.**
1	Customer enters a first and last name
2	Customer enters 8 digit account number
3	Customer chooses hotel location
4	Customer enters the amount of days they will stay at the hotel
5	Customer picks the amount of rooms they want
6	Customer enters the amount of beds they require per room with a limit of 2 per room
7	Customer decides which method of payment they will use (after being shown total price)
Termination Outcome	Customers have successfully booked an online reservation.

Alternative Flow: The rooms that the Customer wants are unavailable

Description	**This scenario describes the situation where no rooms are available for customers to book.**
Termination Outcome	The next date, when a room is available, customers will be notified and provided or other hotel locations where rooms are available, are recommended to the customers.

2.2 <<Acceptance Criteria>>

FirstName	LastName	AccountNumber	Hotel Location	Stay duration	Number of rooms to book	Number of beds (max 2)	Payment Method	Expected result	Expected Message
John	**Doe**	**12345678**	**New York**	**2 nights**	**1**	**3**	**Apple Pay**	**Fail**	**There are no rooms with more than 2 beds.**
John	**Alex**	**12345656**	**NewYork**	**1 night**	**1**	**1**	**GooglePay**	**Pass**	

2.3 <<Assumptions and Constraints>>

Assumptions - Some of the group members won’t participate. Most of the group members will have tests and projects from other classes to veer their focus from this project. Nobody will get sick. We will hold group meetings every few days for updates.

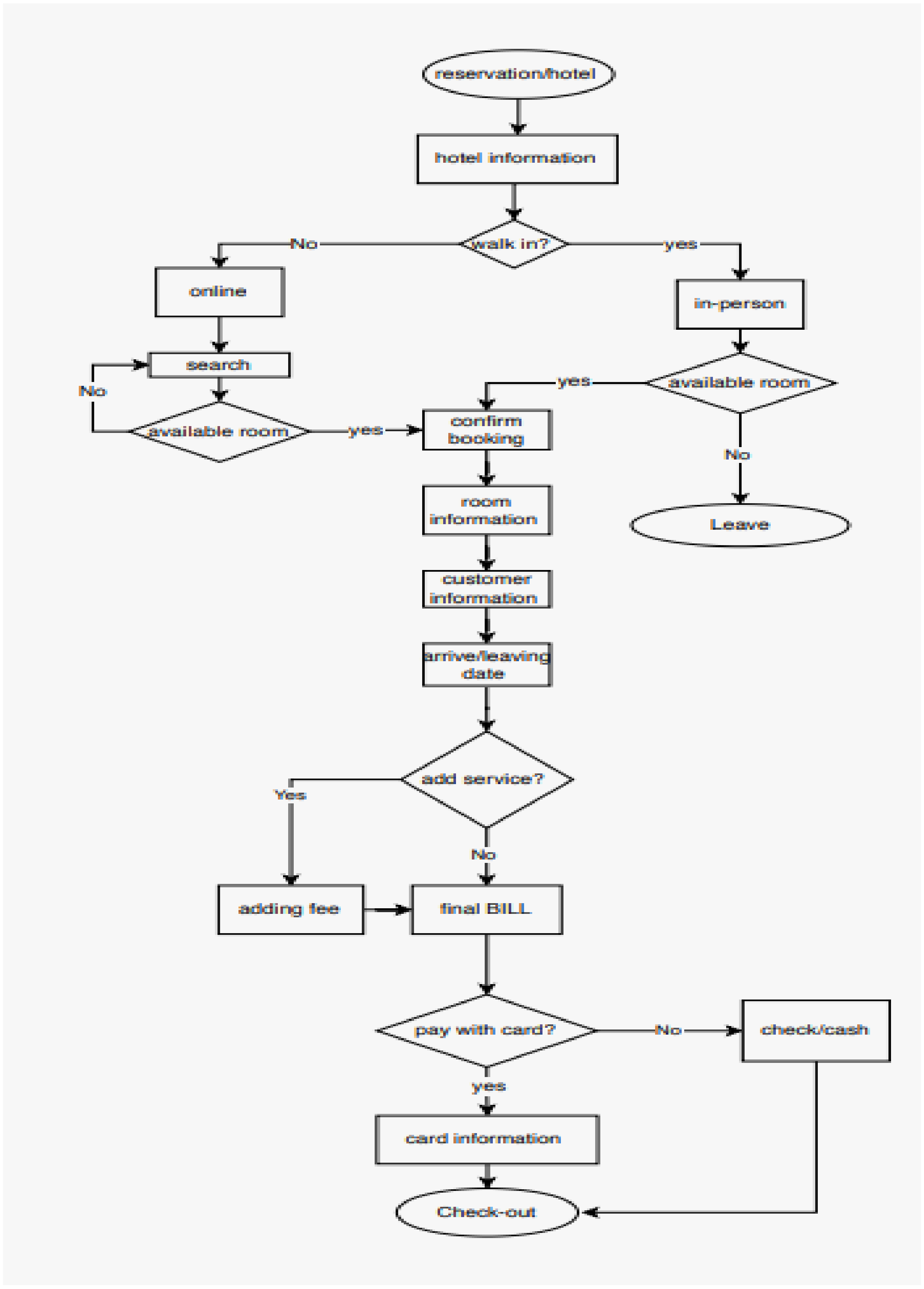
Constraints - We have until 11/29 to finish this project. Every Saturday, group members must finish their given tasks and wait for further instructions.

2.4 <<Platform Requirements Specification hardware/memory/Operating System requirements>>

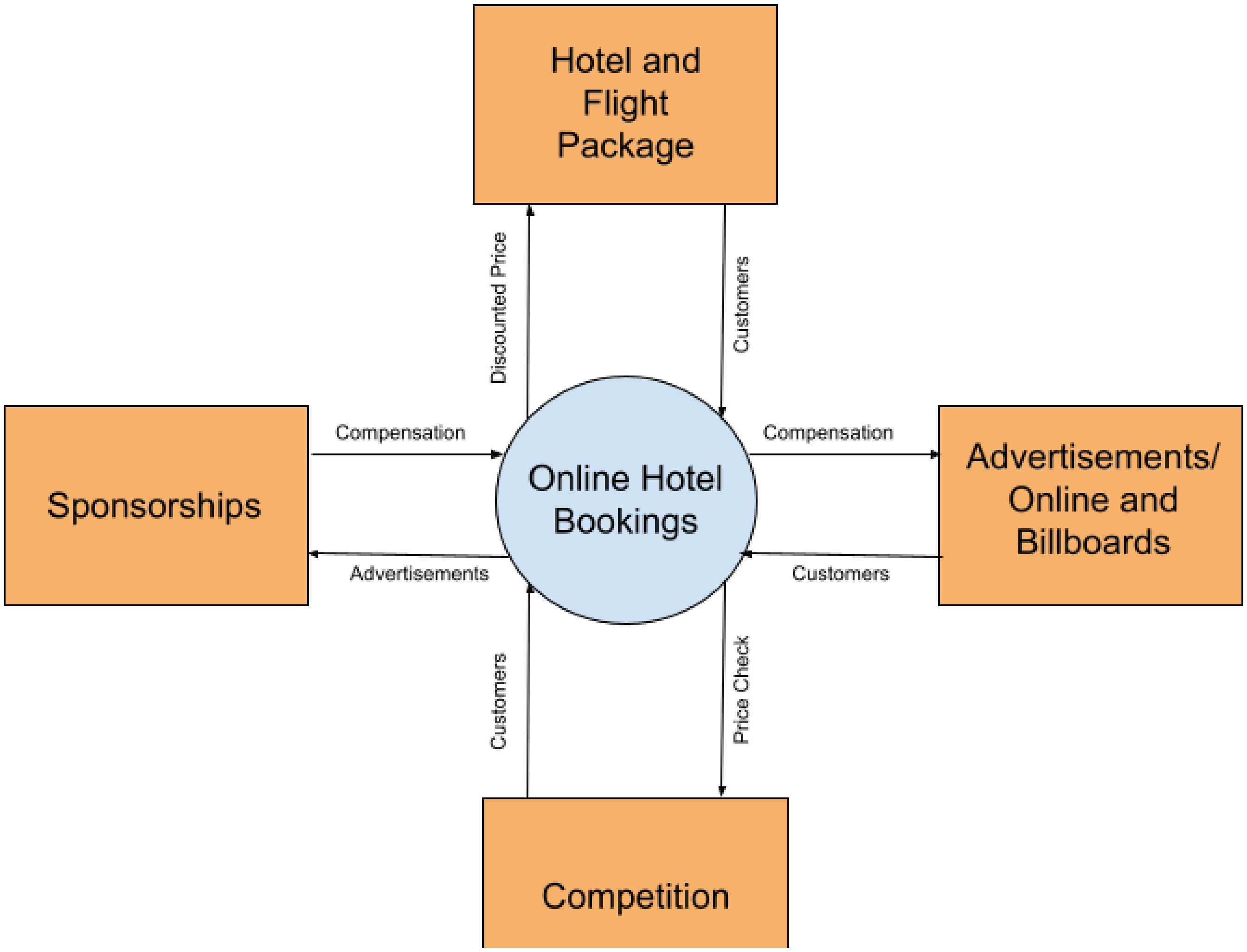
Customers need at least 1.5 GB of memory with a good cellular plan to secure a reservation of a hotel room. There are different methods for customers to book an online reservation such as the Chrome Browser on any device, as well as an application for both Apple users (Apple Store) and Android users (Google Play Store).

