

# Project

---

HOTEL RESERVATION SYSTEM

# Introduction

---

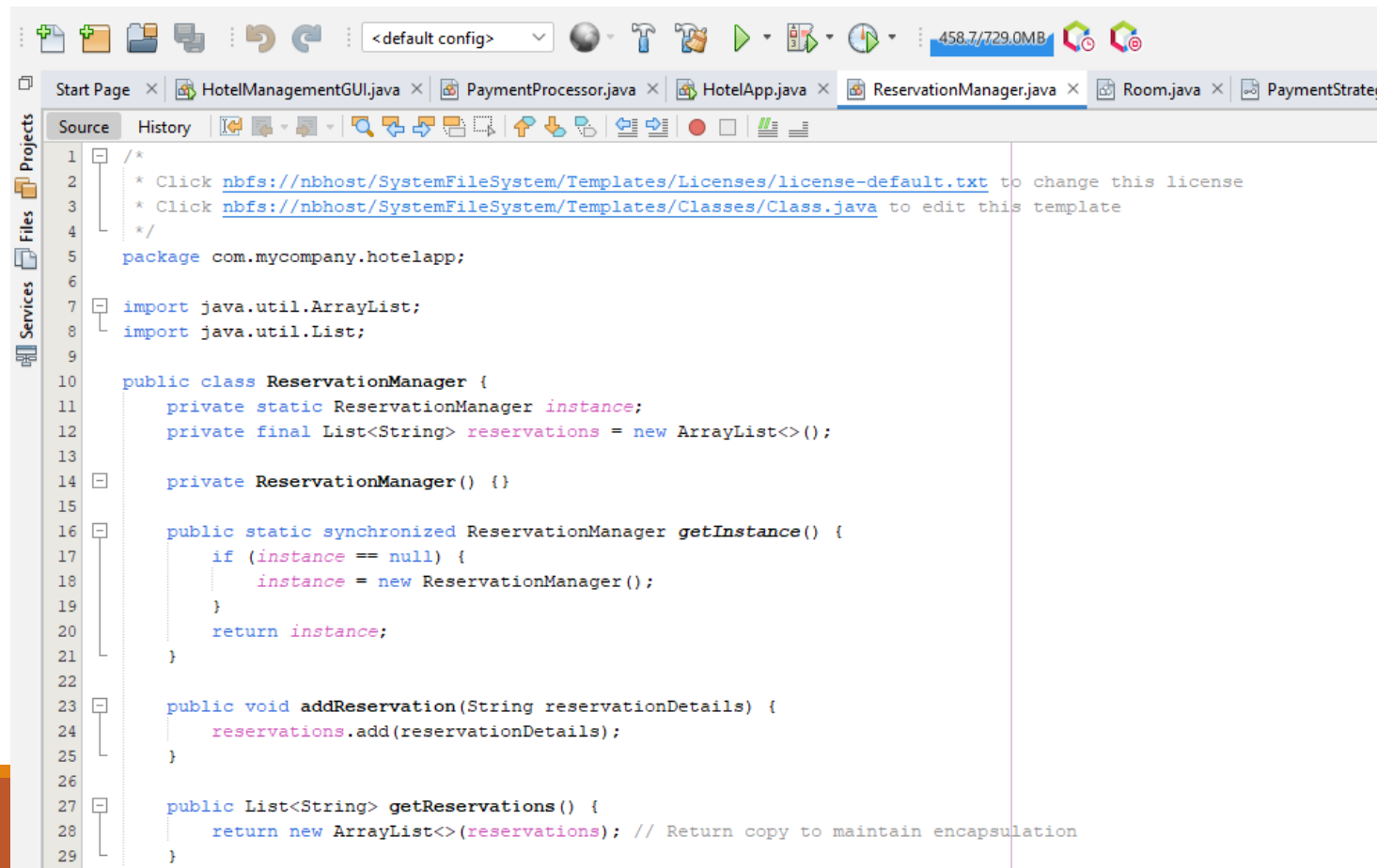
The **Hotel Reservation System** The project is a Java GUI application that allows users to book hotel rooms or make payments, with the ability to choose extras such as breakfast for the rooms. The project is implemented using several design patterns to write clean and extensible code.

# Summary

Discrepyion	Design pattern	Class
الملف الرئيسي الذي يحتوي على الواجهة الرسومية ويربط جميع المكونات.	Main	<b>HotelApp.java</b>
يُدير جميع بيانات الحجوزات.	Singleton	<b>ReservationManager</b>
يُدير عمليات الدفع.	Singleton	<b>PaymentProcessor</b>
يُنشئ كائنات الغرف Standard/Deluxe	Factory	RoomFactory
يُضيف ميزات إضافية للغرفة مثل الإفطار.	Decorator	RoomDecorator
يُدير إستراتيجيات الدفع المختلفة Credit/Cash	Strategy	<b>PaymentStrategy</b>
يعرض التحديثات (الحجوزات/الدفع) فور إدخالها للمستخدم.	Observer	<b>JTextArea in HotelApp</b>

