CS414 – Object Oriented Design

Take Home Final Exam

Taken By: Bryan Saunders

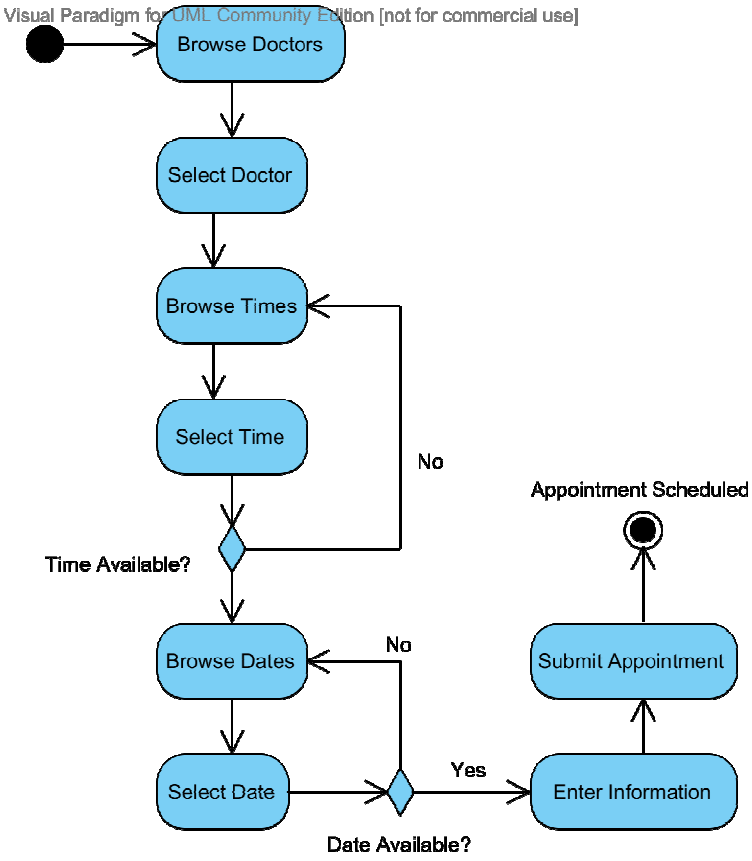
Honor Pledge

"I have not given, received, or used any unauthorized assistance."
Signature: *Bryan Saunders*
Date: 12/11/2012

