Candidate Code: kly215

Criterion C

I. Overview

I.1. Techniques

- Abstract, user-defined objects.
- Encapsulation (private instance variables coupled with public accessor and mutator methods).
- Parameter passing.
- Method returning (a) value(s).
- Overloading constructors (differ by their parameter lists)
- Random number generation
- For and while loops
- Simple and compound selection (if/else)
- Error handling (try and catch exceptions).
- Writing into and reading from text files (for storage).
- Parsing algorithms (that read from text the instance variables of an object of a class).
- Graphic libraries.
- Communicating with the user with JOptionPane.

I.2. Project Tree

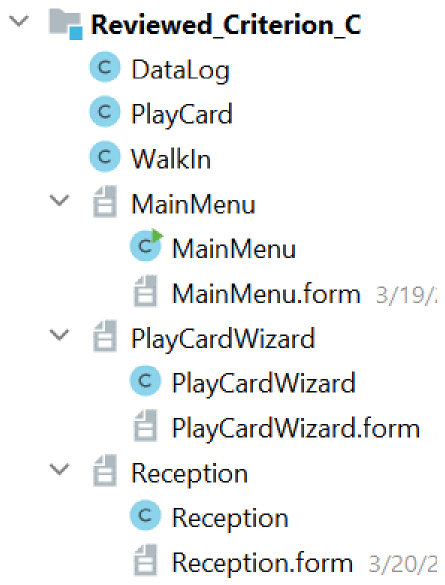


Figure 1 List of Classes Used in The Project

I.3. Brief (In order of relevance)

- **Main Menu**: Launched at startup and connects the two UI forms “PlayCardWizard” and “Reception” together.

- **DataLog**: Abstract class that reads information from, parses, and writes into text files (the storage units used in this Internal Assessment).
- **PlayCard**: Abstract class for a play card. It has a serial number and a balance and methods in accordance.
- **PlayCardWizard**: UI where play cards are created and maintained (through deposits).
- **WalkIn**: A walk-in is any child that enters the soft play area to play for 1 or 2 hours. This class holds the information related to each child entering such as their name, the number of hours they're playing, and fees.
- **Reception**: UI form where walk-ins are registered, paid for using a PlayCard object, and reported in a summary.

I.5. Graphical User Interface

I.5.1. Importance of GUIs

A Graphic User Interface (GUI) omits the user's need to use a command prompt or console (that needs programming knowledge) by making the functionality of an application comprehensible and accessible.

Main Menu

Contains two buttons that call other interfaces from other classes such as the PlayCardWizard and Reception classes to open.

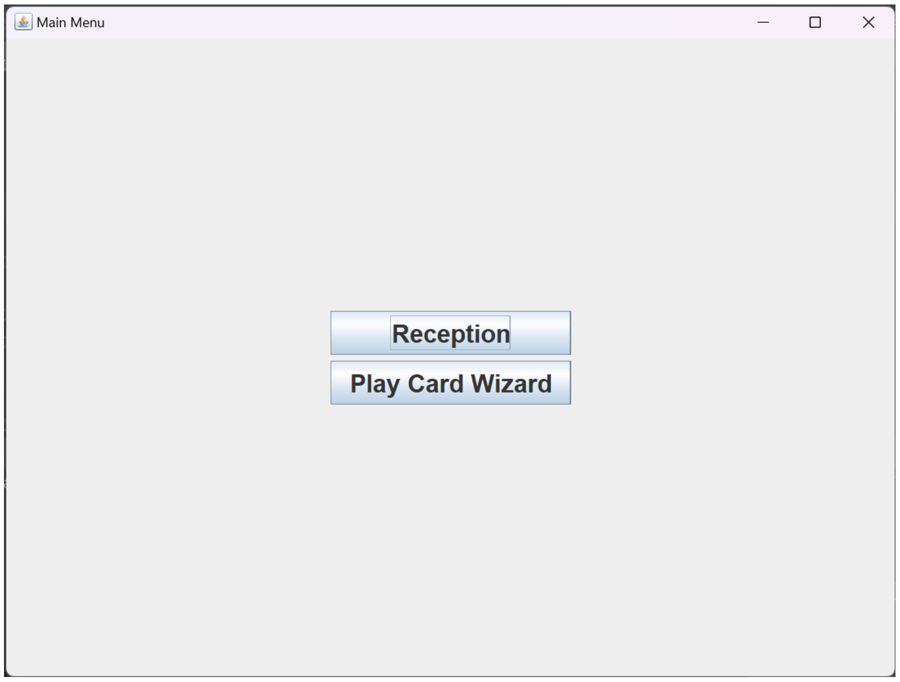
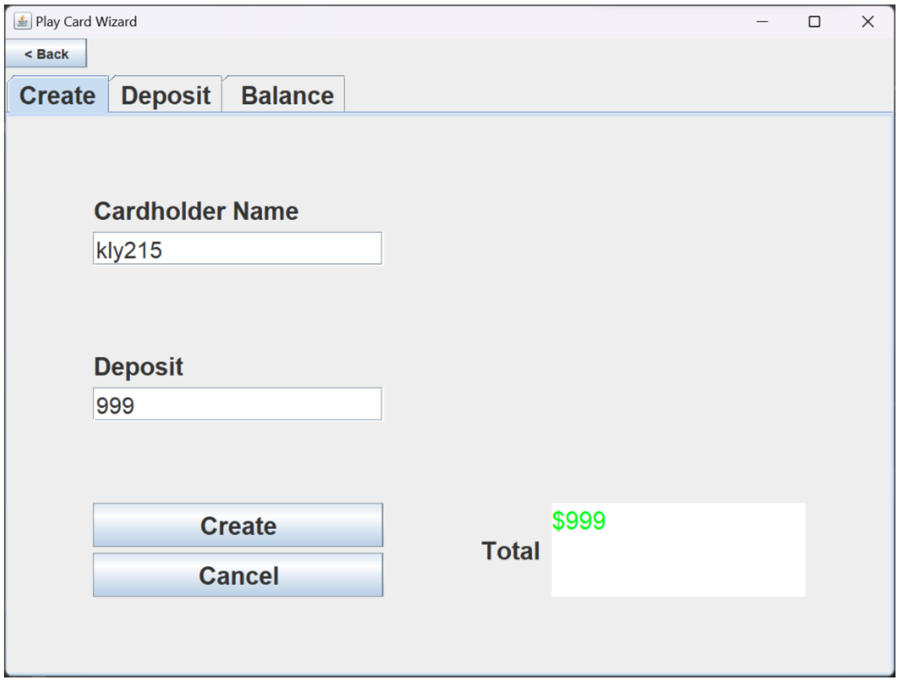


Figure 2 Main Menu GUI

PlayCardWizard

This frame has 3 features:

(1) Creating a new play card: Takes the customer's name and an initial deposit to create a new PlayCard object with.



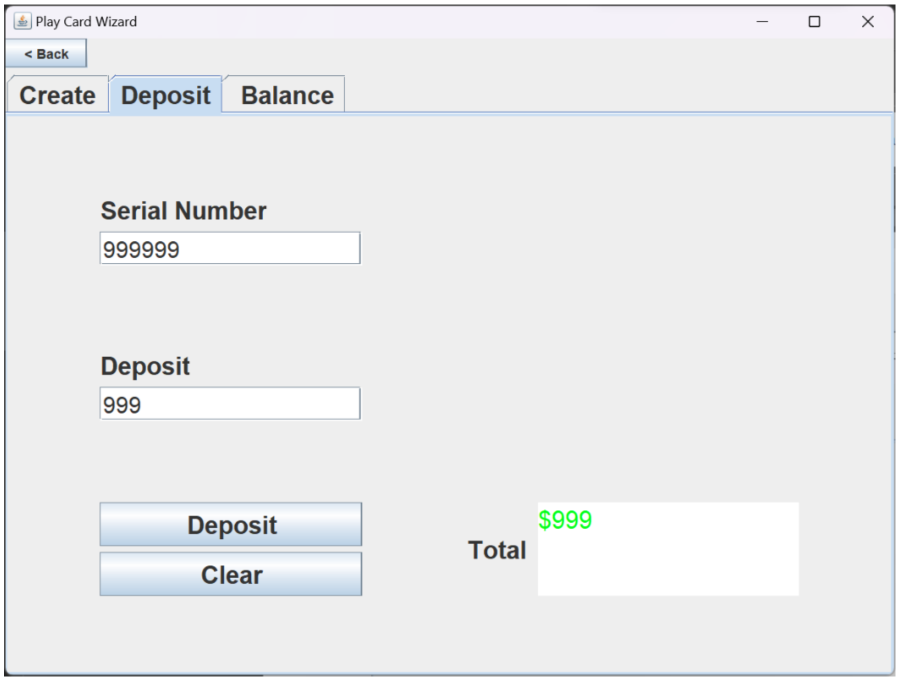
Cardholder Name
kly215

Deposit
999

Total $999

Figure 3 Creating a Play Card Using PlayCardWizard GUI

(2) Depositing into an existing play card: Opens the .txt file pertaining to the serial number provided and adds to the card's balance.