# 1. Singleton Pattern : Purpose: Ensures that there is only one copy of the base class that runs the main functions.

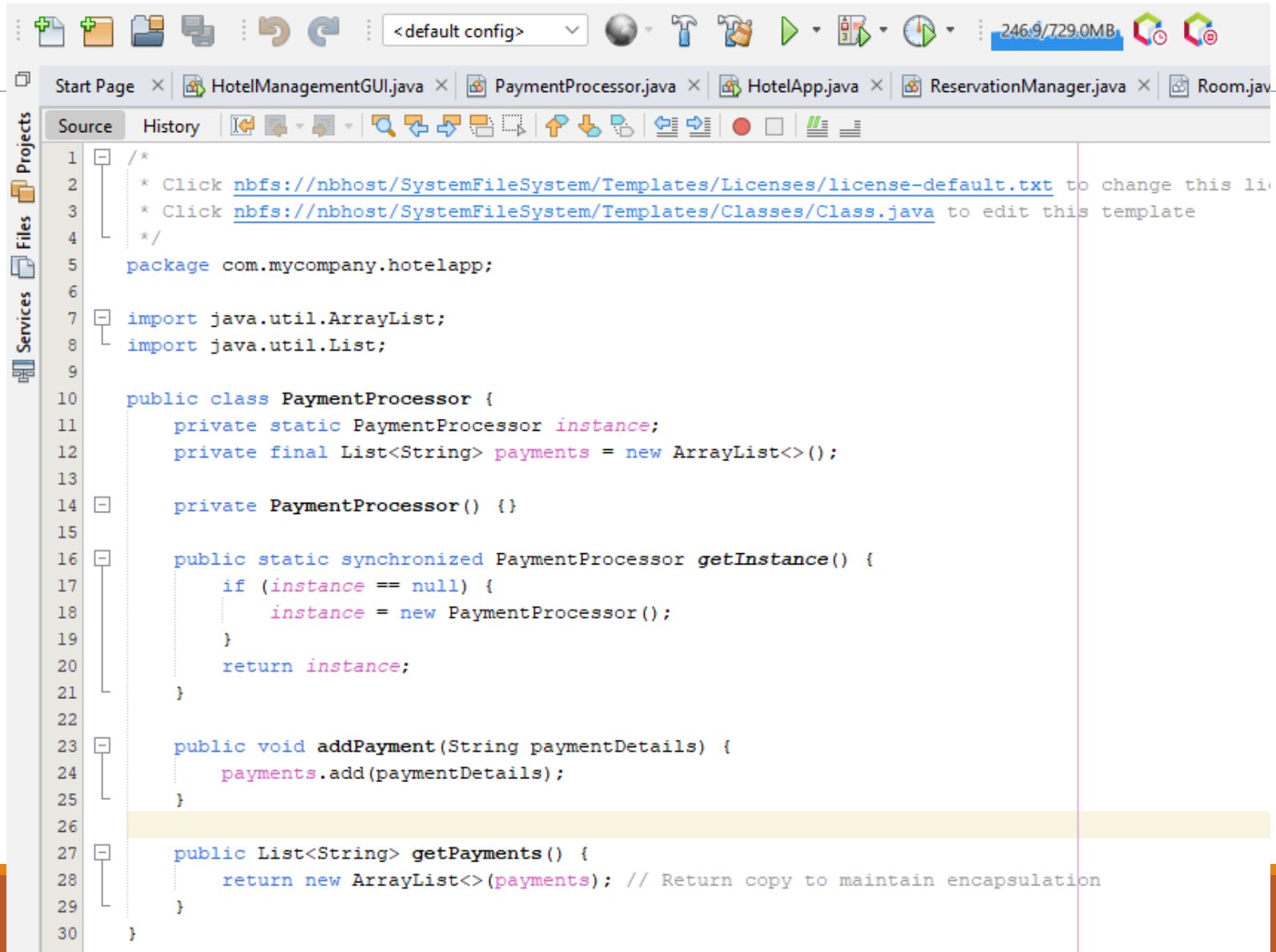
ReservationManager :Manages all reservations data. The getInstance method ensures that there is only one instance of this class in the application.



The screenshot shows an IDE window with the file `ReservationManager.java` open. The code implements the Singleton Pattern for the `ReservationManager` class. The package is `com.mycompany.hotellapp`. It imports `java.util.ArrayList` and `java.util.List`. The class has a private static `ReservationManager` instance and a private final `List<String>` named `reservations` initialized with a new `ArrayList<>()`. The `getInstance()` method is public static and synchronized, returning the instance if it's not null, or creating a new instance if it is. The `addReservation(String reservationDetails)` method adds details to the `reservations` list. The `getReservations()` method returns a new `ArrayList<>` containing the `reservations` list to maintain encapsulation.

```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5  package com.mycompany.hotellapp;
6
7  import java.util.ArrayList;
8  import java.util.List;
9
10 public class ReservationManager {
11     private static ReservationManager instance;
12     private final List<String> reservations = new ArrayList<>();
13
14     private ReservationManager() {}
15
16     public static synchronized ReservationManager getInstance() {
17         if (instance == null) {
18             instance = new ReservationManager();
19         }
20         return instance;
21     }
22
23     public void addReservation(String reservationDetails) {
24         reservations.add(reservationDetails);
25     }
26
27     public List<String> getReservations() {
28         return new ArrayList<>(reservations); // Return copy to maintain encapsulation
29     }
30 }
```