CS414 Final Exam
Bryan Saunders

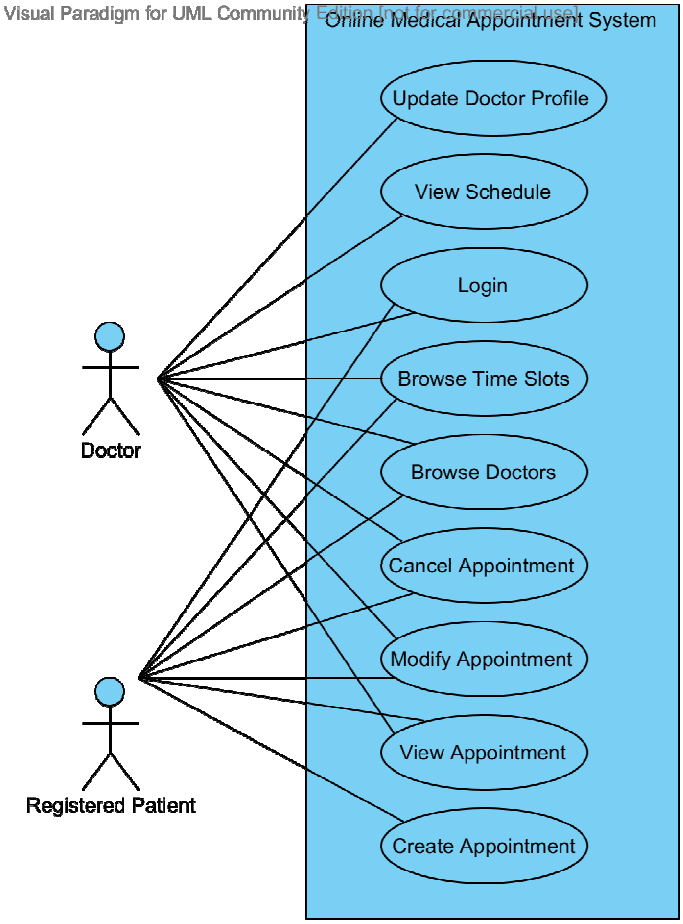
Question 1

Part A



CS414 Final Exam
Bryan Saunders

Part B



CS414 Final Exam
Bryan Saunders

Part C

Use Case Scenario 1

Name	Modify Appointment
Scope	CS414 Online Medical Appointment Scheduling System
Level	User Goal
Primary Actor	Registered Patient
Stakeholders and Interests	Registered Patient: Change appointment information Doctor: Provide the correct care to the patient at an appropriate time
Preconditions	Registered Patient logged in to the system
Success Guarantee	Registered Patient has modified appointment successfully
Main Success Scenario	1. Registered Patient (RP) chooses "View Appointments" 2. System loads list of RP's appointments 3. RP chooses appointment to modify 4. System loads appointment information form 5. System pre-populates form with current data 6. RP changes appointment information 7. RP selects "Save Appointment" 8. System saves new appointment information 9. System notifies Doctor and RP of appointment changes 10. System redirects RP to home screen
Extensions	Step 6: Invalid information entered 1. System displays appropriate error message(s) 2. User corrects invalid field(s) 3. System continues at step 7
Special Requirements	None
Variations in Technology & Data	Step 6: The system shall validate field data after the field is populated and without requiring form submission.
Frequency of Occurrence	Every time a patient needs to modify an appointment, infrequently.
Miscellaneous	What is the best way to notify the Doctor of appointment changes?

Use Case Scenario 2

Name	View Appointment Schedule
Scope	CS414 Online Medical Appointment Scheduling System
Level	User Goal
Primary Actor	Doctor
Stakeholders and Interests	Doctor: View all upcoming appointments
Preconditions	Doctor logged in to the system
Success Guarantee	Doctor has viewed appointment schedule
Main Success Scenario	1. Doctor selects "View Schedule" 2. System loads schedule selection form 3. Doctor selects date 4. Doctor selects "View Schedule" 5. System loads schedule 6. Doctor views schedule 7. Doctors selects "Done" 8. System redirects Doctor to home screen
Extensions	Step 6: Print Schedule 1. Doctor selects "Print" 2. System loads print dialog 3. Doctor sets print options 4. Doctor selects "Ok" 5. System sends print command 6. System closes print dialog 7. System continues at step 7
Special Requirements	None
Variations in Technology & Data	None
Frequency of Occurrence	Daily
Miscellaneous	None

CS414 Final Exam
Bryan Saunders

Question 2

Part A

I Used the "Replace Type Code with State/Strategy" refactoring. This refactoring is for replacing type codes with a State class and a set of TypeCode subclasses. In this case I replaced the Enum with a RentalState parent object and four subclasses.