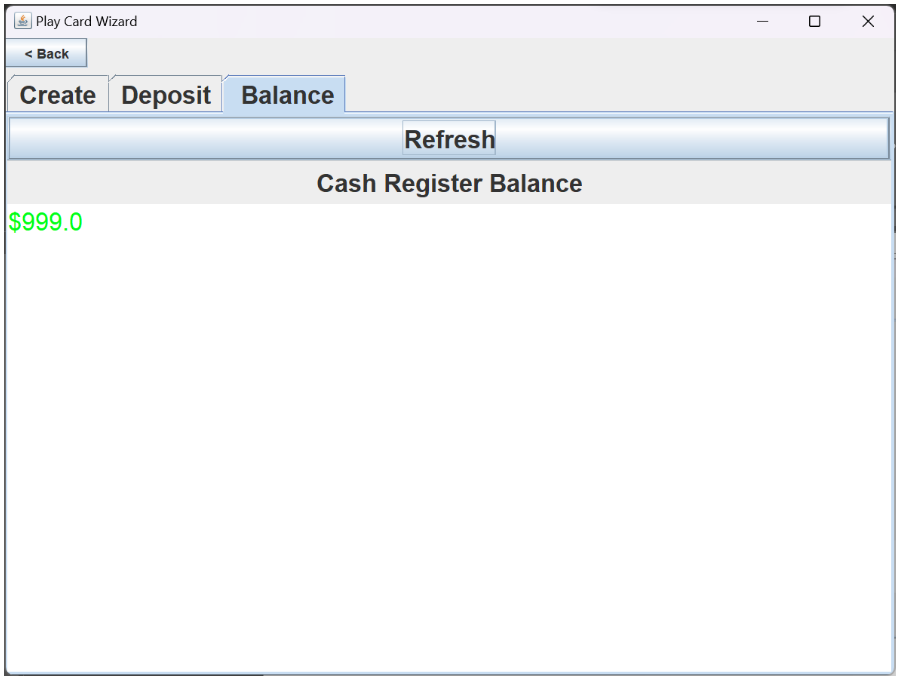
Serial Number
999999

Deposit
999

Total $999

Figure 4 Depositing into a Play Card Using Play Card Wizard GUI

(3) Checking the cash register balance: the sum of all kinds of deposits appears when refreshed.



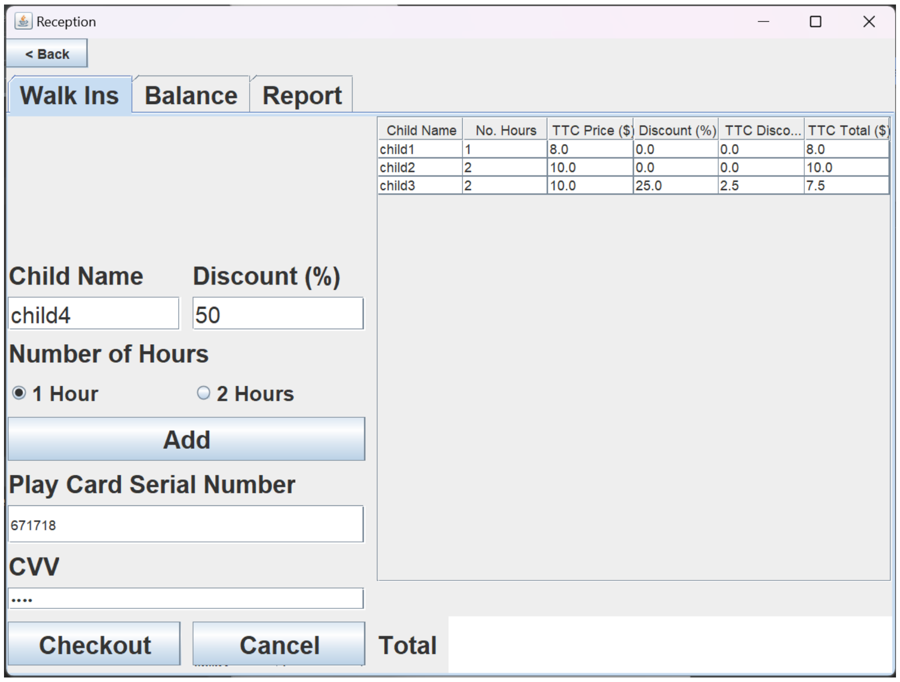
$999.0

Figure 5 Viewing the Cash Register Balance Using Play Card Wizard GUI

Reception

This frame has 3 features:

(1) Registering Walk-ins: Takes the name, percent discount, and number of hours each kid plays to create a WalkIn object. For payment, it collects the serial number of an existing play card to withdraw the invoice and uses the card verification value (CVV) to authenticate.

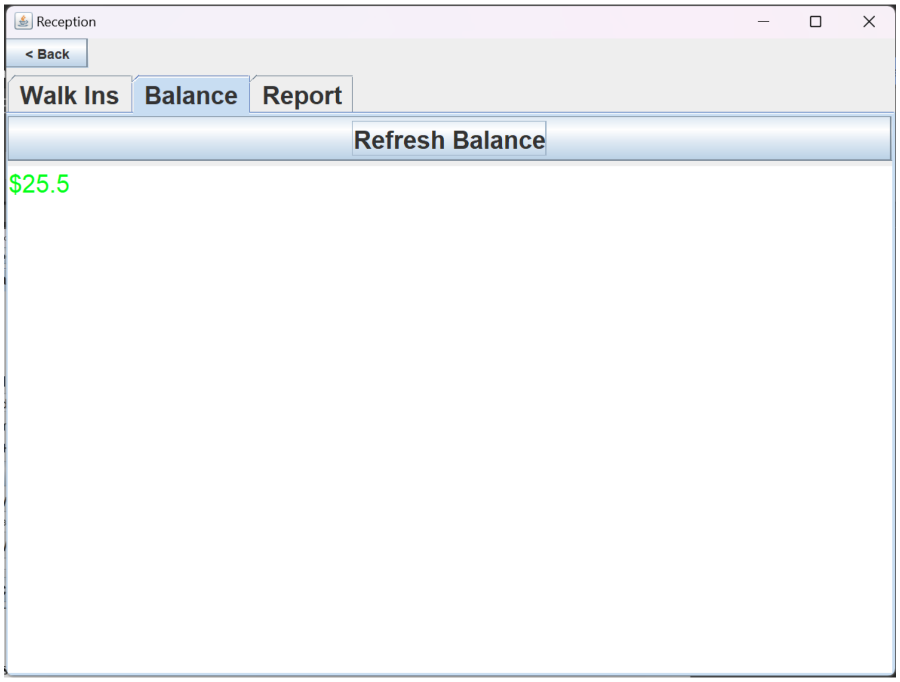


Child Name	No. Hours	TTC Price ($)	Discount (%)	TTC Disco...	TTC Total ($)
child1	1	8.0	0.0	0.0	8.0
child2	2	10.0	0.0	0.0	10.0
child3	2	10.0	25.0	2.5	7.5

Child Name: child4
Discount (%): 50
Number of Hours: [✓ 1 Hour] 2 Hours
Play Card Serial Number: 671718
CVV: ····