2.5 <<Context Diagram and the Process Flow Diagrams>>

Process Flow Diagram:



Context Diagram:



<<Section 3 - Codes and Implementations>>

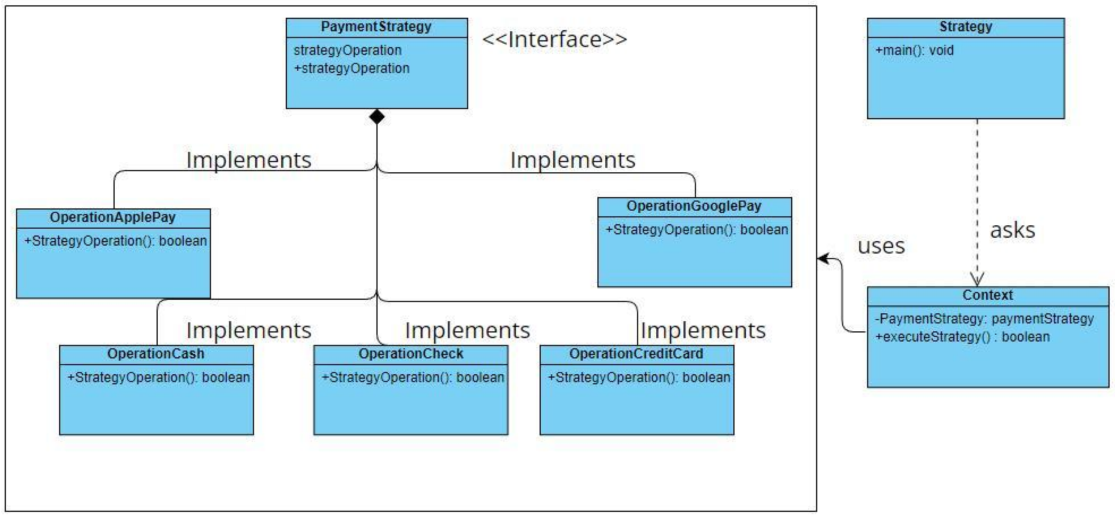
Criteria:

1) **Student Name with Design Pattern Name**
2) **UML Diagram for Design Patterns using your Classes**
3) **Design Pattern Code**
4) **Unit Test (TWO)**
5) **Component Test**

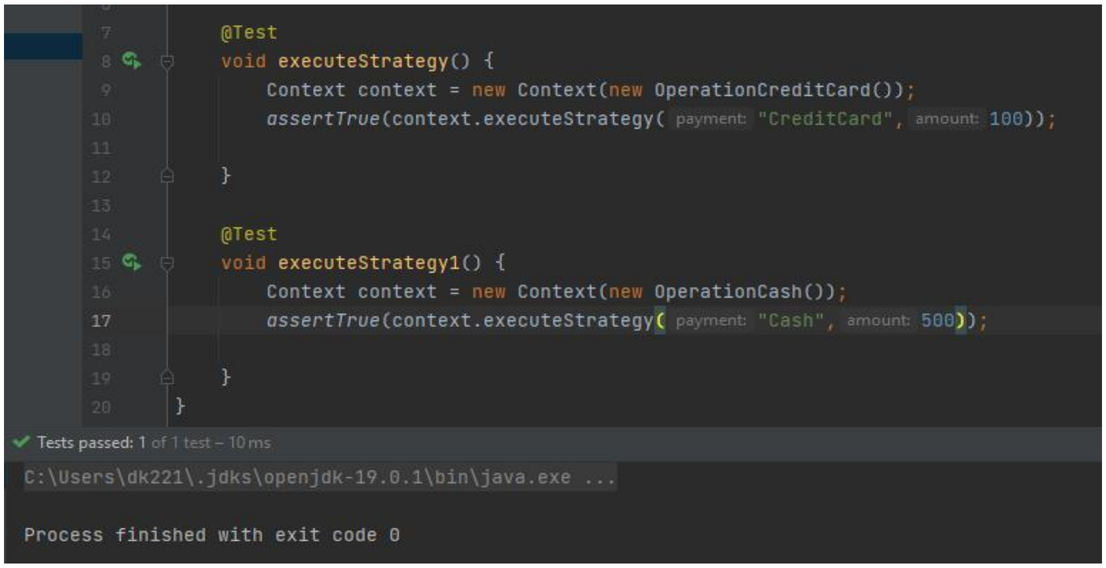
Ke Dong - Strategy Pattern

5 Classes: OperationCash, OperationCheck, OperationApplyPay, OperationGooglePay, and OperationCreditCar

UML Diagram:



Unit Tests:

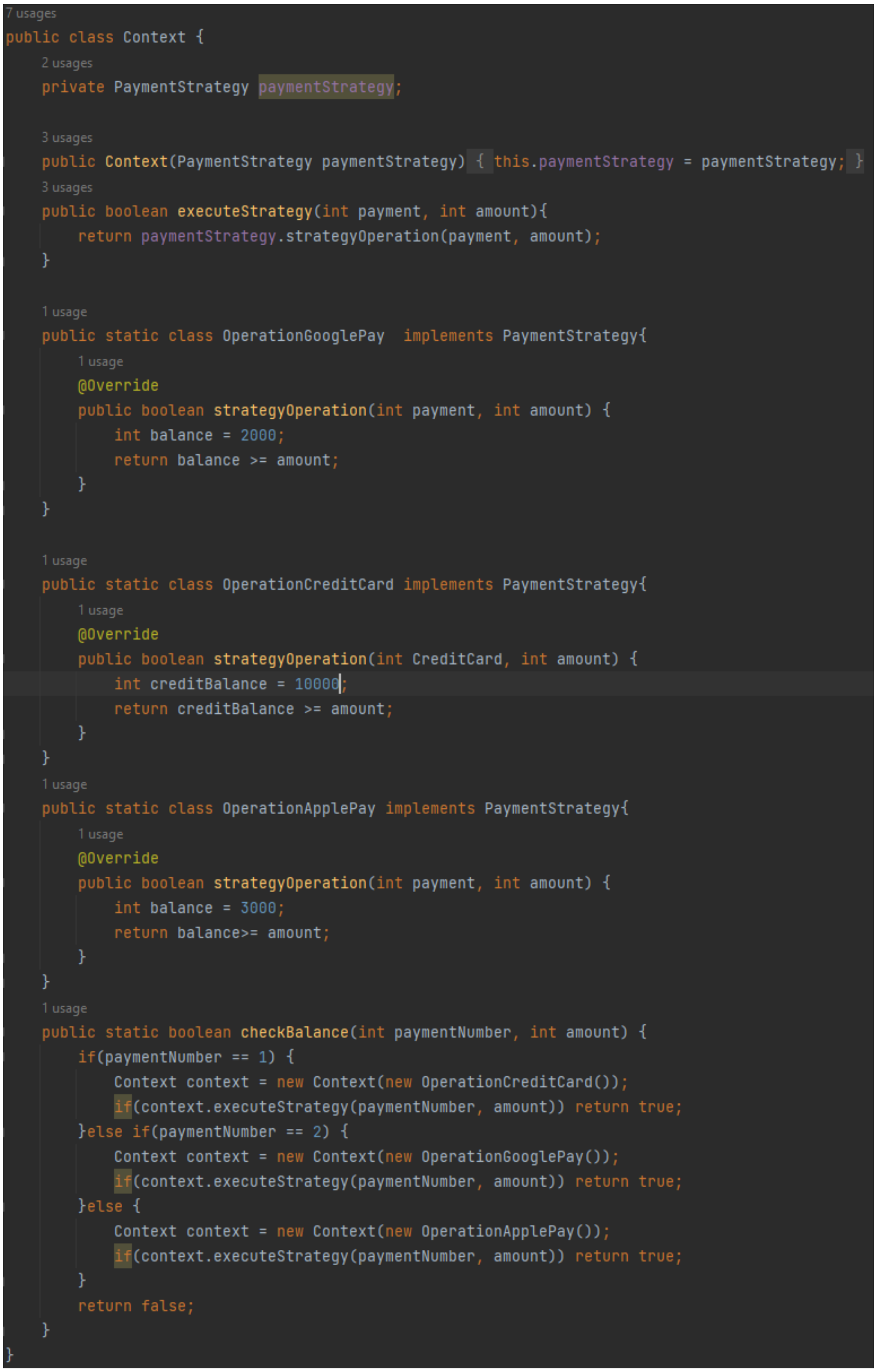


```
    @Test
    void executeStrategy() {
        Context context = new Context(new OperationCreditCard());
        assertTrue(context.executeStrategy( payment: "CreditCard", amount: 100));

    }

    @Test
    void executeStrategy1() {
        Context context = new Context(new OperationCash());
        assertTrue(context.executeStrategy( payment: "Cash", amount: 500));

    }
}
```

```
✓ Tests passed: 1 of 1 test – 10 ms
C:\Users\dk221\.jdks\openjdk-19.0.1\bin\java.exe ...

Process finished with exit code 0
```

Component test: If the customer pays the hotel room fee with Apple Pay. The fee is 560$. However, his applied pay balance is 400$. He will fail.

Strategy Pattern Code:

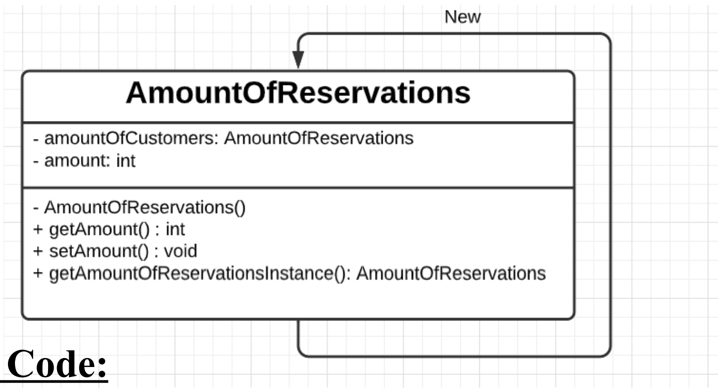


```
7 usages
public class Context {
    2 usages
    private PaymentStrategy paymentStrategy;

    3 usages
    public Context(PaymentStrategy paymentStrategy) { this.paymentStrategy = paymentStrategy; }
    3 usages
    public boolean executeStrategy(int payment, int amount){
        return paymentStrategy.strategyOperation(payment, amount);
    }

    1 usage
    public static class OperationGooglePay  implements PaymentStrategy{
        1 usage
        @Override
        public boolean strategyOperation(int payment, int amount) {
            int balance = 2000;
            return balance >= amount;
        }
    }

    1 usage
    public static class OperationCreditCard implements PaymentStrategy{
        1 usage
        @Override
        public boolean strategyOperation(int CreditCard, int amount) {
            int creditBalance = 10000;
            return creditBalance >= amount;
        }
    }
    1 usage
    public static class OperationApplePay implements PaymentStrategy{
        1 usage
        @Override
        public boolean strategyOperation(int payment, int amount) {
            int balance = 3000;
            return balance>= amount;
        }
    }
    1 usage
    public static boolean checkBalance(int paymentNumber, int amount) {
        if(paymentNumber == 1) {
            Context context = new Context(new OperationCreditCard());
            if(context.executeStrategy(paymentNumber, amount)) return true;
        }else if(paymentNumber == 2) {
            Context context = new Context(new OperationGooglePay());
            if(context.executeStrategy(paymentNumber, amount)) return true;
        }else {
            Context context = new Context(new OperationApplePay());
            if(context.executeStrategy(paymentNumber, amount)) return true;
        }
        return false;
    }
}
```