Final Code

```
public class RentalStateMachine {
	private RentalState currentState;

	public RentalStateMachine() {
		this.currentState = new ForRent();
	}

	public void rent() throws IllegalEventException {
		RentalState newState = this.currentState.rent();
		this.currentState = newState;
	}

	public void returnRental() throws IllegalEventException {
		RentalState newState = this.currentState.returnRental();
		this.currentState = newState;
	}

	public void setForSale() throws IllegalEventException {
		RentalState newState = this.currentState.setForSale();
		this.currentState = newState;
	}

	public void sell() throws IllegalEventException {
		RentalState newState = this.currentState.sell();
		this.currentState = newState;
	}

	public RentalState getCurrentState() {
		return this.currentState;
	}

	public void setCurrentState(RentalState currentState) {
		this.currentState = currentState;
	}
}

abstract class RentalState {
	RentalState sell() throws IllegalEventException {
		throw new IllegalEventException();
	}

	RentalState setForSale() throws IllegalEventException {
		throw new IllegalEventException();
	}
```

```
	RentalState returnRental() throws IllegalEventException {
		throw new IllegalEventException();
	}

	RentalState rent() throws IllegalEventException {
		throw new IllegalEventException();
	}
}

class ForRent extends RentalState {
	@Override
	RentalState rent() throws IllegalEventException {
		return new Rented();
	}

	@Override
	RentalState setForSale() throws IllegalEventException {
		return new ForSale();
	}
}

class Rented extends RentalState {
	@Override
	RentalState returnRental() throws IllegalEventException {
		return new ForRent();
	}
}

class ForSale extends RentalState {
	@Override
	RentalState sell() throws IllegalEventException {
		return new Sold();
	}
}

class Sold extends RentalState {

}

class IllegalEventException extends Exception {
	private static final long serialVersionUID = 1L;

}
```

Part B

```
public class RentalStateMachine {
	private RentalState currentState;

	public RentalStateMachine() {
		this.currentState = new ForRent();
	}

	public void rent() throws IllegalEventException {
		RentalState newState = this.currentState.rent();
		this.currentState = newState;
	}

	public void returnRental() throws IllegalEventException {
		RentalState newState = this.currentState.returnRental();
		this.currentState = newState;
	}

	public void setForSale() throws IllegalEventException {
		RentalState newState = this.currentState.setForSale();
		this.currentState = newState;
	}

	public void sell() throws IllegalEventException {
		RentalState newState = this.currentState.sell();
		this.currentState = newState;
	}

	public void withdraw() throws IllegalEventException {
		RentalState newState = this.currentState.withdraw();
		this.currentState = newState;
	}

	public RentalState getCurrentState() {
		return this.currentState;
	}

	public void setCurrentState(RentalState currentState) {
		this.currentState = currentState;
	}
}

abstract class RentalState {
	RentalState sell() throws IllegalEventException {
		throw new IllegalEventException();
	}

	RentalState setForSale() throws IllegalEventException {
		throw new IllegalEventException();
	}

	RentalState returnRental() throws IllegalEventException {
		throw new IllegalEventException();
```

```
	}

	RentalState rent() throws IllegalEventException {
		throw new IllegalEventException();
	}

	RentalState withdraw() throws IllegalEventException {
		return new Withdrawn();
	}
}

class ForRent extends RentalState {
	@Override
	RentalState rent() throws IllegalEventException {
		return new Rented();
	}

	@Override
	RentalState setForSale() throws IllegalEventException {
		return new ForSale();
	}
}

class Rented extends RentalState {
	@Override
	RentalState returnRental() throws IllegalEventException {
		return new ForRent();
	}
}

class ForSale extends RentalState {
	@Override
	RentalState sell() throws IllegalEventException {
		return new Sold();
	}
}

class Sold extends RentalState {
	@Override
	RentalState withdraw() throws IllegalEventException {
		throw new IllegalEventException();
	}
}

class Withdrawn extends RentalState {
	@Override
	RentalState sell() throws IllegalEventException {
		return new Withdrawn();
	}

	@Override
	RentalState returnRental() throws IllegalEventException {
		return new Withdrawn();
	}
```

```
	@Override
	RentalState rent() throws IllegalEventException {
		return new Withdrawn();
	}

	@Override
	RentalState setForSale() throws IllegalEventException {
		return new Withdrawn();
	}
}

class IllegalEventException extends Exception {
	private static final long serialVersionUID = 1L;

}
```