Figure 6 Registering Walk-Ins Using Reception GUI

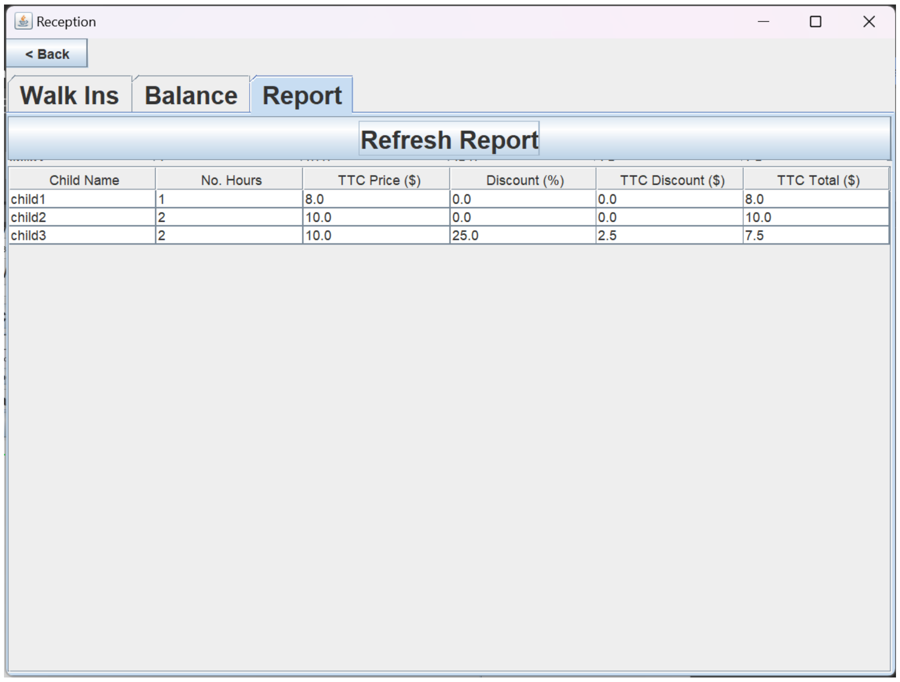
(2) Checking The Cash Register Balance: Upon paying the entrance fees, the invoices are added automatically to the cash register's account. The refresh button displays this value when clicked.



$25.5

Figure 7 Viewing the Cash Register Balance Using Reception GUI

(3) Generating Walk-in Reports: When refreshed, the application outputs each WalkIn object into the report table.



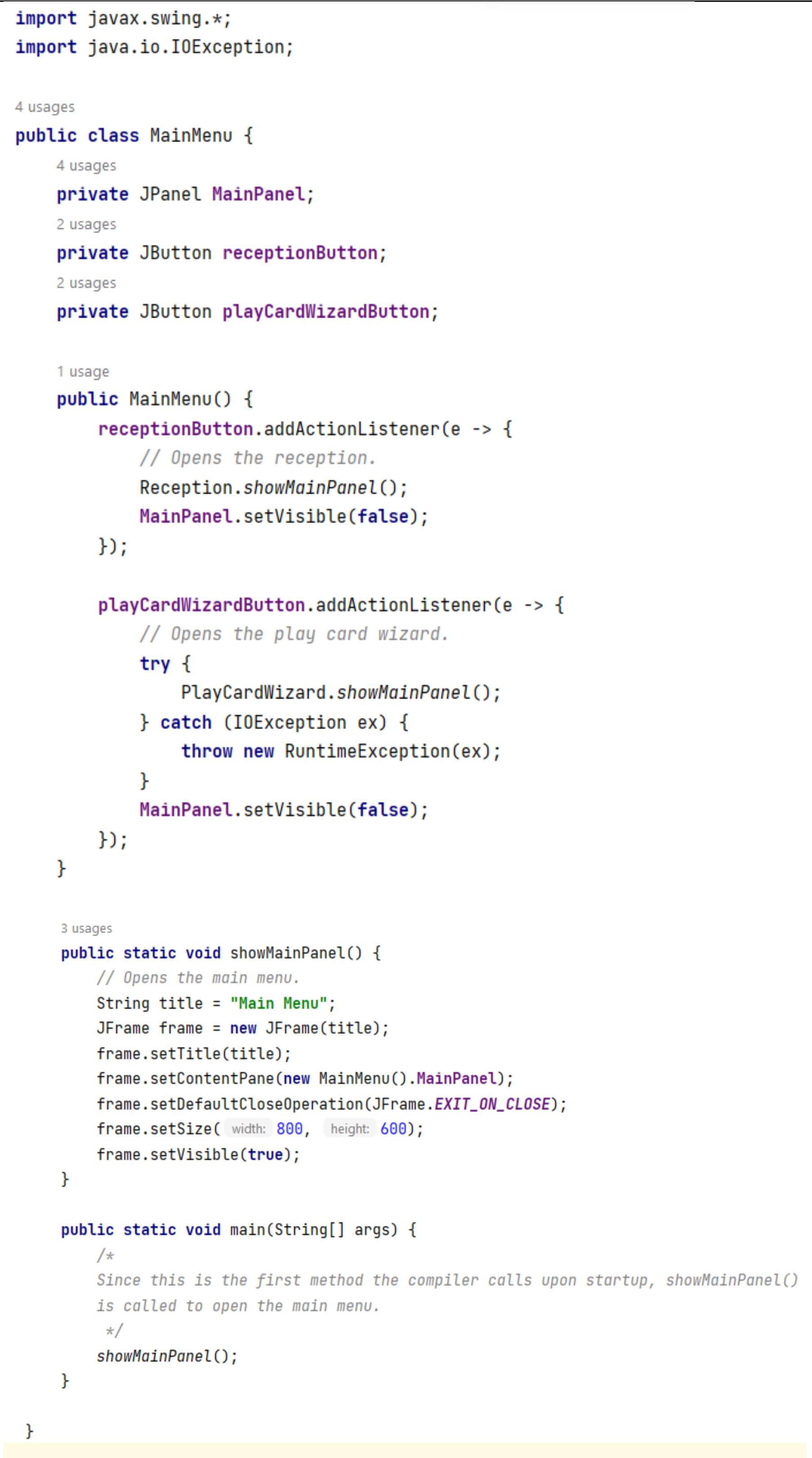
Child Name	No. Hours	TTC Price ($)	Discount (%)	TTC Discount ($)	TTC Total ($)
child1	1	8.0	0.0	0.0	8.0
child2	2	10.0	0.0	0.0	10.0
child3	2	10.0	25.0	2.5	7.5