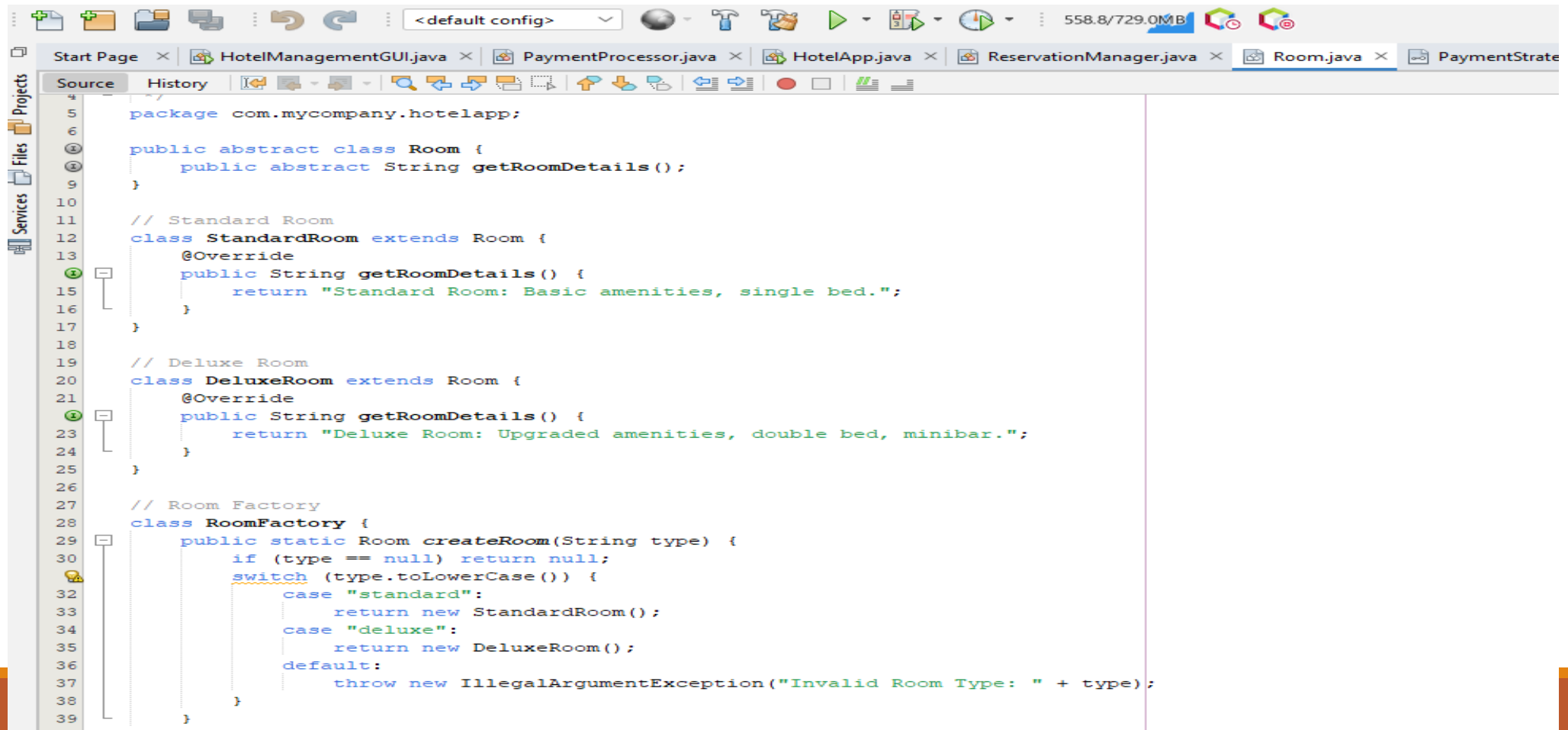
PaymentProcessor : Manages payment processes. Like ReservationManager, Singleton is used to get a single copy.



```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this li
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5   package com.mycompany.hotelapp;
6
7   import java.util.ArrayList;
8   import java.util.List;
9
10  public class PaymentProcessor {
11      private static PaymentProcessor instance;
12      private final List<String> payments = new ArrayList<>();
13
14      private PaymentProcessor() {}
15
16      public static synchronized PaymentProcessor getInstance() {
17          if (instance == null) {
18              instance = new PaymentProcessor();
19          }
20          return instance;
21      }
22
23      public void addPayment(String paymentDetails) {
24          payments.add(paymentDetails);
25      }
26
27      public List<String> getPayments() {
28          return new ArrayList<>(payments); // Return copy to maintain encapsulation
29      }
30  }
```

## 2. Factory Pattern : Purpose: Facilitates the creation of dynamic objects based on user input.

RoomFactory : Creates different types of rooms: StandardRoom and DeluxeRoom.