Part C

```
public class OriginalRentalStateMachine {
	public enum RentalState {
		ForRent, Rented, ForSale, Sold, Withdrawn;
	}

	private RentalState currentState;

	public void rent() throws IllegalEventException {
		if (currentState == RentalState.Withdrawn) {
			currentState = RentalState.Withdrawn;
		} else if (currentState == RentalState.ForRent) {
			currentState = RentalState.Rented;
		} else {
			throw new IllegalEventException();

		}
	}

	public void returnRental() throws IllegalEventException {
		if (currentState == RentalState.Withdrawn) {
			currentState = RentalState.Withdrawn;
		} else if (currentState == RentalState.Rented) {
			currentState = RentalState.ForRent;
		} else {
			throw new IllegalEventException();
		}
	}

	public void setForSale() throws IllegalEventException {
		if (currentState == RentalState.Withdrawn) {
			currentState = RentalState.Withdrawn;
		} else if (currentState == RentalState.ForRent) {
			currentState = RentalState.ForSale;
		} else {
			throw new IllegalEventException();
		}
	}

	public void sell() throws IllegalEventException {
		if (currentState == RentalState.Withdrawn) {
			currentState = RentalState.Withdrawn;
		} else if (currentState == RentalState.ForSale) {
			currentState = RentalState.Sold;
		} else {
			throw new IllegalEventException();
		}
	}

	public void withdraw() throws IllegalEventException{
		if (currentState != RentalState.Sold) {
			currentState = RentalState.Withdrawn;
		} else {
```

```
                    throw new IllegalEventException();
                }
        }
}

class IllegalEventException extends Exception {
        private static final long serialVersionUID = 1L;

}
```

CS414 Final Exam
Bryan Saunders

Part D

Changes made for Part B

+ 1 Class
+ 6 Methods (4 inside new class)
+ 15 Lines of Code Total

Changes made for Part C

+ 1 Enum
+ 1 New Method
+ 4 Method Updates
+ 17 Lines of Code Total

The changes to both Part B and C were about the same number of lines, however the changes to Part B left the code in a much better state. Adding a new state was much easier after applying the state pattern because I only needed to implement a new class. In the original code I had to go back and change all of the previously existing methods to account for the new state.