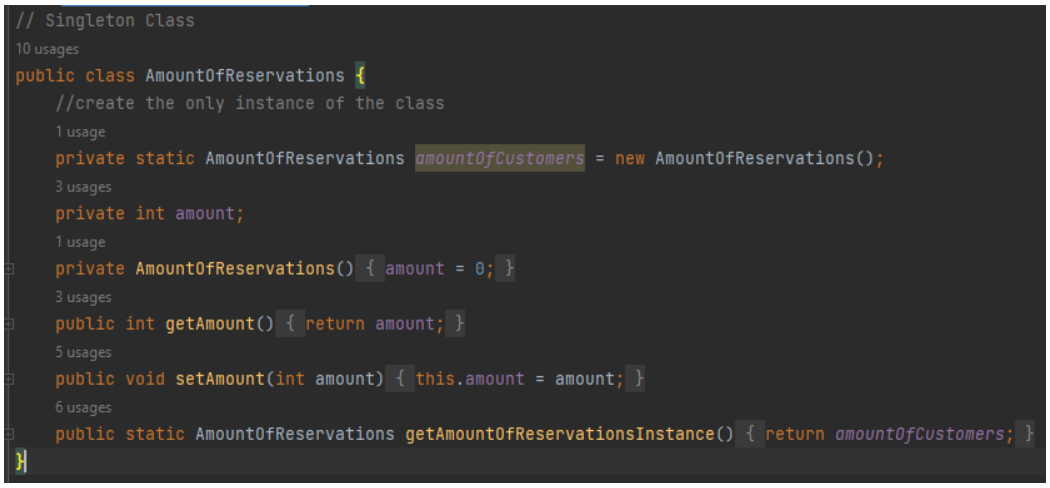
Samuel Mirakov - Singleton Pattern

1 Class: AmountOfReservations()

UML Diagram:

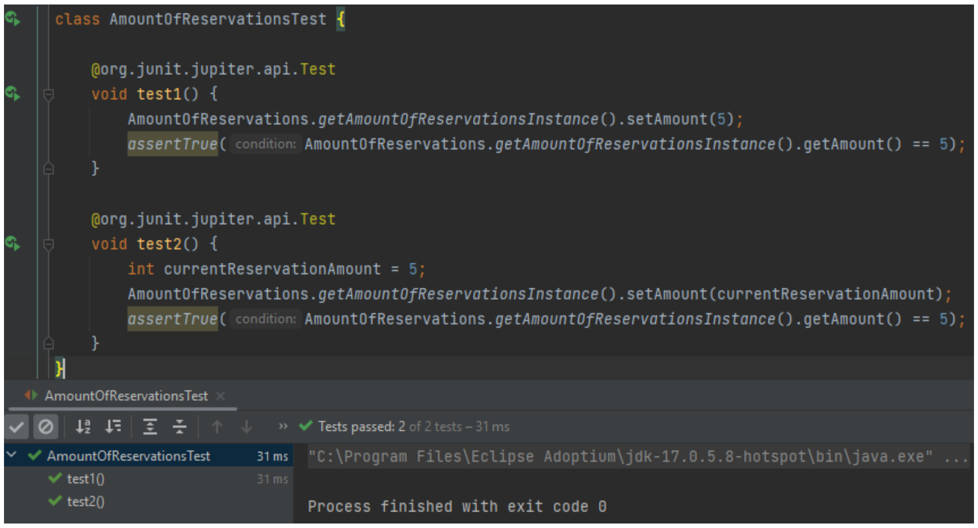


Singleton Pattern Code:



```
// Singleton Class
10 usages
public class AmountOfReservations {
    //create the only instance of the class
    1 usage
    private static AmountOfReservations amountOfCustomers = new AmountOfReservations();
    3 usages
    private int amount;
    1 usage
    private AmountOfReservations() { amount = 0; }
    3 usages
    public int getAmount() { return amount; }
    5 usages
    public void setAmount(int amount) { this.amount = amount; }
    6 usages
    public static AmountOfReservations getAmountOfReservationsInstance() { return amountOfCustomers; }
}
```

Unit Tests:



```
class AmountOfReservationsTest {

    @org.junit.jupiter.api.Test
    void test1() {
        AmountOfReservations.getAmountOfReservationsInstance().setAmount(5);
        assertTrue( condition: AmountOfReservations.getAmountOfReservationsInstance().getAmount() == 5);
    }

    @org.junit.jupiter.api.Test
    void test2() {
        int currentReservationAmount = 5;
        AmountOfReservations.getAmountOfReservationsInstance().setAmount(currentReservationAmount);
        assertTrue( condition: AmountOfReservations.getAmountOfReservationsInstance().getAmount() == 5);
    }
}
```

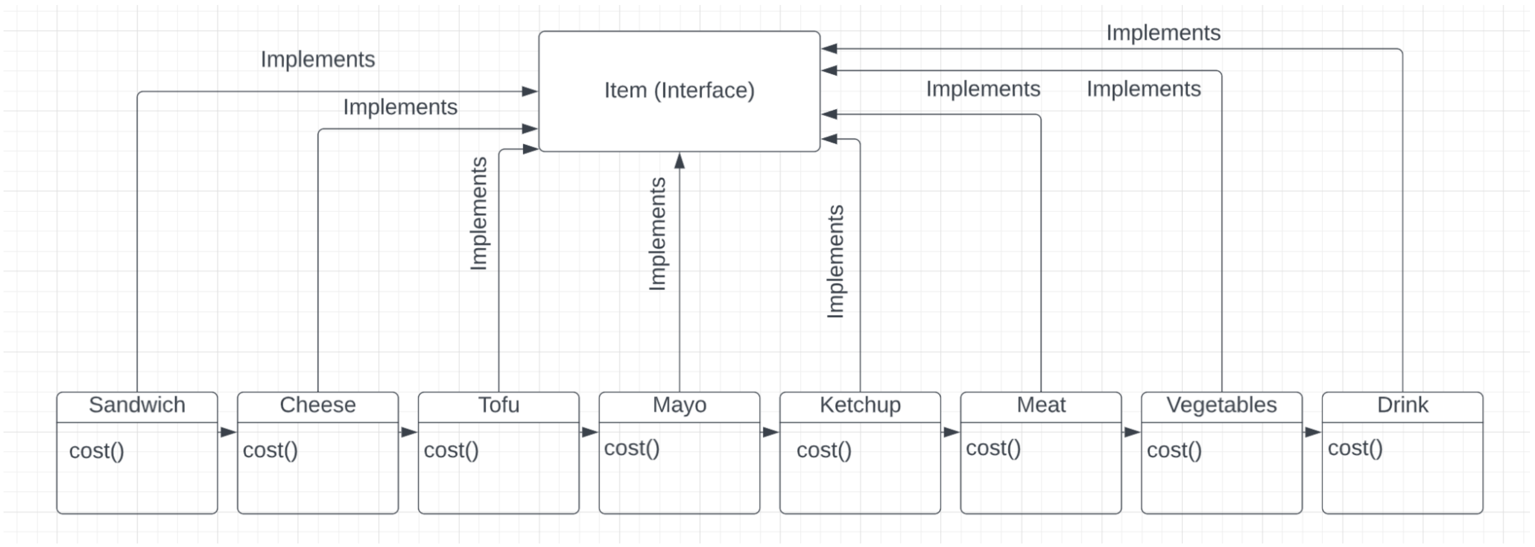
```
AmountOfReservationsTest
Tests passed: 2 of 2 tests – 31 ms
AmountOfReservationsTest  31 ms   "C:\Program Files\Eclipse Adoptium\jdk-17.0.5.8-hotspot\bin\java.exe" ...
  test1()                 31 ms
  test2()                         Process finished with exit code 0
```

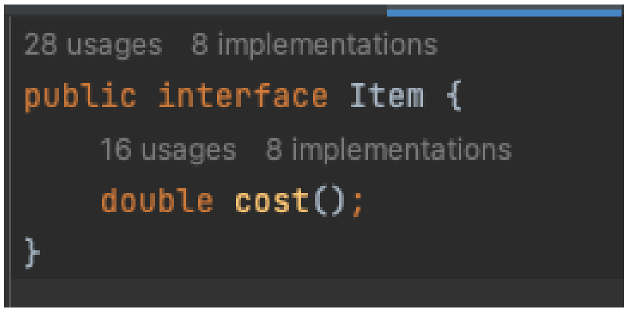
Component Test: A new customer checks in and creates a reservation with this software, it will return an updated number of current customers who have checked in using this software.

Mahfuz Uddin - Decorator Pattern

1 interface: Item
8 classes: Sandwich, Cheese, Drink, Ketchup, Mayo, Meat, Sandwich, Tofu, Vegetable

UML:





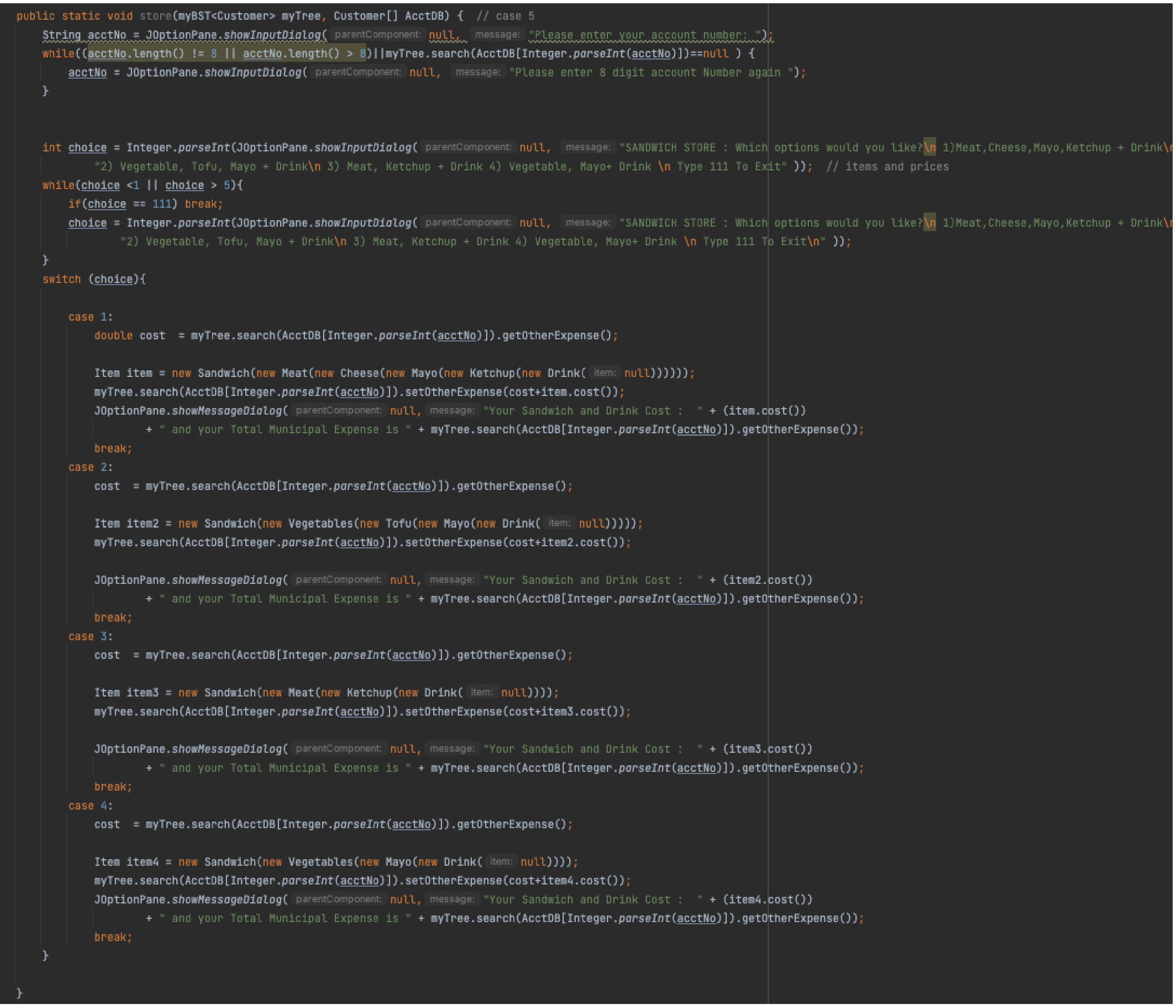
```
28 usages   8 implementations
public interface Item {
    16 usages   8 implementations
    double cost();
}
```