A screenshot of an IDE window showing the implementation of the Factory Method pattern in Java. The window has multiple tabs at the top: Start Page, HotelManagementGUI.java, PaymentProcessor.java, HotelApp.java, ReservationManager.java, Room.java (selected), and PaymentState.java. The left sidebar shows a project tree with 'Projects', 'Files', and 'Services' views. The main editor area displays the source code for Room.java. The code defines an abstract class 'Room' with an abstract method 'getRoomDetails()'. It then defines two concrete classes, 'StandardRoom' and 'DeluxeRoom', both extending 'Room' and overriding 'getRoomDetails()'. Finally, it defines a 'RoomFactory' class with a static method 'createRoom(String type)' that uses a switch statement to instantiate either a 'StandardRoom' or a 'DeluxeRoom' based on the input string. The code is as follows:

```
package com.mycompany.hotelapp;

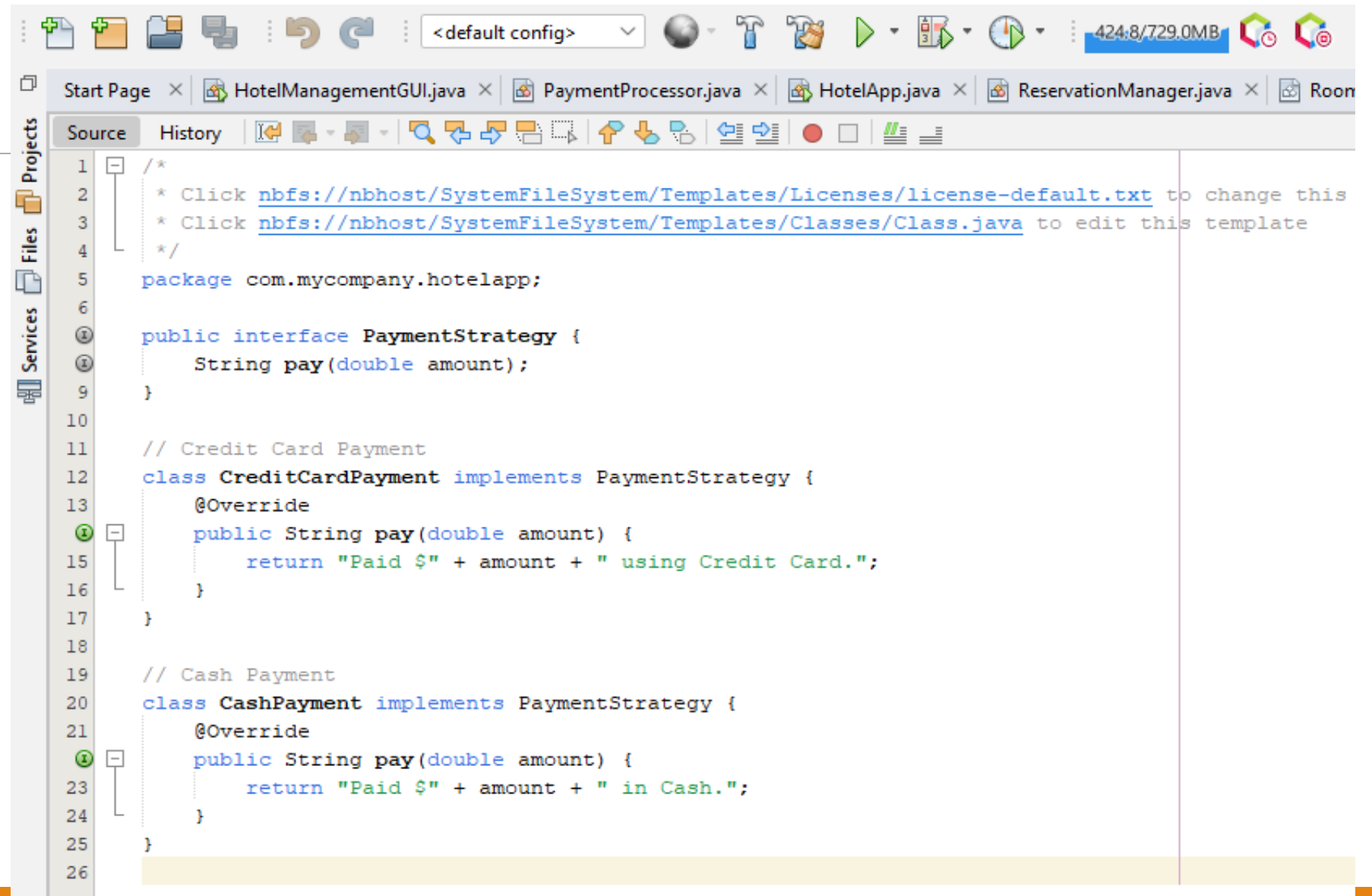
public abstract class Room {
    public abstract String getRoomDetails();
}

// Standard Room
class StandardRoom extends Room {
    @Override
    public String getRoomDetails() {
        return "Standard Room: Basic amenities, single bed.";
    }
}

// Deluxe Room
class DeluxeRoom extends Room {
    @Override
    public String getRoomDetails() {
        return "Deluxe Room: Upgraded amenities, double bed, minibar.";
    }
}

// Room Factory
class RoomFactory {
    public static Room createRoom(String type) {
        if (type == null) return null;
        switch (type.toLowerCase()) {
            case "standard":
                return new StandardRoom();
            case "deluxe":
                return new DeluxeRoom();
            default:
                throw new IllegalArgumentException("Invalid Room Type: " + type);
        }
    }
}
```

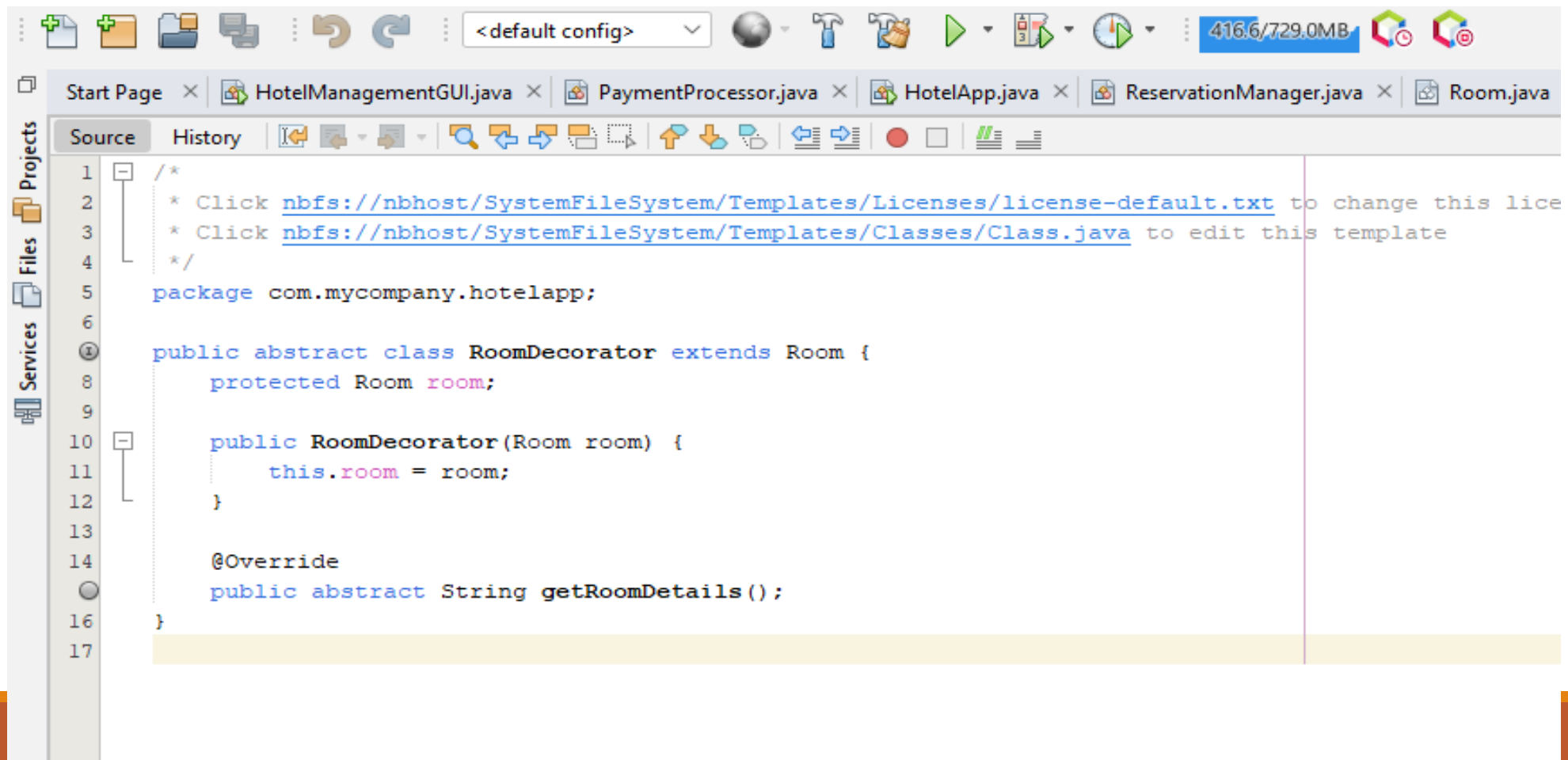
PaymentStrategy : (implicitly, in specifying the payment strategy).



```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5  package com.mycompany.hotelapp;
6
7  public interface PaymentStrategy {
8      String pay(double amount);
9  }
10
11  // Credit Card Payment
12  class CreditCardPayment implements PaymentStrategy {
13      @Override
14      public String pay(double amount) {
15          return "Paid $" + amount + " using Credit Card.";
16      }
17  }
18
19  // Cash Payment
20  class CashPayment implements PaymentStrategy {
21      @Override
22      public String pay(double amount) {
23          return "Paid $" + amount + " in Cash.";
24      }
25  }
26
```