Figure 8 Generating a Summary of the Walk-Ins Using Reception GUI

II. Further Explanation

II.1. Code

<u>II.1.1 Main Menu</u>

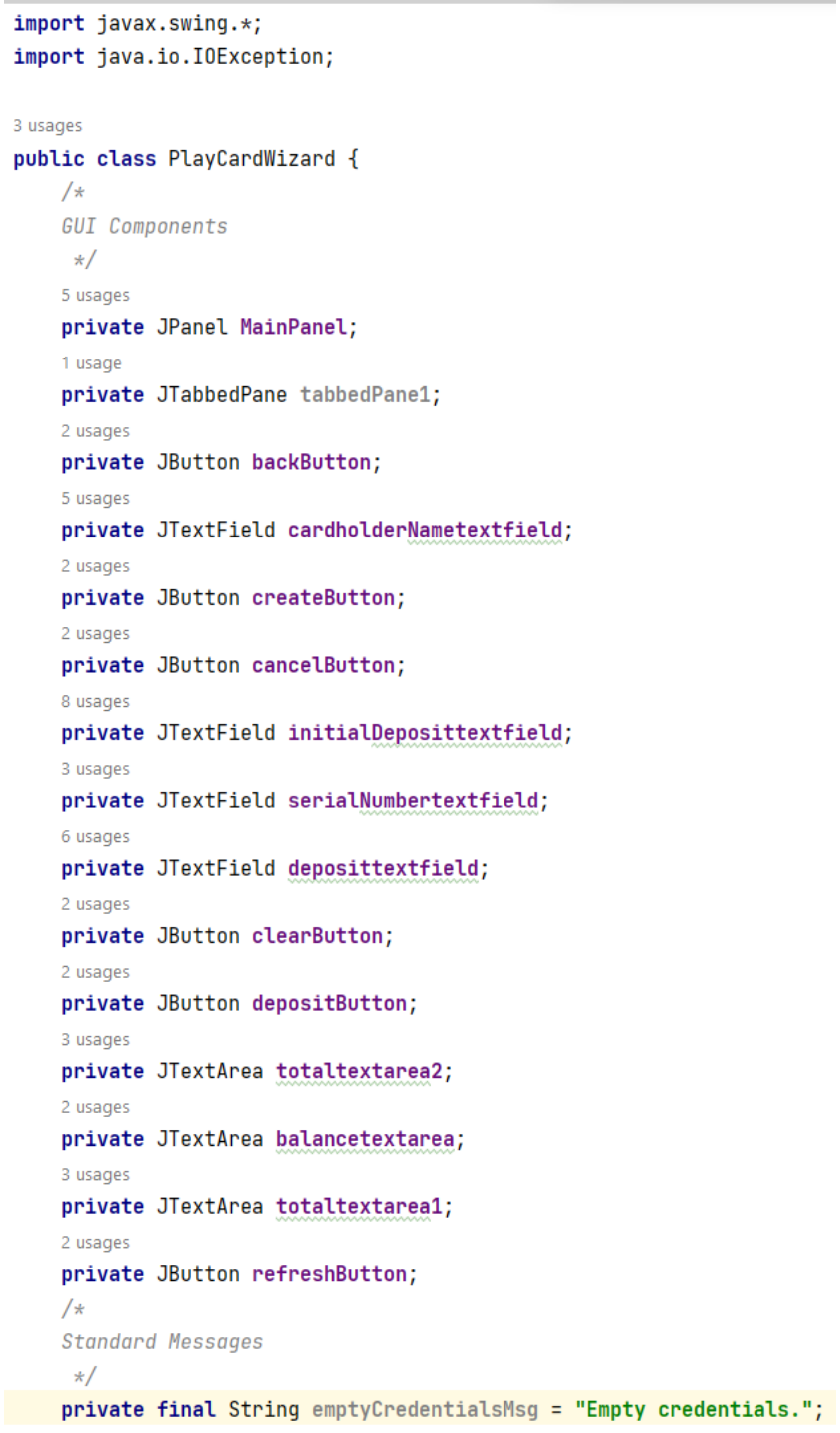


```
import javax.swing.*;
import java.io.IOException;

4 usages
public class MainMenu {
    4 usages
    private JPanel MainPanel;
    2 usages
    private JButton receptionButton;
    2 usages
    private JButton playCardWizardButton;

    1 usage
    public MainMenu() {
        receptionButton.addActionListener(e -> {
            // Opens the reception.
            Reception.showMainPanel();
            MainPanel.setVisible(false);
        });

        playCardWizardButton.addActionListener(e -> {
            // Opens the play card wizard.
            try {
                PlayCardWizard.showMainPanel();
            } catch (IOException ex) {
                throw new RuntimeException(ex);
            }
            MainPanel.setVisible(false);
        });
    }

    3 usages
    public static void showMainPanel() {
        // Opens the main menu.
        String title = "Main Menu";
        JFrame frame = new JFrame(title);
        frame.setTitle(title);
        frame.setContentPane(new MainMenu().MainPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize( width: 800,  height: 600);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        /*
        Since this is the first method the compiler calls upon startup, showMainPanel()
        is called to open the main menu.
         */
        showMainPanel();
    }

}
```

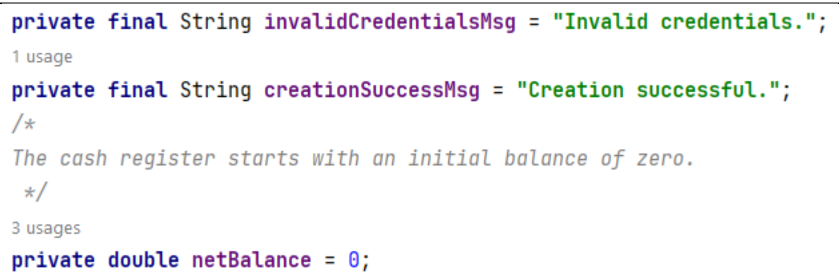
The showMainPanel() method is prominent in all the UI forms in this project. It sets up new JFrames and its media.

It is through clicking either button that this method is invoked to open the other interfaces.

II.1.2. Play Card Wizard

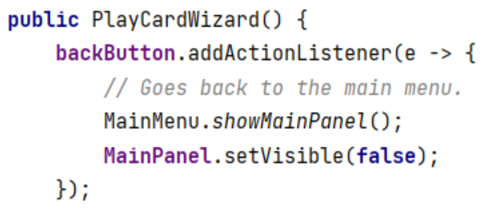


```
import javax.swing.*;
import java.io.IOException;

3 usages
public class PlayCardWizard {
    /*
    GUI Components
     */
    5 usages
    private JPanel MainPanel;
    1 usage
    private JTabbedPane tabbedPane1;
    2 usages
    private JButton backButton;
    5 usages
    private JTextField cardholderNametextfield;
    2 usages
    private JButton createButton;
    2 usages
    private JButton cancelButton;
    8 usages
    private JTextField initialDeposittextfield;
    3 usages
    private JTextField serialNumbertextfield;
    6 usages
    private JTextField deposittextfield;
    2 usages
    private JButton clearButton;
    2 usages
    private JButton depositButton;
    3 usages
    private JTextArea totaltextarea2;
    2 usages
    private JTextArea balancetextarea;
    3 usages
    private JTextArea totaltextarea1;
    2 usages
    private JButton refreshButton;
    /*
    Standard Messages
     */
    private final String emptyCredentialsMsg = "Empty credentials.";
```



```java
private final String invalidCredentialsMsg = "Invalid credentials.";
1 usage
private final String creationSuccessMsg = "Creation successful.";
/*
The cash register starts with an initial balance of zero.
 */
3 usages
private double netBalance = 0;
```

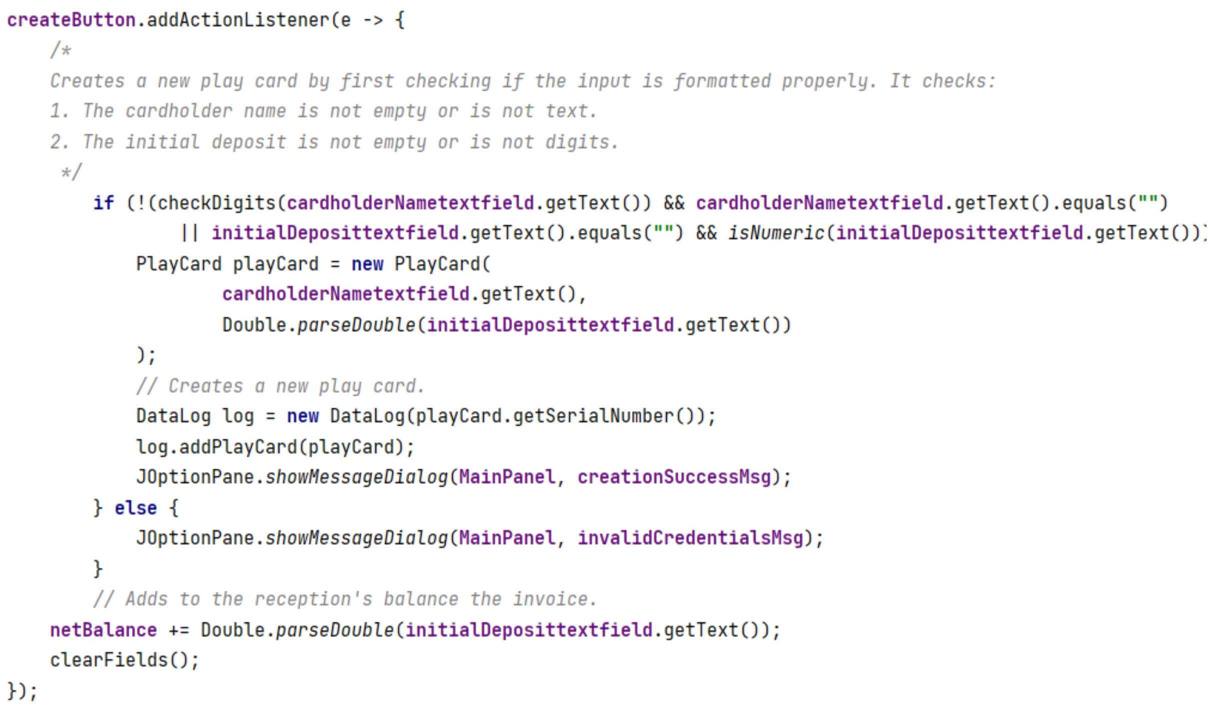
Instance variables.



```java
public PlayCardWizard() {
    backButton.addActionListener(e -> {
        // Goes back to the main menu.
        MainMenu.showMainPanel();
        MainPanel.setVisible(false);
    });
```

Opens the main menu when the user clicks “Back”.