Decorator Pattern Code:

```
public class Sandwich implements Item{
   Item item;
   public Sandwich(Item item) {
      this.item = item;
   }

   public double cost(){
      double TotalCost =0;
      if(item!=null){
         TotalCost= item.cost();
      }
      return TotalCost + 1.00;
   }
}
class Meat implements Item{
   Item item;
   public Meat(Item item) {
      this.item = item;
   }

   public double cost(){
      double TotalCost =0;
      if(item!=null){
         TotalCost= item.cost();
      }
      return TotalCost + 3.00;
   }
}
class Vegetables implements Item{
   Item item;
   public Vegetables(Item item) {
      this.item = item;
   }
   public double cost(){
      double TotalCost =0;
      if(item!=null){
         TotalCost= item.cost();
      }
      return TotalCost + 2.00;
   }
}
class Mayo implements Item{
   Item item;
   public Mayo(Item item) {
      this.item = item;
   }
   public double cost(){
      double TotalCost =0;
      if(item!=null){
         TotalCost= item.cost();
      }
      return TotalCost + .50;
   }
}
```

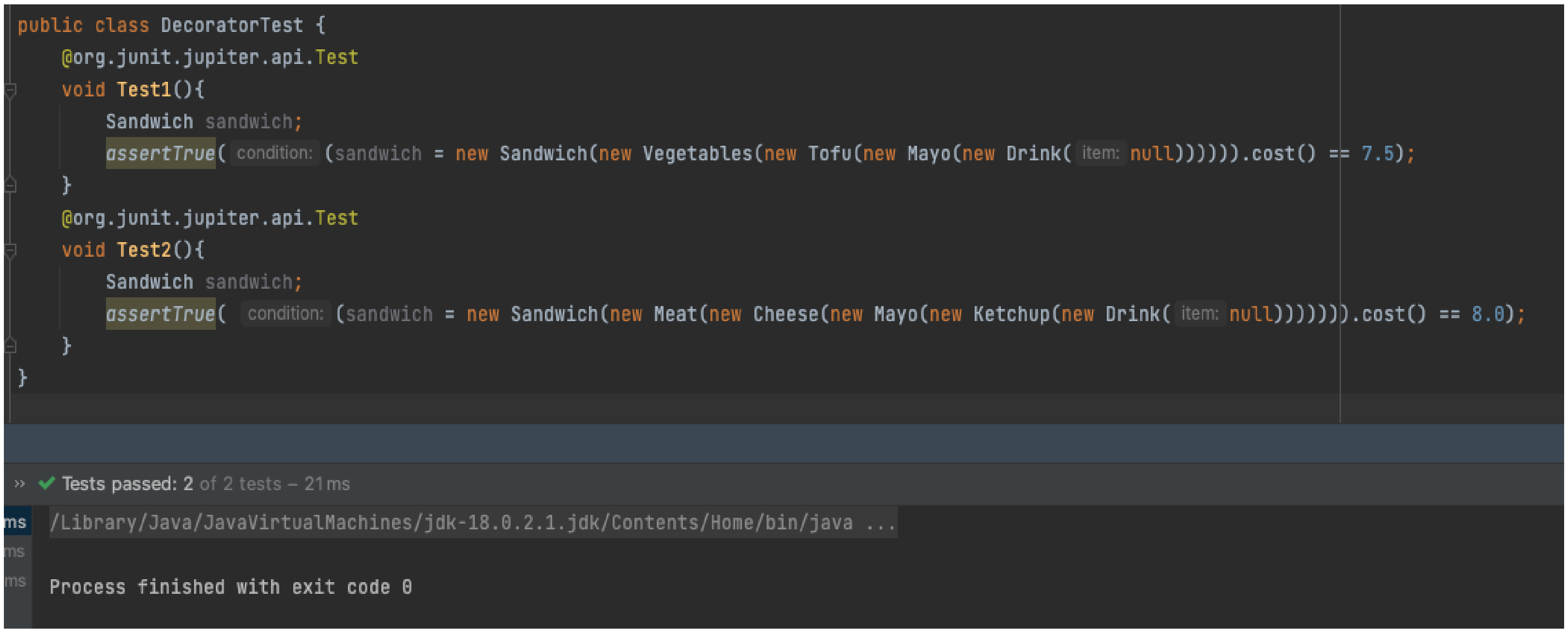
```
class Ketchup implements Item{
   Item item;
   public Ketchup(Item item) {
      this.item = item;
   }
   public double cost(){
      double TotalCost =0;
      if(item!=null){
         TotalCost= item.cost();
      }
      return TotalCost + .50;
   }
}
class Tofu implements Item{

   Item item;
   public Tofu(Item item) {
      this.item = item;
   }
   public double cost(){
      double TotalCost =0;
      if(item!=null){
         TotalCost= item.cost();
      }
      return TotalCost + 2.00;
   }
}
class Cheese implements Item{
   Item item;
   public Cheese(Item item) {
      this.item = item;
   }

   public double cost(){
      double TotalCost =0;
      if(item!=null){
         TotalCost= item.cost();
      }
      return TotalCost + 1.00;
   }
}
class Drink implements Item{
   Item item;

   public Drink(Item item) {
      this.item = item;
   }
   public double cost(){
      double TotalCost =0;
      if(item!=null){
         TotalCost= item.cost();
      }
      return TotalCost + 2.00;
   }
```

Where the pattern is used:



```java
public static void store(myBST<Customer> myTree, Customer[] AcctDB) {  // case 5
    String acctNo = JOptionPane.showInputDialog( parentComponent: null,  message: "Please enter your account number: ");
    while((acctNo.length() != 8 || acctNo.length() > 8)||myTree.search(AcctDB[Integer.parseInt(acctNo)])==null ) {
        acctNo = JOptionPane.showInputDialog( parentComponent: null,  message: "Please enter 8 digit account Number again ");
    }


    int choice = Integer.parseInt(JOptionPane.showInputDialog( parentComponent: null,  message: "SANDWICH STORE : Which options would you like?\n 1)Meat,Cheese,Mayo,Ketchup + Drink\
            "2) Vegetable, Tofu, Mayo + Drink\n 3) Meat, Ketchup + Drink 4) Vegetable, Mayo+ Drink \n Type 111 To Exit" ));  // items and prices
    while(choice <1 || choice > 5){
        if(choice == 111) break;
        choice = Integer.parseInt(JOptionPane.showInputDialog( parentComponent: null,  message: "SANDWICH STORE : Which options would you like?\n 1)Meat,Cheese,Mayo,Ketchup + Drink\
                "2) Vegetable, Tofu, Mayo + Drink\n 3) Meat, Ketchup + Drink 4) Vegetable, Mayo+ Drink \n Type 111 To Exit\n" ));
    }
    switch (choice){

        case 1:
            double cost  = myTree.search(AcctDB[Integer.parseInt(acctNo)]).getOtherExpense();

            Item item = new Sandwich(new Meat(new Cheese(new Mayo(new Ketchup(new Drink( item: null))))));
            myTree.search(AcctDB[Integer.parseInt(acctNo)]).setOtherExpense(cost+item.cost());
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Your Sandwich and Drink Cost :  " + (item.cost())
                    + " and your Total Municipal Expense is " + myTree.search(AcctDB[Integer.parseInt(acctNo)]).getOtherExpense());
            break;
        case 2:
            cost  = myTree.search(AcctDB[Integer.parseInt(acctNo)]).getOtherExpense();

            Item item2 = new Sandwich(new Vegetables(new Tofu(new Mayo(new Drink( item: null)))));
            myTree.search(AcctDB[Integer.parseInt(acctNo)]).setOtherExpense(cost+item2.cost());

            JOptionPane.showMessageDialog( parentComponent: null,  message: "Your Sandwich and Drink Cost :  " + (item2.cost())
                    + " and your Total Municipal Expense is " + myTree.search(AcctDB[Integer.parseInt(acctNo)]).getOtherExpense());
            break;
        case 3:
            cost  = myTree.search(AcctDB[Integer.parseInt(acctNo)]).getOtherExpense();

            Item item3 = new Sandwich(new Meat(new Ketchup(new Drink( item: null))));
            myTree.search(AcctDB[Integer.parseInt(acctNo)]).setOtherExpense(cost+item3.cost());

            JOptionPane.showMessageDialog( parentComponent: null,  message: "Your Sandwich and Drink Cost :  " + (item3.cost())
                    + " and your Total Municipal Expense is " + myTree.search(AcctDB[Integer.parseInt(acctNo)]).getOtherExpense());
            break;
        case 4:
            cost  = myTree.search(AcctDB[Integer.parseInt(acctNo)]).getOtherExpense();

            Item item4 = new Sandwich(new Vegetables(new Mayo(new Drink( item: null))));
            myTree.search(AcctDB[Integer.parseInt(acctNo)]).setOtherExpense(cost+item4.cost());
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Your Sandwich and Drink Cost :  " + (item4.cost())
                    + " and your Total Municipal Expense is " + myTree.search(AcctDB[Integer.parseInt(acctNo)]).getOtherExpense());
            break;
    }

}
```

UNIT TEST:



```
public class DecoratorTest {
    @org.junit.jupiter.api.Test
    void Test1(){
        Sandwich sandwich;
        assertTrue( condition: (sandwich = new Sandwich(new Vegetables(new Tofu(new Mayo(new Drink( item: null)))))).cost() == 7.5);
    }
    @org.junit.jupiter.api.Test
    void Test2(){
        Sandwich sandwich;
        assertTrue( condition: (sandwich = new Sandwich(new Meat(new Cheese(new Mayo(new Ketchup(new Drink( item: null))))))).cost() == 8.0);
    }
}
```

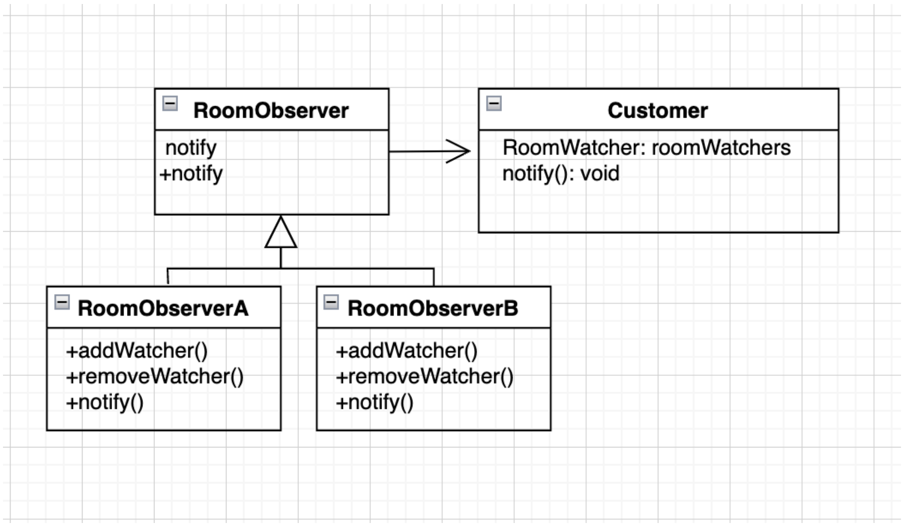
```
» ✓ Tests passed: 2 of 2 tests – 21 ms
ms  /Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...
ms
ms  Process finished with exit code 0
```

COMPONENT TEST: A customer buys one sandwich first, then buys another one a few hours later. Their "Other Expense" record should be updated with the sum of both sandwiches.