3. Decorator Pattern : Purpose: Allows additional features to be added to rooms such as breakfast, without changing the design of the original objects.

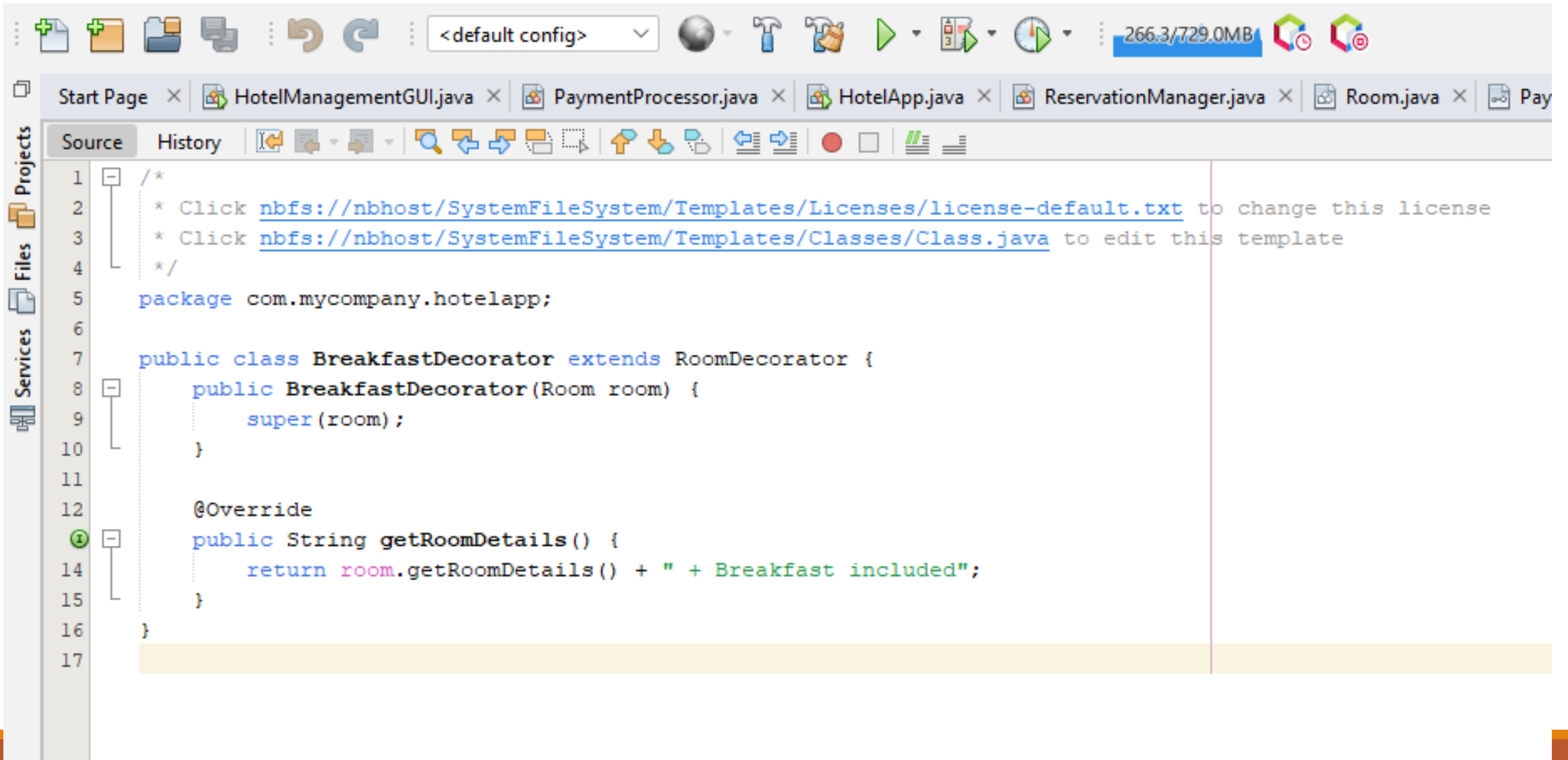
RoomDecorator: An abstract class that is inherited.



```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this lice
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5   package com.mycompany.hotelapp;
6
7
8   public abstract class RoomDecorator extends Room {
9
10      protected Room room;
11
12      public RoomDecorator(Room room) {
13          this.room = room;
14      }
15
16      @Override
17      public abstract String getRoomDetails();
18  }
```