(1) Creating a new play card



```java
createButton.addActionListener(e -> {
    /*
    Creates a new play card by first checking if the input is formatted properly. It checks:
    1. The cardholder name is not empty or is not text.
    2. The initial deposit is not empty or is not digits.
     */
        if (!(checkDigits(cardholderNametextfield.getText()) && cardholderNametextfield.getText().equals("")
                || initialDeposittextfield.getText().equals("") && isNumeric(initialDeposittextfield.getText()))
            PlayCard playCard = new PlayCard(
                    cardholderNametextfield.getText(),
                    Double.parseDouble(initialDeposittextfield.getText())
            );
            // Creates a new play card.
            DataLog log = new DataLog(playCard.getSerialNumber());
            log.addPlayCard(playCard);
            JOptionPane.showMessageDialog(MainPanel, creationSuccessMsg);
        } else {
            JOptionPane.showMessageDialog(MainPanel, invalidCredentialsMsg);
        }
        // Adds to the reception's balance the invoice.
    netBalance += Double.parseDouble(initialDeposittextfield.getText());
    clearFields();
});
```

Pressing “Create” button checks if the inputs are in appropriate format to proceed with the instantiation of the new PlayCard object. The DataLog class creates for the newly created card a .txt file having the card serial number as its file name and includes in its contents the card’s CVV, name, and balance. The program adds the initial deposit to the PlayCardWizard’s balance.

```
cancelButton.addActionListener(e -> clearFields());
initialDeposittextfield.addActionListener(e -> totaltextarea1.setText("$" + initialDeposittextfield.getText())
```

(2) Depositing into a PlayCard

```
depositextfield.addActionListener(e -> totaltextarea2.setText("$" + deposittextfield.getText()));
depositButton.addActionListener(e -> {
    DataLog log = new DataLog(serialNumbertextfield.getText());
    PlayCard playCard;
    try {
        playCard = log.readPlayCardFromFile();
        playCard.deposit(Double.parseDouble(deposittextfield.getText()));
    } catch (IOException ex) {
        throw new RuntimeException(ex);
    }
    netBalance += Double.parseDouble(deposittextfield.getText());
    clearFields();
});
```

Clicking the “Deposit” button **parses** from the .txt file of the specified Serial Number a playCard object. PlayCardWizard adds the deposit to the cash register (instance variable netBalance) and to the card’s balance by invoking playCard.deposit(). This method calls DataLog.overwriteBalance(), changing the value in the .txt file.

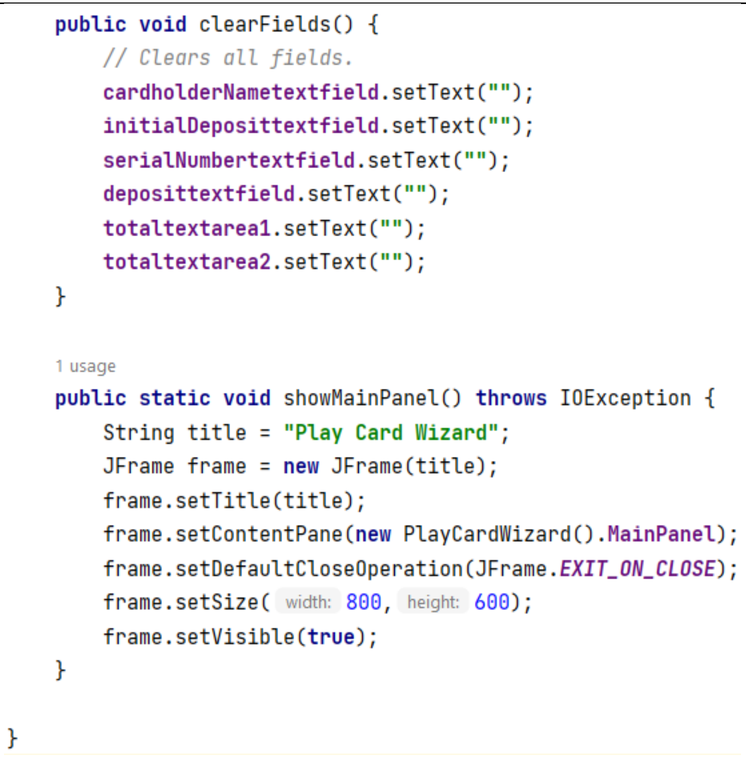
(3) Checking the cash register balance

```
refreshButton.addActionListener(e -> balancetextarea.setText("$" + netBalance));
```

Refreshing outputs the cash register’s balance, i.e. the sum of all the deposits made during the day.

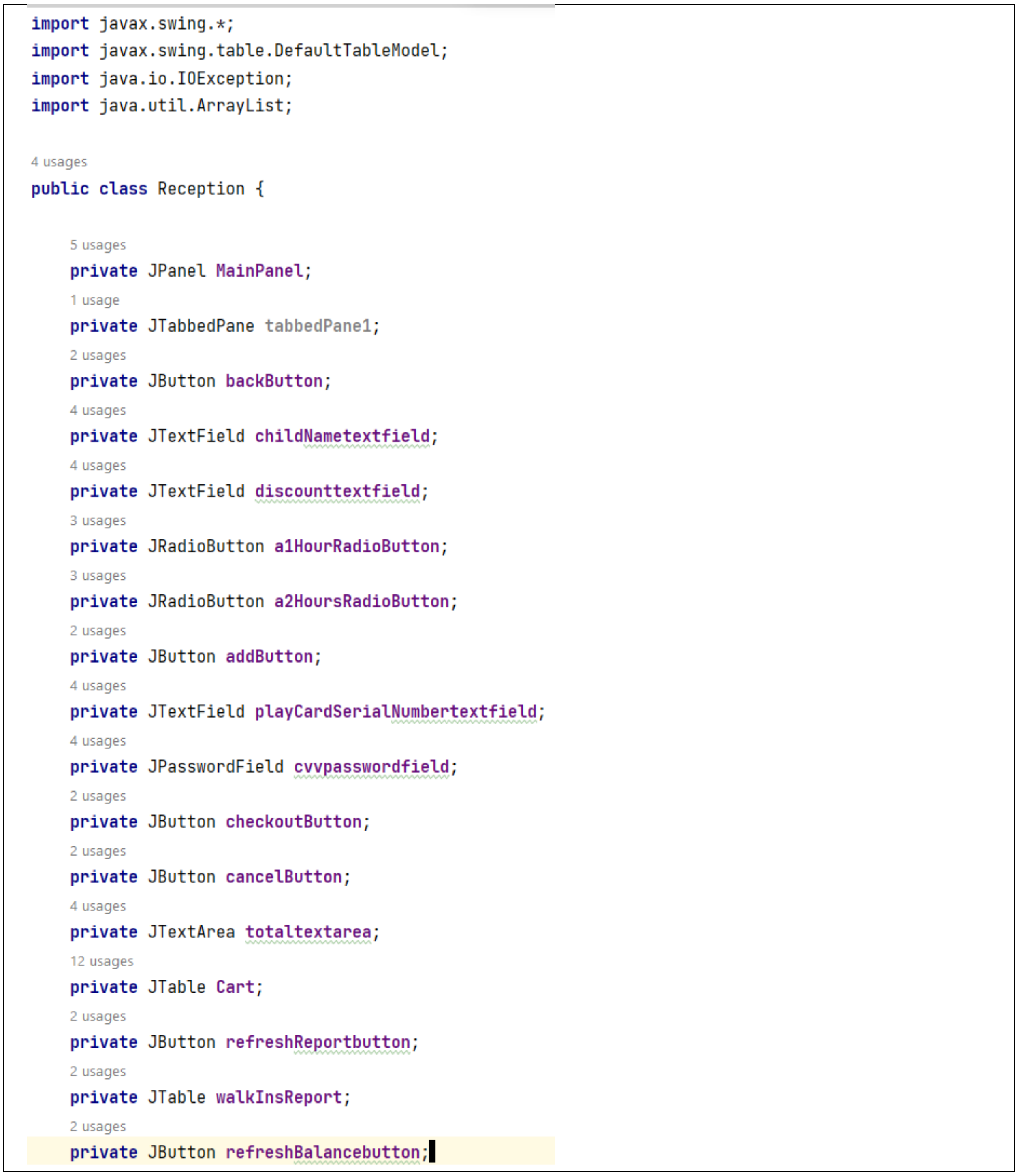
```
1 usage
public static boolean isNumeric(String str) {
    // Checks if all the characters of a string are numeric.
    try {
        Double.parseDouble(str);
        return true;
    } catch(NumberFormatException e){
        return false;
    }
}
```

Used when the user clicks the “Create” button to check if the value entered in the deposit is strictly numerical. If it is not, **an exception is thrown.**