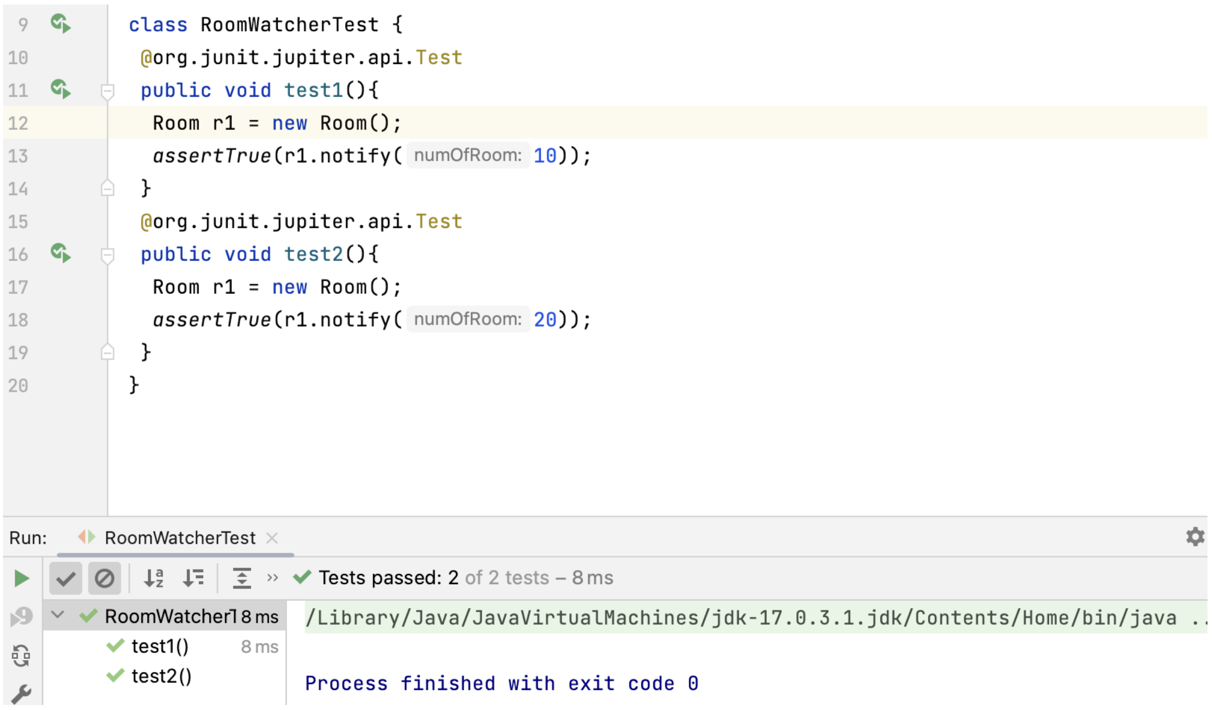
Xiaoqian He - Observer Pattern

3 Functions: addRoomWatcher(), removeRoomWatcher() and notify()

UML:



Unit Test:



```
class RoomWatcherTest {
 @org.junit.jupiter.api.Test
 public void test1(){
  Room r1 = new Room();
  assertTrue(r1.notify( numOfRoom: 10));
 }
 @org.junit.jupiter.api.Test
 public void test2(){
  Room r1 = new Room();
  assertTrue(r1.notify( numOfRoom: 20));
 }
}
```

```
Run:  RoomWatcherTest
Tests passed: 2 of 2 tests – 8 ms
RoomWatcherT 8 ms
  test1()   8 ms
  test2()
/Library/Java/JavaVirtualMachines/jdk-17.0.3.1.jdk/Contents/Home/bin/java ..

Process finished with exit code 0
```

Component test: If the room available changed, then it will notify how many rooms are still available.

Observer Pattern Code:

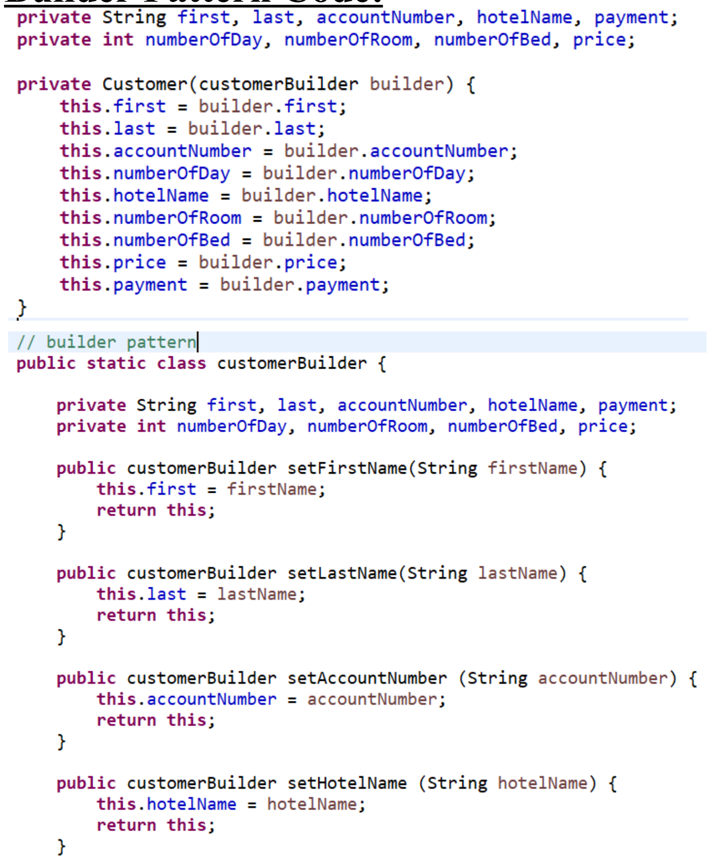
Customer and Room classes implement Observer design pattern

```
// RoomWatcher interface
public interface RoomWatcher {

  public default void notify (int numOfRoom){
  }

}

// Customer Class
public class Customer implements Comparable<Customer>, RoomWatcher{

@Override

public void notify(int roomNum) {

  System.out.print("\nThe number of empty rooms now is " + roomNum + "\n");

         }

}

//Room class
import java.util.ArrayList;
import java.util.List;

public class Room{

         private List<RoomWatcher> roomWatcher = new ArrayList<>();

         public void addWatcher(RoomWatcher cw){

         roomWatcher.add(cw);

         }
         public void removeWatch(RoomWatcher cw){

         roomWatcher.remove(cw);
         }
         public boolean notify(int numOfRoom){

         for(RoomWatcher cw: roomWatcher)

         cw.notify(numOfRoom);
         return true;
         }
}
```

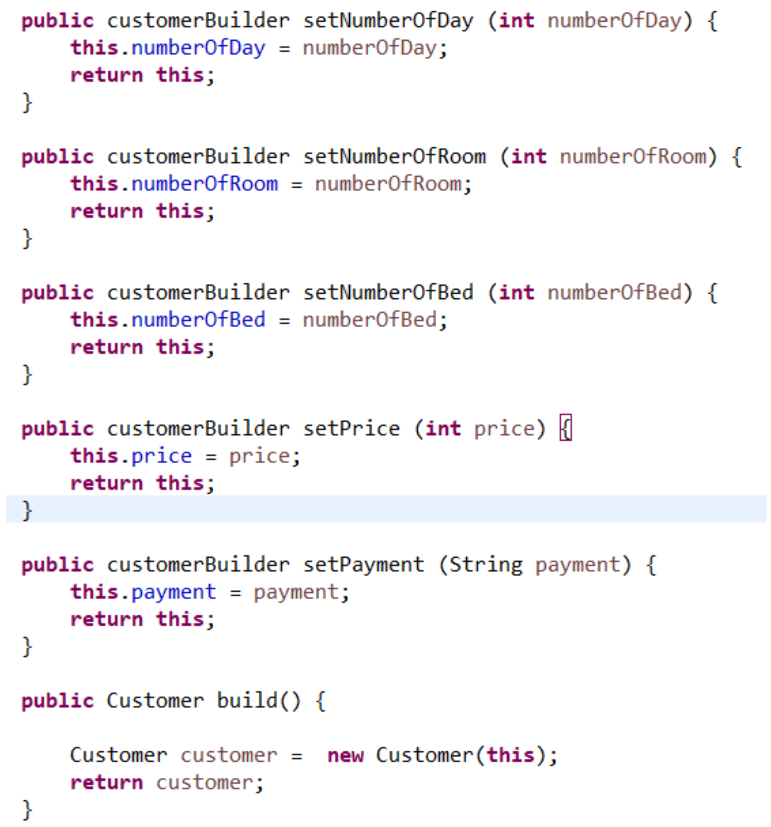
Redwanul Haque - Builder Pattern

2 Classes: customer(), customerBuilder()

Builder Pattern Code:

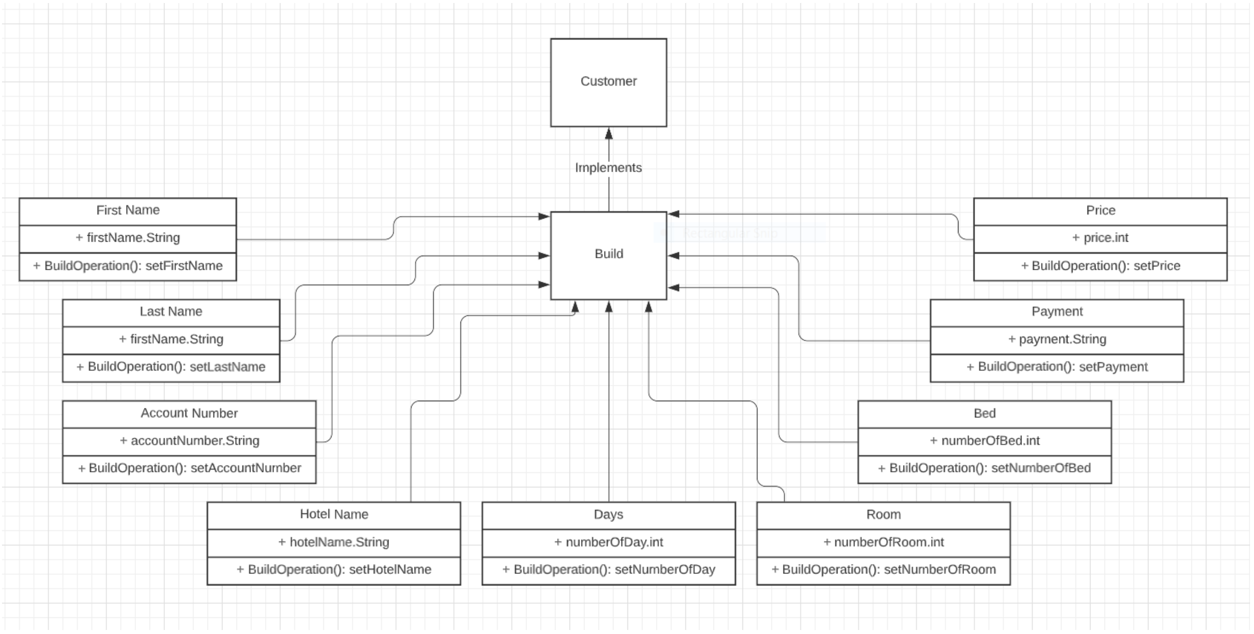


```java
private String first, last, accountNumber, hotelName, payment;
private int numberOfDay, numberOfRoom, numberOfBed, price;

private Customer(customerBuilder builder) {
    this.first = builder.first;
    this.last = builder.last;
    this.accountNumber = builder.accountNumber;
    this.numberOfDay = builder.numberOfDay;
    this.hotelName = builder.hotelName;
    this.numberOfRoom = builder.numberOfRoom;
    this.numberOfBed = builder.numberOfBed;
    this.price = builder.price;
    this.payment = builder.payment;
}

// builder pattern
public static class customerBuilder {

    private String first, last, accountNumber, hotelName, payment;
    private int numberOfDay, numberOfRoom, numberOfBed, price;

    public customerBuilder setFirstName(String firstName) {
        this.first = firstName;
        return this;
    }

    public customerBuilder setLastName(String lastName) {
        this.last = lastName;
        return this;
    }

    public customerBuilder setAccountNumber (String accountNumber) {
        this.accountNumber = accountNumber;
        return this;
    }

    public customerBuilder setHotelName (String hotelName) {
        this.hotelName = hotelName;
        return this;
    }
```

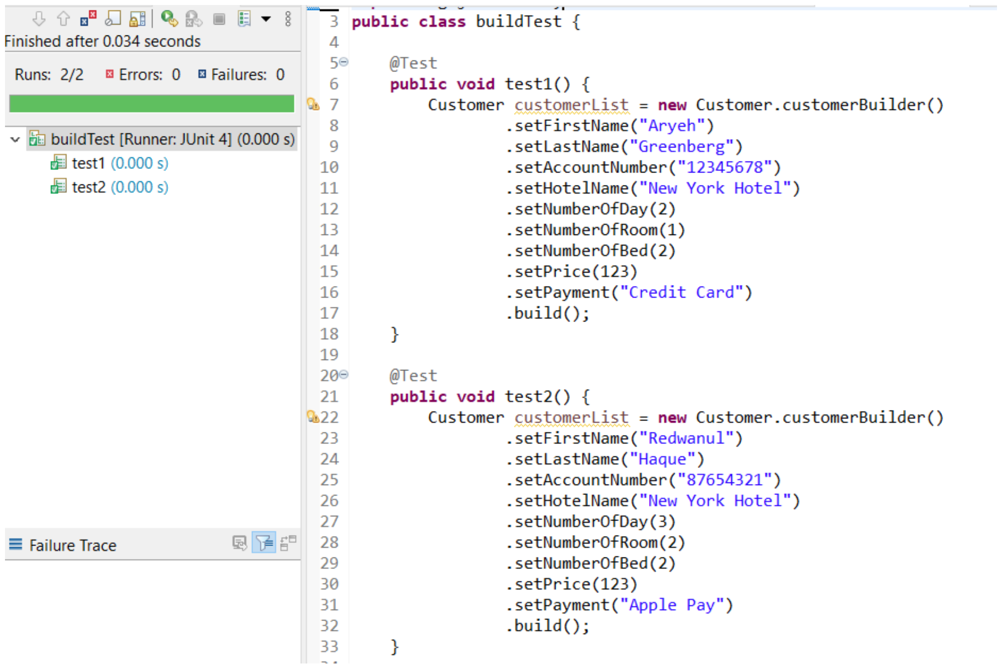


```java
public customerBuilder setNumberOfDay (int numberOfDay) {
    this.numberOfDay = numberOfDay;
    return this;
}

public customerBuilder setNumberOfRoom (int numberOfRoom) {
    this.numberOfRoom = numberOfRoom;
    return this;
}

public customerBuilder setNumberOfBed (int numberOfBed) {
    this.numberOfBed = numberOfBed;
    return this;
}

public customerBuilder setPrice (int price) {
    this.price = price;
    return this;
}

public customerBuilder setPayment (String payment) {
    this.payment = payment;
    return this;
}

public Customer build() {

    Customer customer =  new Customer(this);
    return customer;
}
```

UML:


Customer
Implements
Build
First Name
+ firstName.String
+ BuildOperation(): setFirstName
Last Name
+ firstName.String
+ BuildOperation(): setLastName
Account Number
+ accountNumber.String
+ BuildOperation(): setAccountNumber
Hotel Name
+ hotelName.String
+ BuildOperation(): setHotelName
Days
+ numberOfDay.int
+ BuildOperation(): setNumberOfDay
Room
+ numberOfRoom.int
+ BuildOperation(): setNumberOfRoom
Price
+ price.int
+ BuildOperation(): setPrice
Payment
+ payment.String
+ BuildOperation(): setPayment
Bed
+ numberOfBed.int
+ BuildOperation(): setNumberOfBed

Unit Test:



```
Finished after 0.034 seconds

Runs: 2/2    Errors: 0    Failures: 0

buildTest [Runner: JUnit 4] (0.000 s)
    test1 (0.000 s)
    test2 (0.000 s)

Failure Trace
```

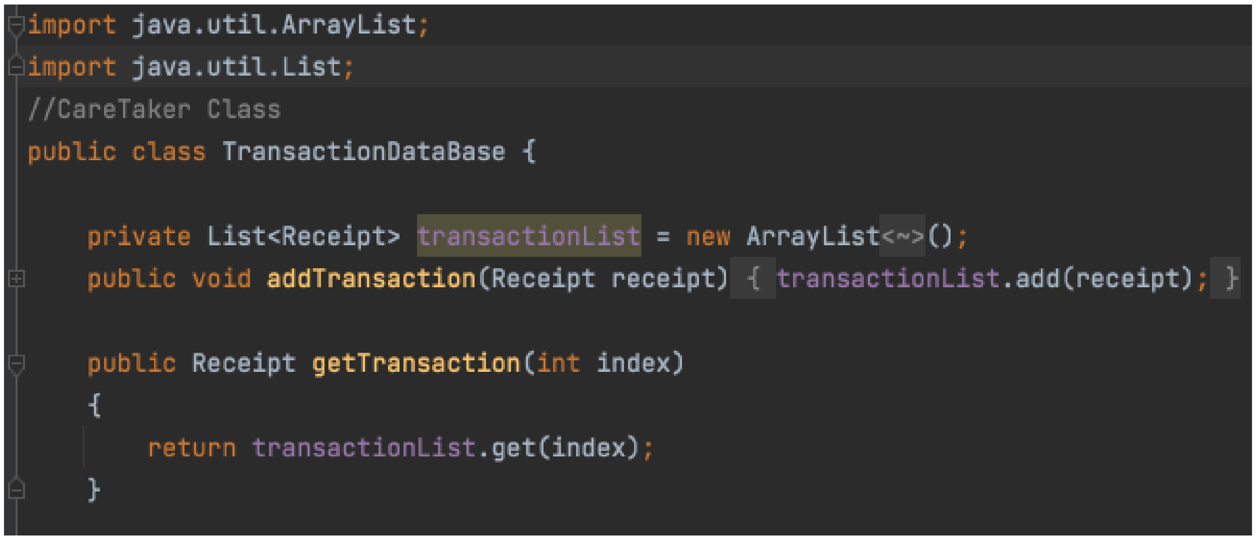
```java
public class buildTest {

    @Test
    public void test1() {
        Customer customerList = new Customer.customerBuilder()
                .setFirstName("Aryeh")
                .setLastName("Greenberg")
                .setAccountNumber("12345678")
                .setHotelName("New York Hotel")
                .setNumberOfDay(2)
                .setNumberOfRoom(1)
                .setNumberOfBed(2)
                .setPrice(123)
                .setPayment("Credit Card")
                .build();
    }

    @Test
    public void test2() {
        Customer customerList = new Customer.customerBuilder()
                .setFirstName("Redwanul")
                .setLastName("Haque")
                .setAccountNumber("87654321")
                .setHotelName("New York Hotel")
                .setNumberOfDay(3)
                .setNumberOfRoom(2)
                .setNumberOfBed(2)
                .setPrice(123)
                .setPayment("Apple Pay")
                .build();
    }
```

Component Test: The CustomerBuilder will read all the information from the data.txt and also will keep track if a customer creates a reservation, delete a reservation, or edit a reservation and update the data.txt.