# BreakfastDecorator: Adds breakfast to the room.

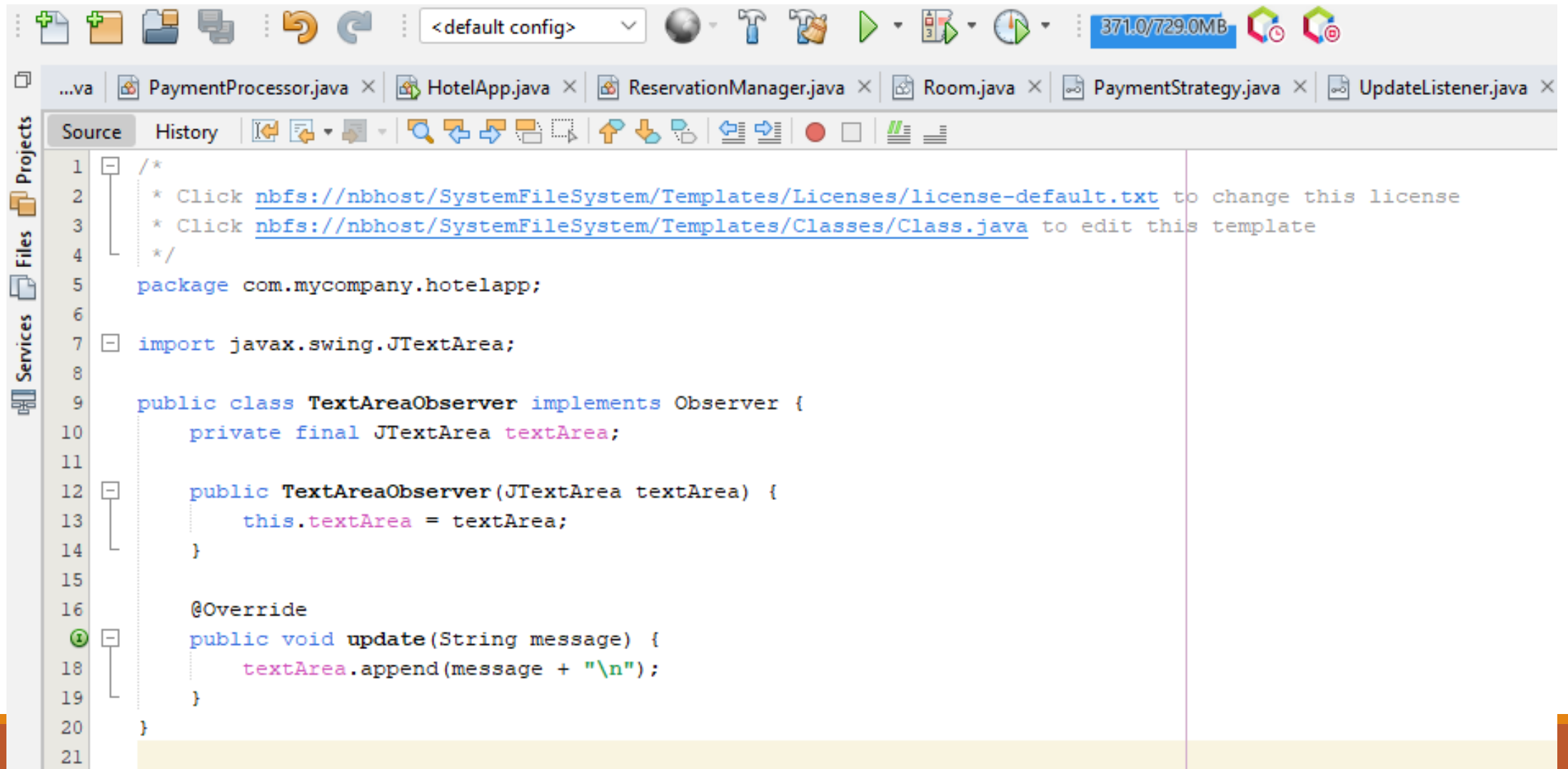


The screenshot shows an IDE window with the file `BreakfastDecorator.java` open. The code defines a `BreakfastDecorator` class that extends `RoomDecorator`. The class has a constructor that takes a `Room` object and calls `super(room)`. It also has an `@Override` method `getRoomDetails()` that returns the room details with the text "Breakfast included" appended.

```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5   package com.mycompany.hotelapp;
6
7   public class BreakfastDecorator extends RoomDecorator {
8       public BreakfastDecorator(Room room) {
9           super(room);
10      }
11
12      @Override
13      public String getRoomDetails() {
14          return room.getRoomDetails() + " + Breakfast included";
15      }
16  }
17
```

Observer Pattern: ReservationManager and PaymentProcessor are now "Subjects", notifying moderators when there is an update.

TextAreaObserver keeps track of data changes, and automatically displays new bookings and payments in the text interface (JTextArea)



```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5  package com.mycompany.hotelapp;
6
7  import javax.swing.JTextArea;
8
9  public class TextAreaObserver implements Observer {
10     private final JTextArea textArea;
11
12     public TextAreaObserver(JTextArea textArea) {
13         this.textArea = textArea;
14     }
15
16     @Override
17     public void update(String message) {
18         textArea.append(message + "\n");
19     }
20 }
21
```

# How to work the project

---

## Booking:

The user enters the room type, name, and phone number.

Can add additional features (such as breakfast).

Your reservation details appear at the bottom of the text interface.

## Payment:

User chooses payment method (cash/card).

The amount and card number are entered (to pay by card).

Payment details are displayed in the text interface.

## Show Reservation

IDE interface showing the source code of `HotelApp.java` and a running application window titled "Hotel Reservation System".