CS414 Final Exam
Bryan Saunders

Question 3

Part A

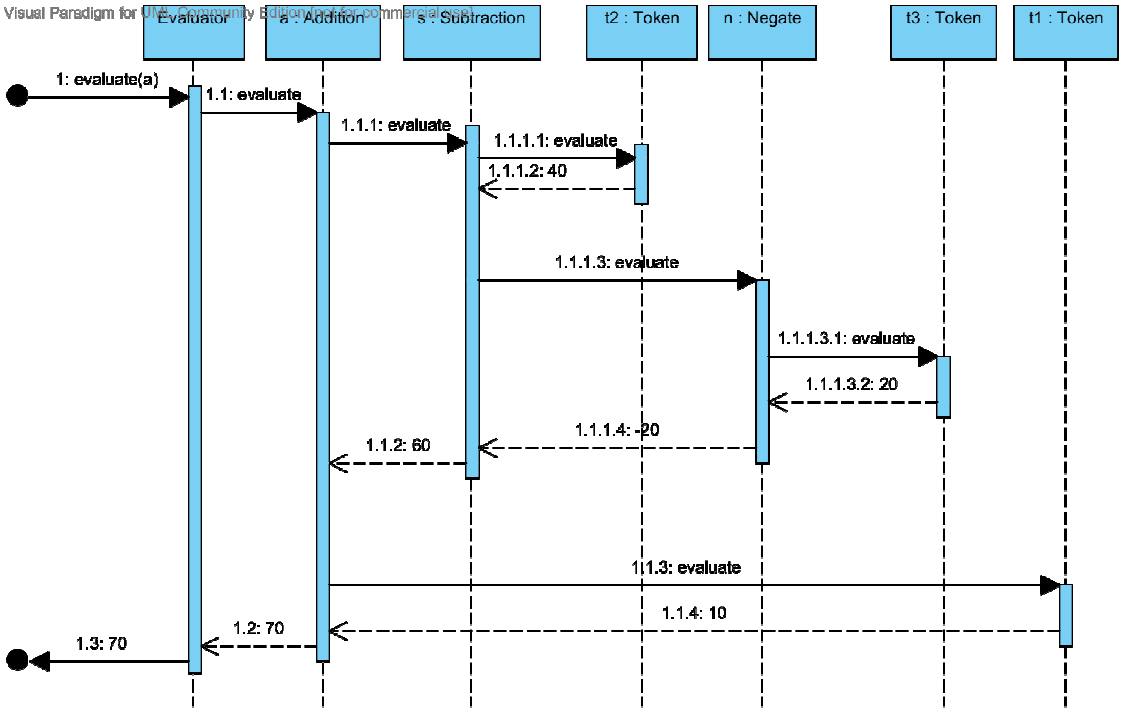
I added an Abstract method to the Node class called Evaluate. I then moved the logic from each of the if branches to its appropriate sub-class and updated them where necessary to compile. Then I removed the if conditionals and replaced them with a call to the abstract evaluate method.

```
import java.util.HashMap;

/*** File Evaluator.java ***/
public class Evaluator {

	public int evaluate(Node node) {
		return node.evaluate();
	}
}

/*** File Node.java ***/
abstract class Node {
	abstract public int evaluate();
}

/*** File Expression.java ***/
abstract class Expression extends Node {
	private Node left;
	private Node right;

	public Node getLeft() {
		return left;
	}

	public Node getRight() {
		return right;
	}
}

/*** File Addition.java ***/
class Addition extends Expression {
	@Override
	public int evaluate() {
		int leftOperand = getLeft().evaluate();
		int rightOperand = getRight().evaluate();
		return leftOperand + rightOperand;
	}
}

/*** File Subtraction.java ***/
class Subtraction extends Expression {
	@Override
	public int evaluate() {
		int leftOperand = getLeft().evaluate();
```

```
			int rightOperand = getRight().evaluate();
			return leftOperand - rightOperand;
		}
}

/*** File Negate.java ***/
class Negate extends Expression {
		@Override
		public int evaluate() {
			int leftOperand = getLeft().evaluate();
			return (-leftOperand);
		}
}

/*** File Token.java ***/
class Token extends Node {
		public static final int IDENTIFIER = 1;
		public static final int INTEGER_LITERAL = 2;
		private HashMap<String, Integer> IdentifierToValue;
		private int type;
		private String text;

		public String getText() {
			return text;
		}

		public int getType() {
			return type;
		}

		@Override
		public int evaluate() {
			if (type == Token.IDENTIFIER)
				return IdentifierToValue.get(getText()).intValue();
			else
				return Integer.parseInt(getText());
		}
}
```