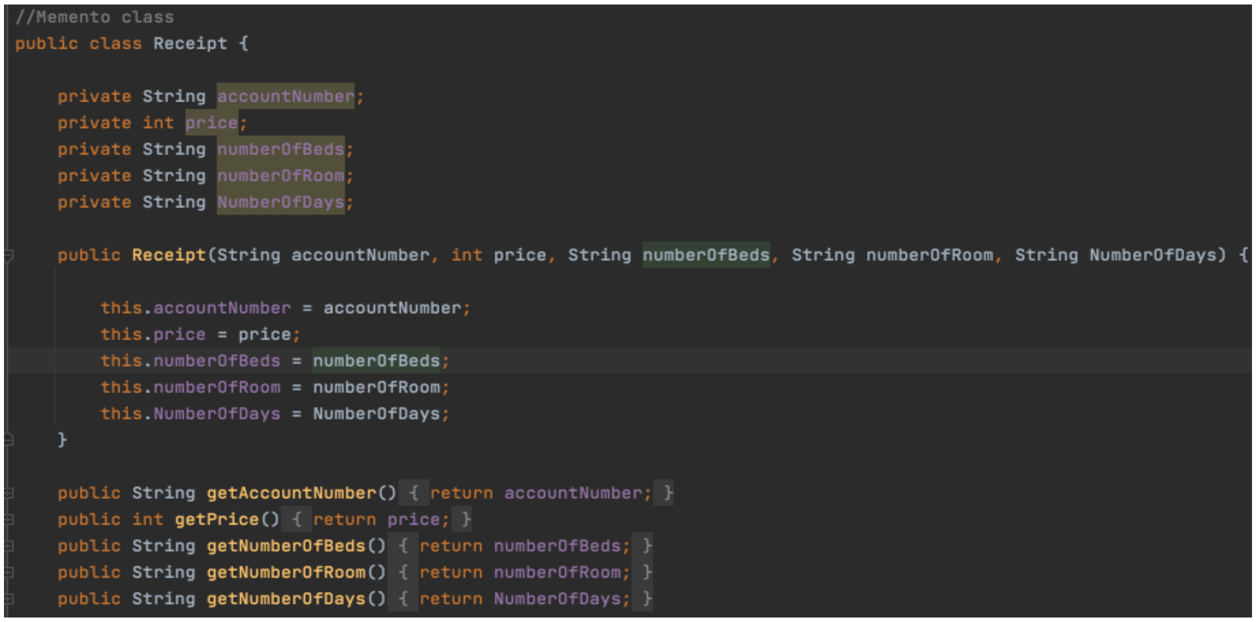
Uneeb Siddiqui - Memento Pattern

3 classes: **TransactionDatabase, Receipt, and BillGenerator**

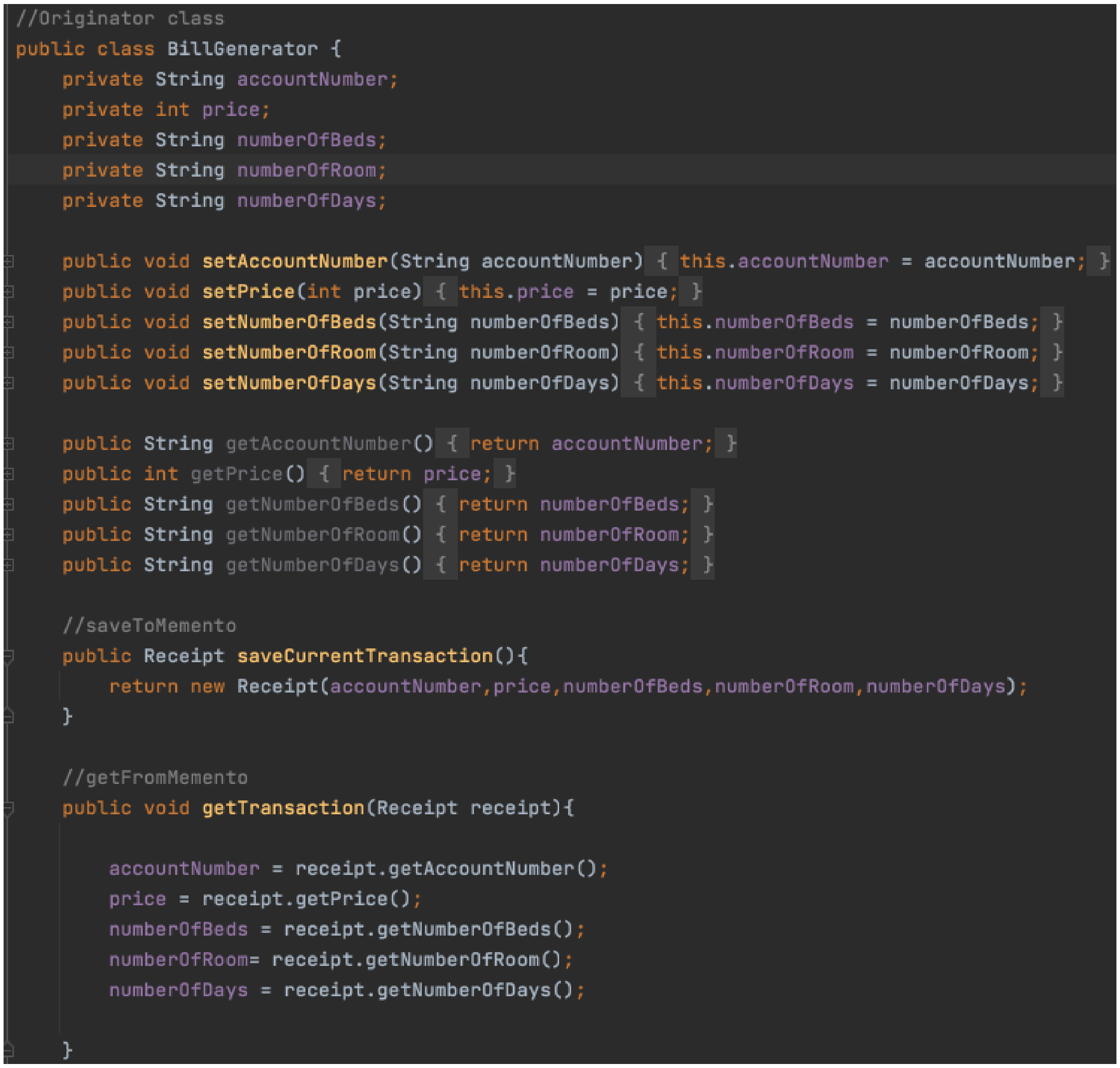
Memento Pattern Code:



```java
import java.util.ArrayList;
import java.util.List;
//CareTaker Class
public class TransactionDataBase {

    private List<Receipt> transactionList = new ArrayList<~>();
    public void addTransaction(Receipt receipt) { transactionList.add(receipt); }

    public Receipt getTransaction(int index)
    {
        return transactionList.get(index);
    }
```

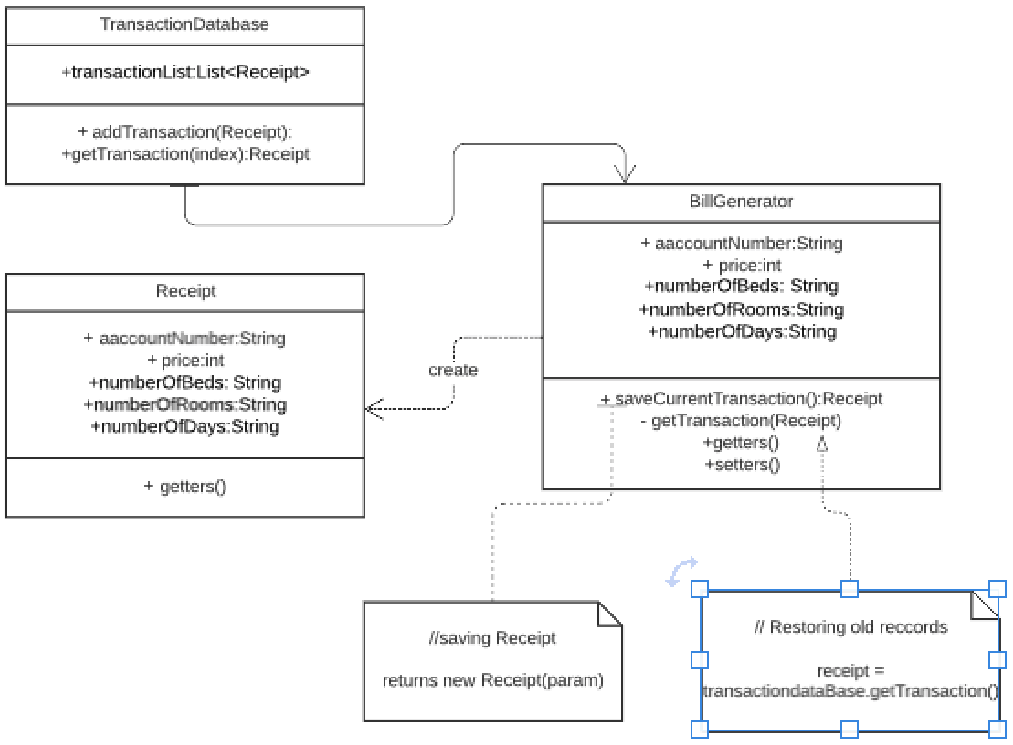


```java
//Memento class
public class Receipt {

    private String accountNumber;
    private int price;
    private String numberOfBeds;
    private String numberOfRoom;
    private String NumberOfDays;

    public Receipt(String accountNumber, int price, String numberOfBeds, String numberOfRoom, String NumberOfDays) {

        this.accountNumber = accountNumber;
        this.price = price;
        this.numberOfBeds = numberOfBeds;
        this.numberOfRoom = numberOfRoom;
        this.NumberOfDays = NumberOfDays;
    }

    public String getAccountNumber() { return accountNumber; }
    public int getPrice() { return price; }
    public String getNumberOfBeds() { return numberOfBeds; }
    public String getNumberOfRoom() { return numberOfRoom; }
    public String getNumberOfDays() { return NumberOfDays; }
```