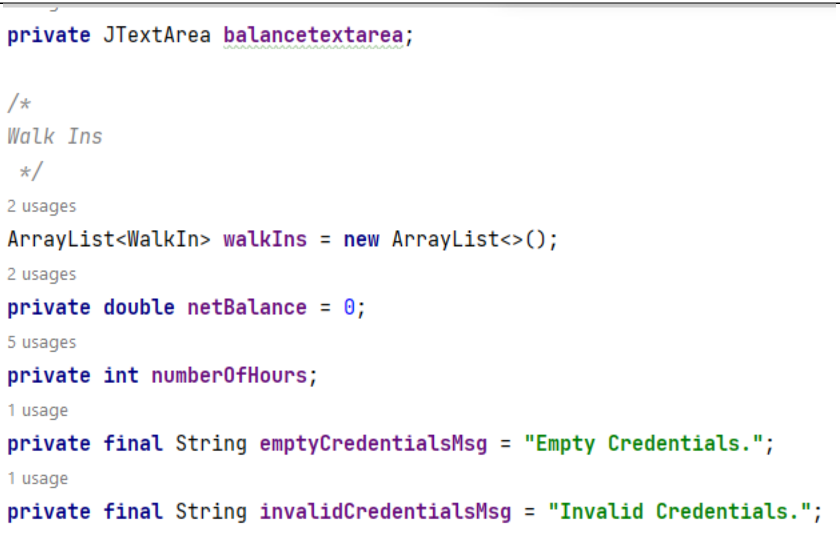


```
    public void clearFields() {
        // Clears all fields.
        cardholderNametextfield.setText("");
        initialDeposittextfield.setText("");
        serialNumbertextfield.setText("");
        deposittextfield.setText("");
        totaltextarea1.setText("");
        totaltextarea2.setText("");
    }

    1 usage
    public static void showMainPanel() throws IOException {
        String title = "Play Card Wizard";
        JFrame frame = new JFrame(title);
        frame.setTitle(title);
        frame.setContentPane(new PlayCardWizard().MainPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize( width: 800, height: 600);
        frame.setVisible(true);
    }

}
```

II.1.3. Reception

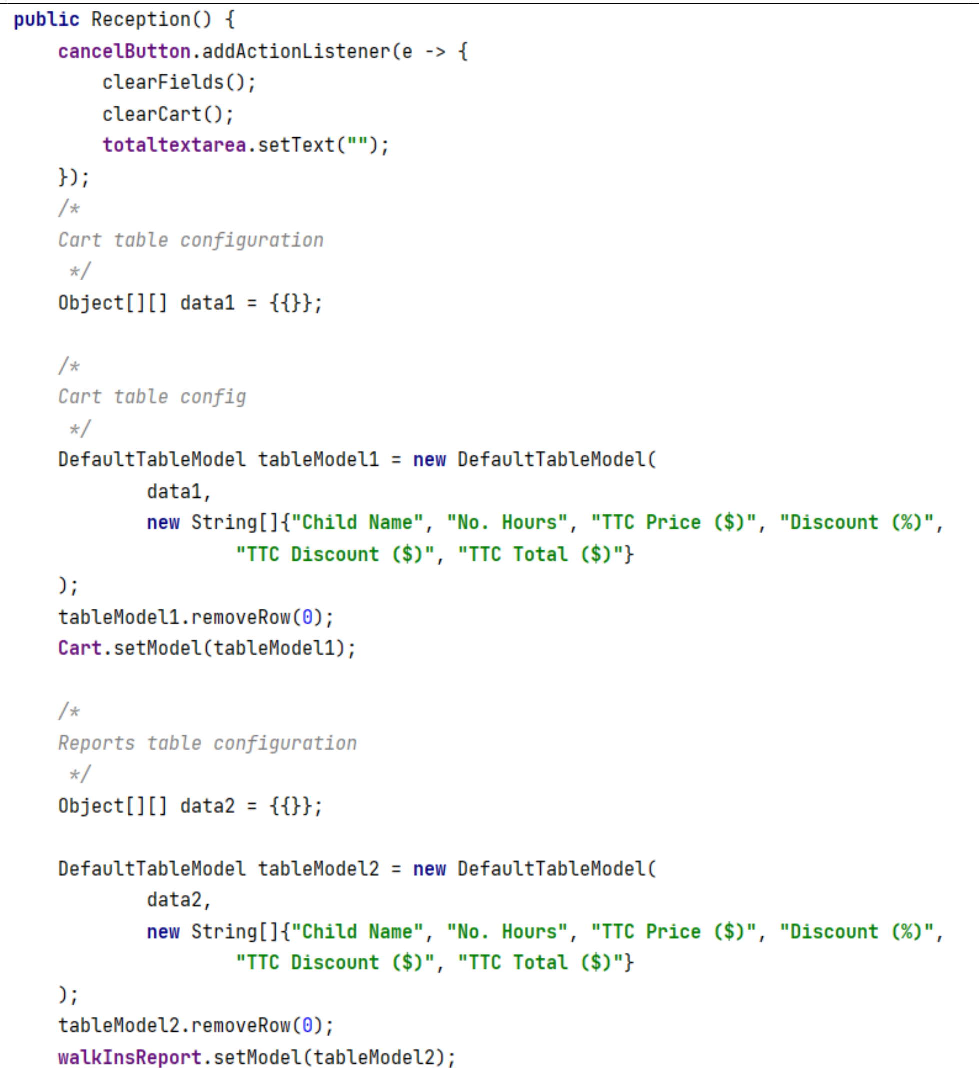


```java
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.io.IOException;
import java.util.ArrayList;

4 usages
public class Reception {

    5 usages
    private JPanel MainPanel;
    1 usage
    private JTabbedPane tabbedPane1;
    2 usages
    private JButton backButton;
    4 usages
    private JTextField childNametextfield;
    4 usages
    private JTextField discounttextfield;
    3 usages
    private JRadioButton a1HourRadioButton;
    3 usages
    private JRadioButton a2HoursRadioButton;
    2 usages
    private JButton addButton;
    4 usages
    private JTextField playCardSerialNumbertextfield;
    4 usages
    private JPasswordField cvvpasswordfield;
    2 usages
    private JButton checkoutButton;
    2 usages
    private JButton cancelButton;
    4 usages
    private JTextArea totaltextarea;
    12 usages
    private JTable Cart;
    2 usages
    private JButton refreshReportbutton;
    2 usages
    private JTable walkInsReport;
    2 usages
    private JButton refreshBalancebutton;
```



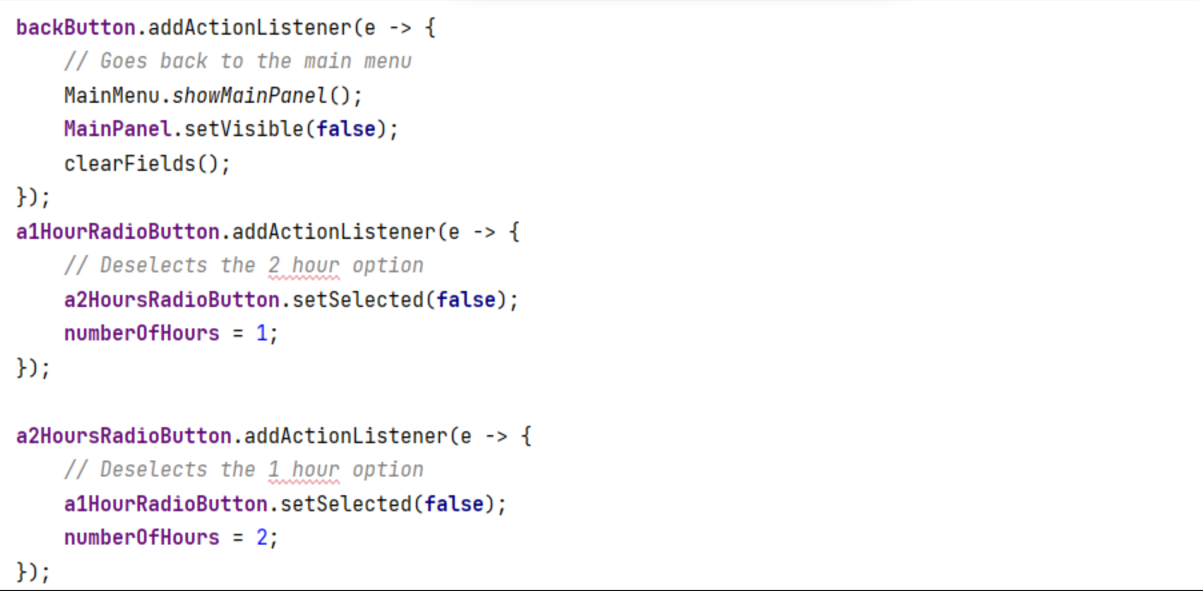
```
private JTextArea balancetextarea;

/*
Walk Ins
 */
2 usages
ArrayList<WalkIn> walkIns = new ArrayList<>();
2 usages
private double netBalance = 0;
5 usages
private int numberOfHours;
1 usage
private final String emptyCredentialsMsg = "Empty Credentials.";
1 usage
private final String invalidCredentialsMsg = "Invalid Credentials.";
```

Instance variables. The arraylist walkIns<> stores all the walk-ins registered during the day so that it can be used for the report.