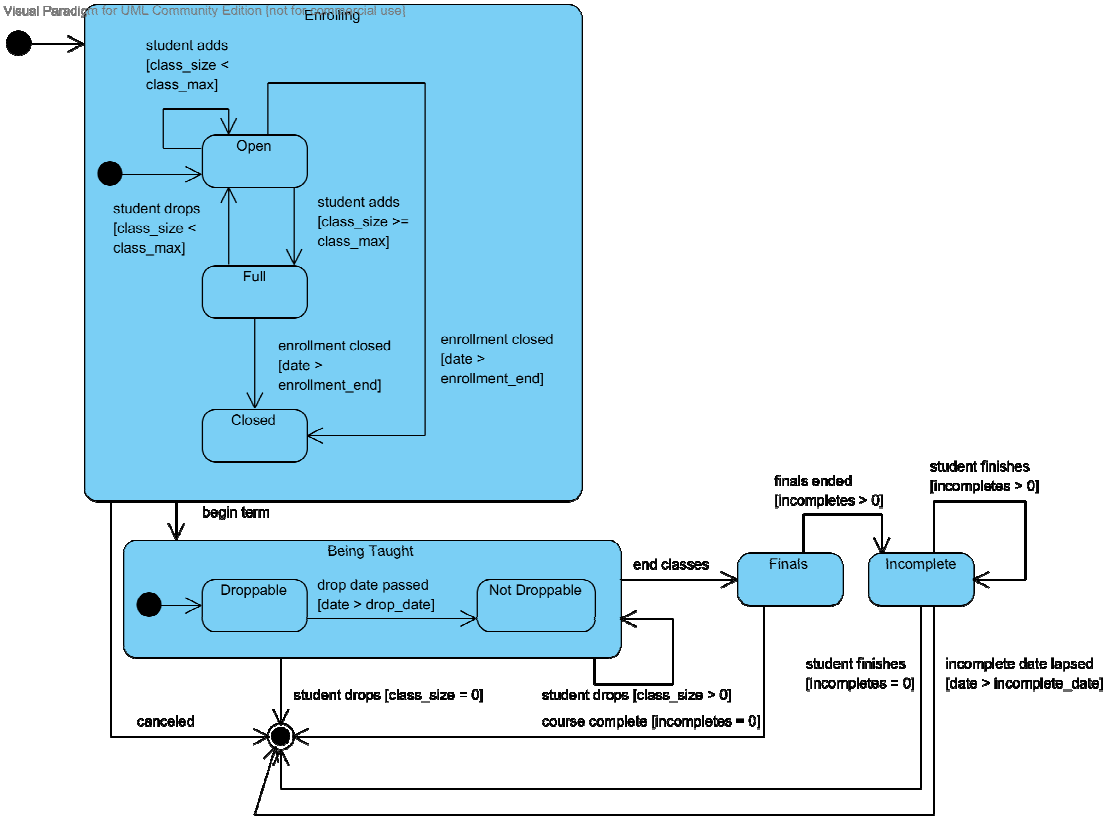
CS414 Final Exam
Bryan Saunders

Part B



CS414 Final Exam
Bryan Saunders

Question 4



CS414 Final Exam
Bryan Saunders

Question 5

Part A

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.HashMap;

public class SaleFrame extends JFrame implements ActionListener {
	private static final long serialVersionUID = 1L;

	private JButton enterItemButton = new JButton("Enter Item");
	private JTextField priceField = new JTextField(30);
	private JTextField itemIDField = new JTextField(10);
	private JTextField quantityField = new JTextField(10);

	public SaleFrame() {
		Container container = getContentPane();
		container.setLayout(new FlowLayout(FlowLayout.CENTER));
		container.add(new JLabel("Enter Item ID(1/2)"));

		container.add(itemIDField);
		container.add(new JLabel("Enter quantity of the item"));
		container.add(quantityField);
		container.add(enterItemButton);
		container.add(new JLabel("The total price of the item"));
		container.add(priceField);
		priceField.setEditable(false);
		enterItemButton.addActionListener(this);
	}

	public void actionPerformed(ActionEvent e) {
		int itemID = Integer.parseInt(itemIDField.getText());
		int quantity = Integer.parseInt(quantityField.getText());

		SaleController controller = new SaleController();
		int price = controller.computePrice(itemID, quantity);
		displayPrice(price);
	}

	public void displayPrice(int price) {
		priceField.setText("The price for the entered item is " + price);
	}

	public static void main(String[] args) {
		SaleFrame saleframe = new SaleFrame();
		saleframe.setSize(200, 200);
		saleframe.setVisible(true);
	}
}



class SaleController {
```

```
	int computePrice(int itemID, int quantity) {
		ProductCatalog productCatalog = ProductCatalog.getInstance();
		ProductDescription description = productCatalog
				.getProductDescription(itemID);
		PriceCalculator calculator = new PriceCalculator();
		int price = calculator.calculatePrice(quantity, description);
		return price;
	}
}

class ProductCatalog {
	private HashMap<Integer, ProductDescription>
itemIDtoProductDescriptionMap = new HashMap<Integer, ProductDescription>();

	protected ProductCatalog() {
		super();
		itemIDtoProductDescriptionMap.put(1, new ProductDescription(10));
		itemIDtoProductDescriptionMap.put(2, new ProductDescription(15));
	}

	private static ProductCatalog INSTANCE;

	public static ProductCatalog getInstance() {
		if (INSTANCE != null)
			return INSTANCE;
		else
			return new ProductCatalog();
	}

	public ProductDescription getProductDescription(int itemID) {

		return itemIDtoProductDescriptionMap.get(itemID);
	}
}

class ProductDescription {
	private int unitPrice;

	public ProductDescription(int unitPrice) {
		this.unitPrice = unitPrice;
	}

	public int getUnitPrice() {
		return unitPrice;
	}
}

class PriceCalculator {
	public int calculatePrice(int quantity, ProductDescription description)
{
		int price = quantity * description.getUnitPrice();
		return price;
	}
}
```

CS414 Final Exam
Bryan Saunders

Part B