**Source Code (HotelApp.java):**

```
86      outputArea.append("New Payment:\n" + paymentDetails + "\n\n");
87
88      JOptionPane.showMessageDialog(frame, "Payment Completed!");
89  });
90
91  // Show Reservations Button
92  JButton showReservationsButton = new JButton("Show Reservations");
93  showReservationsButton.addActionListener(new ActionListener() {
94      public void actionPerformed(ActionEvent e) {
95          outputArea.append("All Reservations:\n");
96          for (String reservation : reservationList) {
97              outputArea.append(reservation + "\n");
98          }
99          outputArea.append("\n");
100      });
101
102  // Show Payments Button
103  JButton showPaymentsButton = new JButton("Show Payments");
104  showPaymentsButton.addActionListener(new ActionListener() {
105      public void actionPerformed(ActionEvent e) {
106          outputArea.append("All Payments:\n");
107          for (String payment : paymentList) {
108              outputArea.append(payment + "\n");
109          }
110          outputArea.append("\n");
111      });
112
113  buttonPanel.add(bookButton);
114  buttonPanel.add(showReservationsButton);
115  buttonPanel.add(showPaymentsButton);
116
117  frame.setVisible(true);
118 }
```

**Hotel Reservation System Window:**

The window displays a text area for output and three buttons: "Book Room", "Make Payment", and "Show Reservations".

**Input Dialog:**

An "Input" dialog box is shown, prompting the user to "Enter Room Type (Standard/Deluxe):". The dialog includes a text input field and "OK" and "Cancel" buttons.

...va | PaymentProcessor.java x HotelApp.java x ReservationManager.java x Room.java x PaymentStrategy.java x UpdateListener.java x RoomDeco

Source History

```
86      outputArea.append("New Payment:\n" + paymentDetails + "\n\n");
87
88      JOptionPane.showMessageDialog(frame, "Payment Completed!");
89  });
90
91  // Show Reservations Button
92  JButton showReservationsBut
93  showReservationsButton.addA
94  outputArea.append("All
95  for (String reservation
96  outputArea.append(r
97  }
98  outputArea.append("\n")
99  });
100
101  // Show Payments Button
102  JButton showPaymentsButton
103  showPaymentsButton.addAction
104  outputArea.append("All
105  for (String payment : p
106  outputArea.append(p
107  }
108  outputArea.append("\n")
109  });
110
111  buttonPanel.add(bookButton)
112  buttonPanel.add(payButton);
113  buttonPanel.add(showReserva
114  buttonPanel.add(showPayment
115
116  frame.setVisible(true);
117  });
118 }
```

Hotel Reservation System

Input

Add-ons (Enter 'Breakfast' for breakfast, or 'None':)

OK Cancel

Book Room Make Payment Show Reservations

IDE interface showing a Java project named "Hotel Reservation System". The main window displays the code for `HotelApp.java`, which includes a `JOptionPane.showMessageDialog` call and a `for` loop for displaying reservations. The code is as follows:

```
86      outputArea.append("New Payment:\n" + paymentDetails + "\n\n");
87
88      JOptionPane.showMessageDialog(frame, "Payment Completed!");
89  });
90
91  // Show Reservations Button
92  JButton showReservationsBut
93  showReservationsButton.addA
94  outputArea.append("All
95  for (String reservation
96  outputArea.append(r
97  }
98  outputArea.append("\n")
99  });
100
101  // Show Payments Button
102  JButton showPaymentsButton
103  showPaymentsButton.addAction
104  outputArea.append("All
105  for (String payment : p
106  outputArea.append(p
107  }
108  outputArea.append("\n")
109  });
110
111  buttonPanel.add(bookButton)
112  buttonPanel.add(payButton);
113  buttonPanel.add(showReserva
114  buttonPanel.add(showPayment
115
116  frame.setVisible(true);
117  });
118  }
119  }
```

The application window, titled "Hotel Reservation System", features three buttons: "Book Room", "Make Payment", and "Show Reservations". An "Input" dialog box is open, prompting the user to "Enter Customer Name:" with the text "karim" entered in the input field. The dialog box has "OK" and "Cancel" buttons.

IDE interface showing the source code of `HotelApp.java` and a running application window titled "Hotel Reservation System".

The source code in `HotelApp.java` includes:

```
86      outputArea.append("New Payment:\n" + paymentDetails + "\n\n");
87
88      JOptionPane.showMessageDialog(frame, "Payment Completed!");
89  });
90
91  // Show Reservations Button
92  JButton showReservationsButton = new JButton("Show Reservations");
93  showReservationsButton.addActionListener(new ActionListener() {
94      public void actionPerformed(ActionEvent e) {
95          outputArea.append("All Reservations:\n");
96          for (String reservation : reservationList) {
97              outputArea.append(reservation + "\n");
98          }
99      });
100
101  // Show Payments Button
102  JButton showPaymentsButton = new JButton("Show Payments");
103  showPaymentsButton.addActionListener(new ActionListener() {
104      public void actionPerformed(ActionEvent e) {
105          outputArea.append("All Payments:\n");
106          for (String payment : paymentList) {
107              outputArea.append(payment + "\n");
108          }
109      });
110
111  buttonPanel.add(bookButton);
112  buttonPanel.add(payButton);
113  buttonPanel.add(showReservationsButton);
114  buttonPanel.add(showPaymentsButton);
115
116  frame.setVisible(true);
117
118  }
119 }
```

The running application window "Hotel Reservation System" displays a main panel with three buttons: "Book Room", "Make Payment", and "Show Reservations". An "Input" dialog box is open, prompting the user to "Enter Customer Phone:" with the input field containing "0000012222". The dialog has "OK" and "Cancel" buttons.

IDE interface showing a Java project with multiple files open: `PaymentProcessor.java`, `HotelApp.java`, `ReservationManager.java`, `Room.java`, `PaymentStrategy.java`, `UpdateListener.java`, and `RoomDecorator.java`.