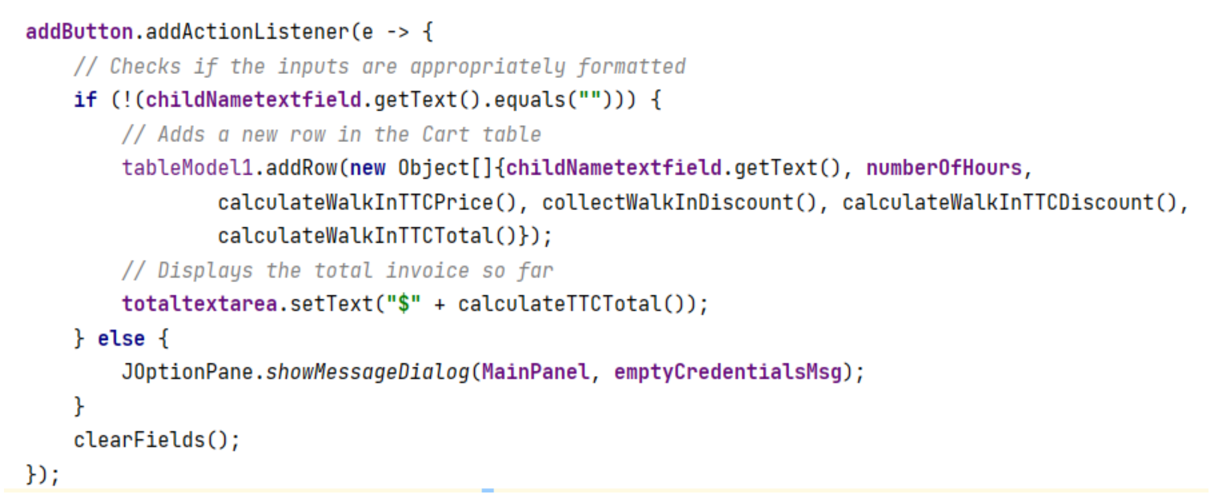
```java
public Reception() {
    cancelButton.addActionListener(e -> {
        clearFields();
        clearCart();
        totaltextarea.setText("");
    });
    /*
    Cart table configuration
     */
    Object[][] data1 = {{}};

    /*
    Cart table config
     */
    DefaultTableModel tableModel1 = new DefaultTableModel(
            data1,
            new String[]{"Child Name", "No. Hours", "TTC Price ($)", "Discount (%)",
                    "TTC Discount ($)", "TTC Total ($)"}
    );
    tableModel1.removeRow(0);
    Cart.setModel(tableModel1);

    /*
    Reports table configuration
     */
    Object[][] data2 = {{}};

    DefaultTableModel tableModel2 = new DefaultTableModel(
            data2,
            new String[]{"Child Name", "No. Hours", "TTC Price ($)", "Discount (%)",
                    "TTC Discount ($)", "TTC Total ($)"}
    );
    tableModel2.removeRow(0);
    walkInsReport.setModel(tableModel2);
```

The two 2D arrays are for the tables Cart and walkInsReport.

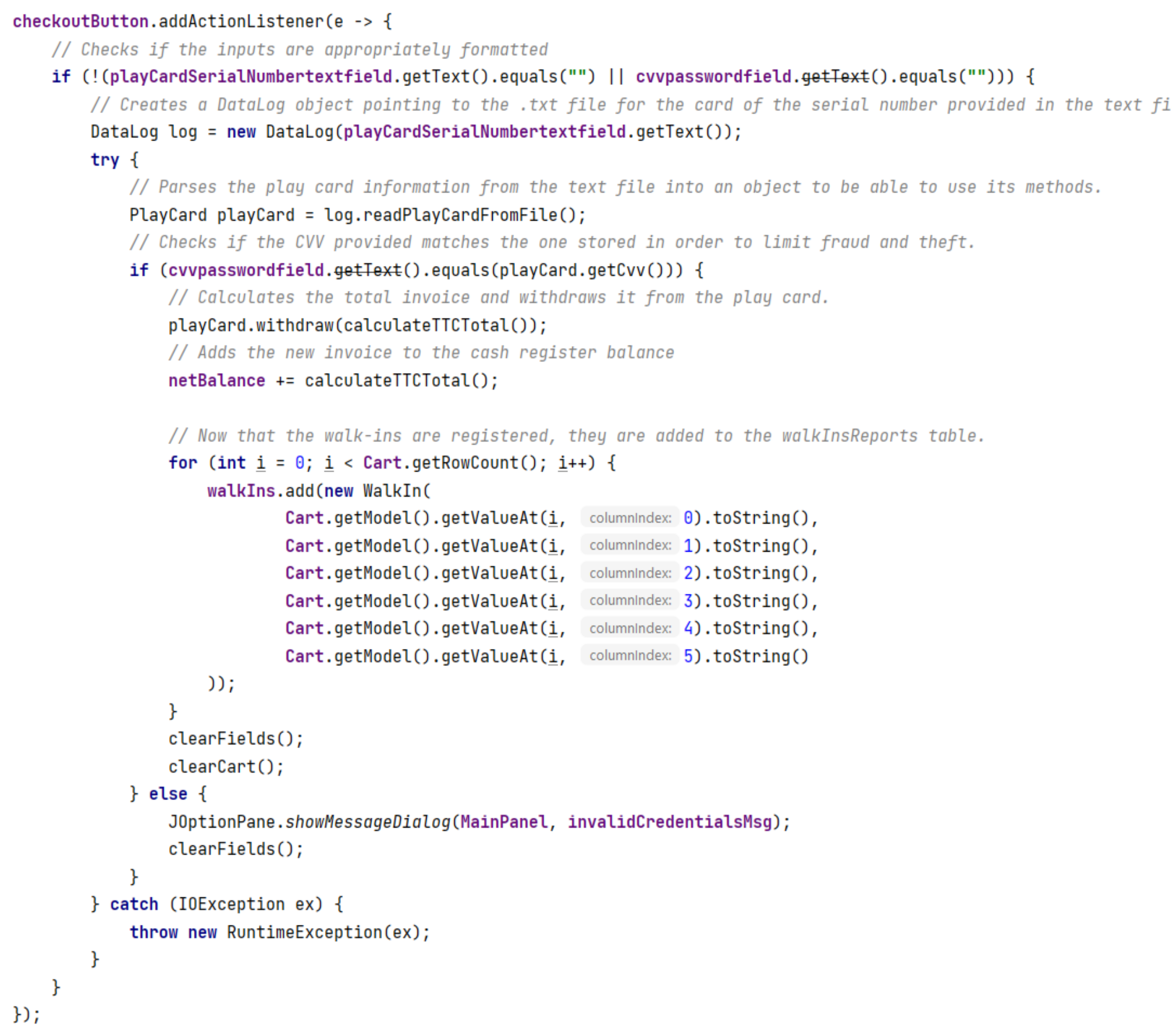


```java
backButton.addActionListener(e -> {
    // Goes back to the main menu
    MainMenu.showMainPanel();
    MainPanel.setVisible(false);
    clearFields();
});
a1HourRadioButton.addActionListener(e -> {
    // Deselects the 2 hour option
    a2HoursRadioButton.setSelected(false);
    numberOfHours = 1;
});

a2HoursRadioButton.addActionListener(e -> {
    // Deselects the 1 hour option
    a1HourRadioButton.setSelected(false);
    numberOfHours = 2;
});
```