```
import java.awt.Container;
import java.awt.FlowLayout;
import java.util.LinkedList;
import java.util.List;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class SaleFrame extends JFrame implements ActionListener, PriceObserver
{
	private static final long serialVersionUID = 1L;

	private JButton enterItemButton = new JButton("Enter Item");
	private JTextField priceField = new JTextField(30);
	private JTextField itemIDField = new JTextField(10);
	private JTextField quantityField = new JTextField(10);

	private SaleController controller;

	public SaleFrame() {
		Container container = getContentPane();
		container.setLayout(new FlowLayout(FlowLayout.CENTER));
		container.add(new JLabel("Enter Item ID(1/2)"));

		container.add(itemIDField);
		container.add(new JLabel("Enter quantity of the item"));
		container.add(quantityField);
		container.add(enterItemButton);
		container.add(new JLabel("The total price of the item"));
		container.add(priceField);
		priceField.setEditable(false);
		enterItemButton.addActionListener(this);

		controller = new SaleController();
		controller.addObserver(this);
	}

	public void actionPerformed(ActionEvent e) {
		int itemID = Integer.parseInt(itemIDField.getText());
		int quantity = Integer.parseInt(quantityField.getText());

		controller.computePrice(itemID, quantity);
	}

	public void displayPrice(int price) {
		priceField.setText("The price for the entered item is " + price);
	}
```

```
	public static void main(String[] args) {
		SaleFrame saleframe = new SaleFrame();
		saleframe.setSize(200, 200);
		saleframe.setVisible(true);
	}

	@Override
	public void priceUpdated(int price) {
		displayPrice(price);
	}
}

class SaleController {

	List<PriceObserver> observers;

	public SaleController(){
		observers = new LinkedList<PriceObserver>();
	}

	void addObserver(PriceObserver observer){
		observers.add(observer);
	}

	int computePrice(int itemID, int quantity) {
		ProductCatalog productCatalog = ProductCatalog.getInstance();

		ProductDescription description = productCatalog
				.getProductDescription(itemID);
		PriceCalculator calculator = new PriceCalculator();

		int price = calculator.calculatePrice(quantity, description);

		firePriceUpdatedEvent(price);
		return price;
	}

	private void firePriceUpdatedEvent(int price) {
		for(PriceObserver observer : observers){
			observer.priceUpdated(price);
		}
	}
}

interface PriceObserver {
	void priceUpdated(int price);
}

class ProductCatalog {
	private HashMap<Integer, ProductDescription>
itemIDtoProductDescriptionMap = new HashMap<Integer, ProductDescription>();

	protected ProductCatalog() {
		super();
```

```
			itemIDtoProductDescriptionMap.put(1, new ProductDescription(10));
			itemIDtoProductDescriptionMap.put(2, new ProductDescription(15));
		}

		private static ProductCatalog INSTANCE;

		public static ProductCatalog getInstance() {
			if (INSTANCE != null)
				return INSTANCE;
			else
				return new ProductCatalog();
		}

		public ProductDescription getProductDescription(int itemID) {

			return itemIDtoProductDescriptionMap.get(itemID);
		}
}

class ProductDescription {
		private int unitPrice;

		public ProductDescription(int unitPrice) {
			this.unitPrice = unitPrice;
		}

		public int getUnitPrice() {
			return unitPrice;
		}
}

class PriceCalculator {
		public int calculatePrice(int quantity, ProductDescription description)
{
			int price = quantity * description.getUnitPrice();
			return price;
		}
}
```