The `HotelApp.java` file is selected, showing the following code:

```
86      outputArea.append("New Payment:\n" + paymentDetails + "\n\n");
87
88      JOptionPane.showMessageDialog(frame, "Payment Completed!");
89  });
90
91  // Show Reservations Button
92  JButton showReservationsButton = new JButton("Show Reservations");
93  showReservationsButton.addActionListener(e -> {
94      outputArea.append("All Reservations:\n");
95      for (String reservation : reservationManager.getReservations()) {
96          outputArea.append(reservation + "\n");
97      }
98      outputArea.append("\n");
99  });
100
101  // Show Payments Button
102  JButton showPaymentsButton = new JButton("Show Payments");
103  showPaymentsButton.addActionListener(e -> {
104      outputArea.append("All Payments:\n");
105      for (String payment : paymentProcessor.getPayments()) {
106          outputArea.append(payment + "\n");
107      }
108      outputArea.append("\n");
109  });
110
111  buttonPanel.add(bookButton);
112  buttonPanel.add(payButton);
113  buttonPanel.add(showReservationsButton);
114  buttonPanel.add(showPaymentsButton);
115
116  frame.setVisible(true);
117  });
118  }
119 }
```

The IDE also displays a preview of the `Hotel Reservation System` application window. The window contains a text area with the following text:

New Reservation:  
Customer: karim (0000012222), Room: Standard Room: Basic amenities, single bed. +

A `Message` dialog box is shown, displaying the message "Room Booked!" with an information icon and an `OK` button.

The application window also features a button panel with three buttons: `Book Room`, `Make Payment`, and `Show Reservations`.



IDE interface showing a Java project named "Hotel Reservation System". The main window displays the source code for "HotelApp.java". The code includes methods for handling reservations and payments, such as `outputArea.append("New Payment:\n" + paymentDetails + "\n\n");` and `JOptionPane.showMessageDialog(frame, "Payment Completed!");`. The code is organized into sections for showing reservations and payments.

Overlaid on the IDE is a small dialog box titled "Input". The dialog contains a green question mark icon and the text "Enter Payment Type (Credit/Cash):". Below the text is a text input field. At the bottom of the dialog are "OK" and "Cancel" buttons.

The IDE's toolbar and project explorer are visible at the top and left, respectively. The project explorer shows the following structure:

- Projects
  - Hotel Reservation System
    - src
      - HotelApp.java
      - ReservationManager.java
      - Room.java
      - PaymentStrategy.java
      - UpdateListener.java
      - RoomDecorator.java

IDE interface showing a Java project named "HotelApp.java" with several tabs open: PaymentProcessor.java, HotelApp.java, ReservationManager.java, Room.java, PaymentStrategy.java, UpdateListener.java, and RoomDecorator.java. The code in HotelApp.java is visible, showing a payment completion message dialog and a reservation details display.

```
86      outputArea.append("New Payment:\n" + paymentDetails + "\n\n");
87
88      JOptionPane.showMessageDialog(frame, "Payment Completed!");
89  });
90
91  // Show Reservations Button
92  JButton showReservationsButton = new JButton("Show Reservations");
93  showReservationsButton.addActionListener(e) {
94      outputArea.append("All Reservations:\n");
95      for (String reservation : reservationManager.getAllReservations()) {
96          outputArea.append(reservation + "\n");
97      }
98      outputArea.append("\n");
99  });
100
101  // Show Payments Button
102  JButton showPaymentsButton = new JButton("Show Payments");
103  showPaymentsButton.addActionListener(e) {
104      outputArea.append("All Payments:\n");
105      for (String payment : paymentProcessor.getAllPayments()) {
106          outputArea.append(payment + "\n");
107      }
108      outputArea.append("\n");
109  });
110
111  buttonPanel.add(bookButton);
112  buttonPanel.add(paymentButton);
113  buttonPanel.add(showReservationsButton);
114  buttonPanel.add(showPaymentsButton);
115
116  frame.setVisible(true);
117  }
118 }
```

The application window, titled "Hotel Reservation System", displays the following reservation details:

New Reservation:  
Customer: karim (0000012222), Room: Standard Room: Basic amenities, single bed. +

An "Input" dialog box is shown, prompting the user to "Enter Amount:" with the value "20000" entered. The dialog has "OK" and "Cancel" buttons.

The application window also features three buttons at the bottom: "Book Room", "Make Payment", and "Show Reservations".

IDE interface showing a Java project named "HotelApp.java" with multiple tabs open: PaymentProcessor.java, HotelApp.java, ReservationManager.java, Room.java, PaymentStrategy.java, UpdateListener.java, and RoomDecorator.java. The code in HotelApp.java includes logic for displaying reservation and payment details, and a message dialog for "Payment Completed!".

```
86      outputArea.append("New Payment:\n" + paymentDetails + "\n\n");
87
88      JOptionPane.showMessageDialog(frame, "Payment Completed!");
89  }
90
91  // Show Reservations Button
92  JButton showReservationsButton = new JButton("Show Reservations");
93  showReservationsButton.addActionListener(e -> {
94      outputArea.append("All Reservations:\n");
95      for (String reservation : reservationManager.getReservations()) {
96          outputArea.append(reservation + "\n");
97      }
98      outputArea.append("\n");
99  });
100
101  // Show Payments Button
102  JButton showPaymentsButton = new JButton("Show Payments");
103  showPaymentsButton.addActionListener(e -> {
104      outputArea.append("All Payments:\n");
105      for (String payment : paymentProcessor.getPayments()) {
106          outputArea.append(payment + "\n");
107      }
108      outputArea.append("\n");
109  });
110
111  buttonPanel.add(bookButton);
112  buttonPanel.add(payButton);
113  buttonPanel.add(showReservationsButton);
114  buttonPanel.add(showPaymentsButton);
115
116  frame.setVisible(true);
117 }
118
119 }
```

The "Hotel Reservation System" window displays the following information:

New Reservation:  
Customer: karim (0000012222), Room: Standard Room: Basic amenities, single bed. +

New Payment:  
Paid \$20000.0 using Credit Card. | Card Number: 222236514584

A "Message" dialog box is shown with the text "Payment Completed!" and an "OK" button.

The bottom of the window features three buttons: "Book Room", "Make Payment", and "Show Reservations".

---

Thanks.....