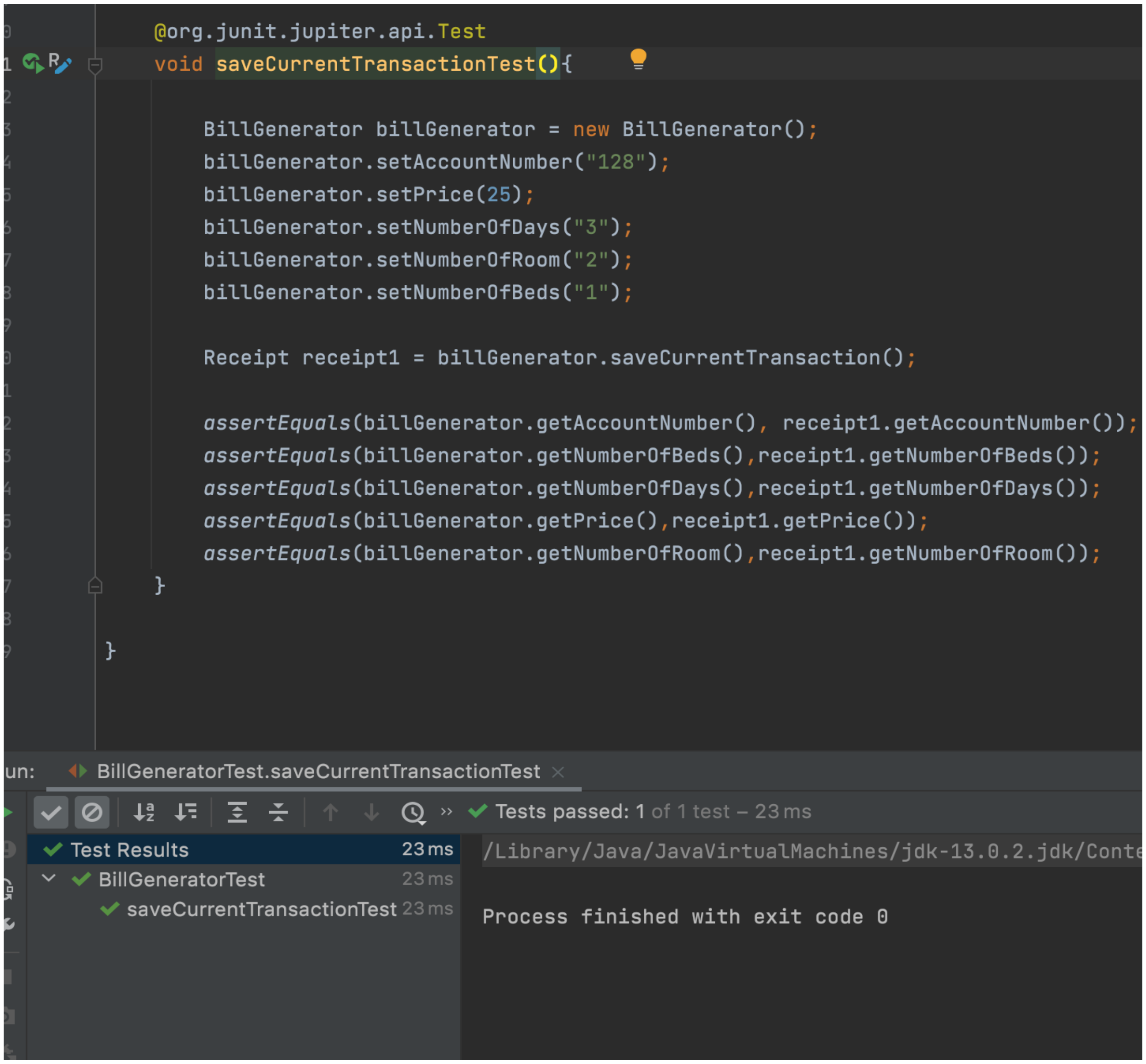


```java
//Originator class
public class BillGenerator {
    private String accountNumber;
    private int price;
    private String numberOfBeds;
    private String numberOfRoom;
    private String numberOfDays;

    public void setAccountNumber(String accountNumber) { this.accountNumber = accountNumber; }
    public void setPrice(int price) { this.price = price; }
    public void setNumberOfBeds(String numberOfBeds) { this.numberOfBeds = numberOfBeds; }
    public void setNumberOfRoom(String numberOfRoom) { this.numberOfRoom = numberOfRoom; }
    public void setNumberOfDays(String numberOfDays) { this.numberOfDays = numberOfDays; }

    public String getAccountNumber() { return accountNumber; }
    public int getPrice() { return price; }
    public String getNumberOfBeds() { return numberOfBeds; }
    public String getNumberOfRoom() { return numberOfRoom; }
    public String getNumberOfDays() { return numberOfDays; }

    //saveToMemento
    public Receipt saveCurrentTransaction(){
        return new Receipt(accountNumber,price,numberOfBeds,numberOfRoom,numberOfDays);
    }

    //getFromMemento
    public void getTransaction(Receipt receipt){

        accountNumber = receipt.getAccountNumber();
        price = receipt.getPrice();
        numberOfBeds = receipt.getNumberOfBeds();
        numberOfRoom= receipt.getNumberOfRoom();
        numberOfDays = receipt.getNumberOfDays();

    }
```

UML:

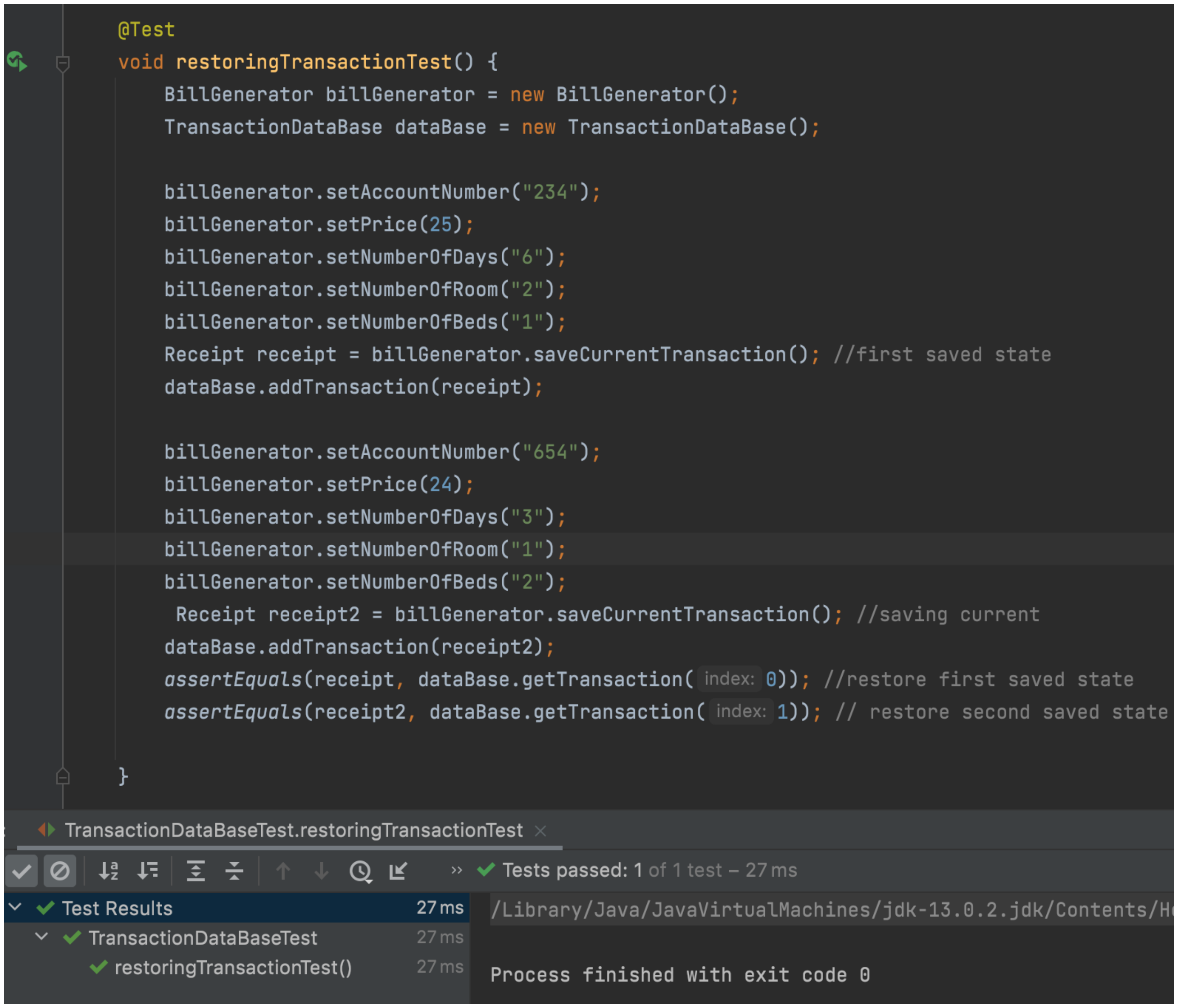

TransactionDatabase
+transactionList:List<Receipt>
+ addTransaction(Receipt):
+getTransaction(index):Receipt
BillGenerator
+ aaccountNumber:String
+ price:int
+numberOfBeds: String
+numberOfRooms:String
+numberOfDays:String
+ saveCurrentTransaction():Receipt
- getTransaction(Receipt)
+getters()
+setters()
Receipt
+ aaccountNumber:String
+ price:int
+numberOfBeds: String
+numberOfRooms:String
+numberOfDays:String
+ getters()
create
//saving Receipt
returns new Receipt(param)
// Restoring old reccords
receipt =
transactiondataBase.getTransaction()

Unit Test:



```
@org.junit.jupiter.api.Test
void saveCurrentTransactionTest(){

    BillGenerator billGenerator = new BillGenerator();
    billGenerator.setAccountNumber("128");
    billGenerator.setPrice(25);
    billGenerator.setNumberOfDays("3");
    billGenerator.setNumberOfRoom("2");
    billGenerator.setNumberOfBeds("1");

    Receipt receipt1 = billGenerator.saveCurrentTransaction();

    assertEquals(billGenerator.getAccountNumber(), receipt1.getAccountNumber());
    assertEquals(billGenerator.getNumberOfBeds(),receipt1.getNumberOfBeds());
    assertEquals(billGenerator.getNumberOfDays(),receipt1.getNumberOfDays());
    assertEquals(billGenerator.getPrice(),receipt1.getPrice());
    assertEquals(billGenerator.getNumberOfRoom(),receipt1.getNumberOfRoom());
}

}
```

```
un:  BillGeneratorTest.saveCurrentTransactionTest

Tests passed: 1 of 1 test – 23 ms

Test Results                       23 ms
  BillGeneratorTest                23 ms
    saveCurrentTransactionTest     23 ms

/Library/Java/JavaVirtualMachines/jdk-13.0.2.jdk/Conte

Process finished with exit code 0
```



```
@Test
void restoringTransactionTest() {
    BillGenerator billGenerator = new BillGenerator();
    TransactionDataBase dataBase = new TransactionDataBase();

    billGenerator.setAccountNumber("234");
    billGenerator.setPrice(25);
    billGenerator.setNumberOfDays("6");
    billGenerator.setNumberOfRoom("2");
    billGenerator.setNumberOfBeds("1");
    Receipt receipt = billGenerator.saveCurrentTransaction(); //first saved state
    dataBase.addTransaction(receipt);

    billGenerator.setAccountNumber("654");
    billGenerator.setPrice(24);
    billGenerator.setNumberOfDays("3");
    billGenerator.setNumberOfRoom("1");
    billGenerator.setNumberOfBeds("2");
     Receipt receipt2 = billGenerator.saveCurrentTransaction(); //saving current
    dataBase.addTransaction(receipt2);
    assertEquals(receipt, dataBase.getTransaction( index: 0)); //restore first saved state
    assertEquals(receipt2, dataBase.getTransaction( index: 1)); // restore second saved state

}
```

TransactionDataBaseTest.restoringTransactionTest

Tests passed: 1 of 1 test – 27 ms

```
Test Results                          27 ms
  TransactionDataBaseTest             27 ms
    restoringTransactionTest()        27 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-13.0.2.jdk/Contents/H

Process finished with exit code 0
```

Component Test: A customer with a current hotel reservation wants an overview of their reservation. After entering their personal account number, an overview of their reservation details will be presented to them.