(1) Register a new walk-in and checkout



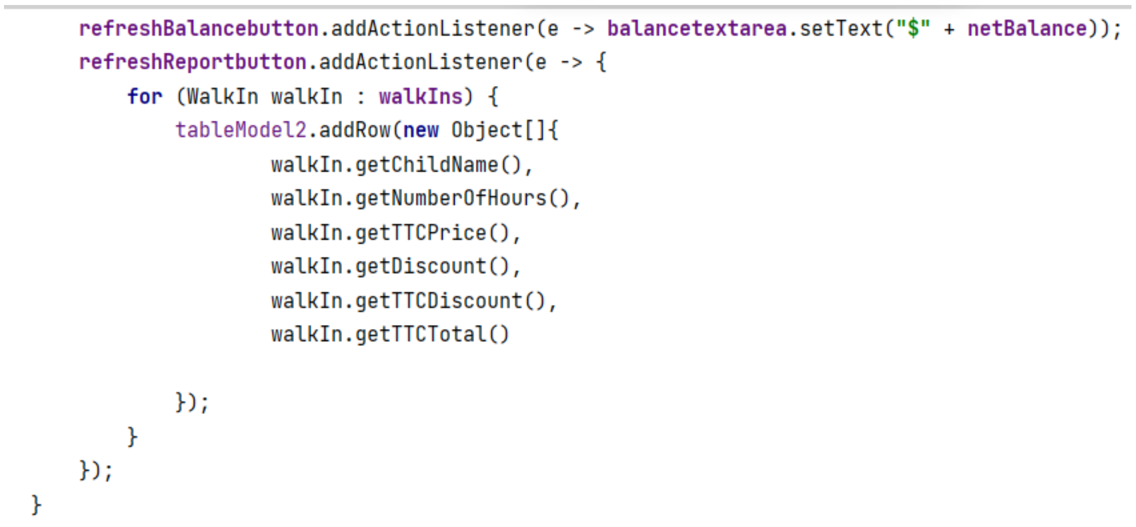
```java
addButton.addActionListener(e -> {
    // Checks if the inputs are appropriately formatted
    if (!(childNametextfield.getText().equals(""))) {
        // Adds a new row in the Cart table
        tableModel1.addRow(new Object[]{childNametextfield.getText(), numberOfHours,
                calculateWalkInTTCPrice(), collectWalkInDiscount(), calculateWalkInTTCDiscount(),
                calculateWalkInTTCTotal()});
        // Displays the total invoice so far
        totaltextarea.setText("$" + calculateTTCTotal());
    } else {
        JOptionPane.showMessageDialog(MainPanel, emptyCredentialsMsg);
    }
    clearFields();
});
```



```java
checkoutButton.addActionListener(e -> {
    // Checks if the inputs are appropriately formatted
    if (!(playCardSerialNumbertextfield.getText().equals("") || cvvpasswordfield.~~getText~~().equals(""))) {
        // Creates a DataLog object pointing to the .txt file for the card of the serial number provided in the text fi
        DataLog log = new DataLog(playCardSerialNumbertextfield.getText());
        try {
            // Parses the play card information from the text file into an object to be able to use its methods.
            PlayCard playCard = log.readPlayCardFromFile();
            // Checks if the CVV provided matches the one stored in order to limit fraud and theft.
            if (cvvpasswordfield.~~getText~~().equals(playCard.getCvv())) {
                // Calculates the total invoice and withdraws it from the play card.
                playCard.withdraw(calculateTTCTotal());
                // Adds the new invoice to the cash register balance
                netBalance += calculateTTCTotal();

                // Now that the walk-ins are registered, they are added to the walkInsReports table.
                for (int i = 0; i < Cart.getRowCount(); i++) {
                    walkIns.add(new WalkIn(
                            Cart.getModel().getValueAt(i,  columnIndex: 0).toString(),
                            Cart.getModel().getValueAt(i,  columnIndex: 1).toString(),
                            Cart.getModel().getValueAt(i,  columnIndex: 2).toString(),
                            Cart.getModel().getValueAt(i,  columnIndex: 3).toString(),
                            Cart.getModel().getValueAt(i,  columnIndex: 4).toString(),
                            Cart.getModel().getValueAt(i,  columnIndex: 5).toString()
                    ));
                }
                clearFields();
                clearCart();
            } else {
                JOptionPane.showMessageDialog(MainPanel, invalidCredentialsMsg);
                clearFields();
            }
        } catch (IOException ex) {
            throw new RuntimeException(ex);
        }
    }
});
```

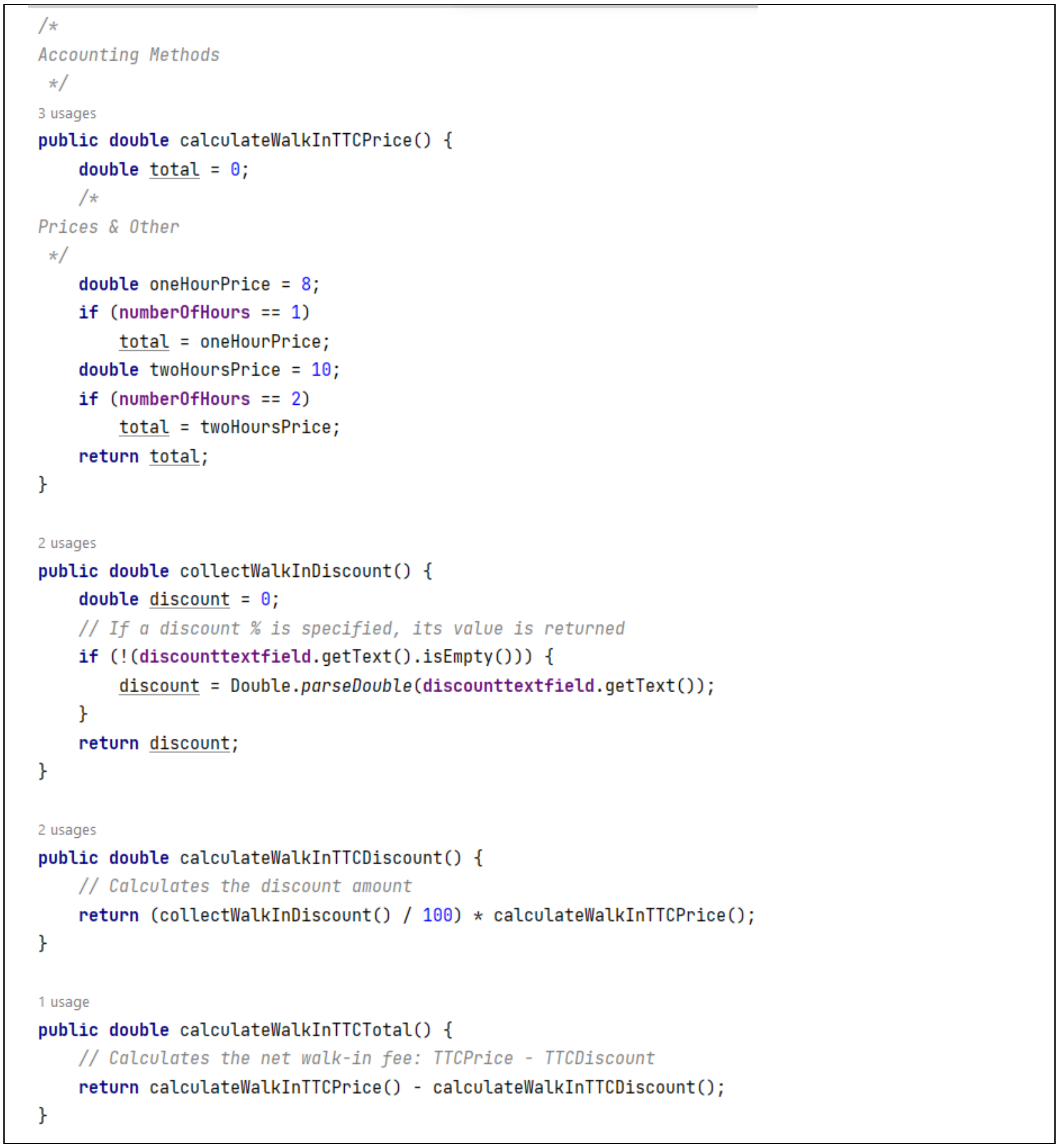
Clicking “Checkout” checks the input’s format and if the serial number provided does not exist, **an exception is thrown.** If the CVVs don’t match, the transaction does not go through.
Reception adds the total invoice to the cash register balance and the walk-ins registered to the walkInsReports.

(2) Checking the cash register balance and (3) Checking the walk-ins report

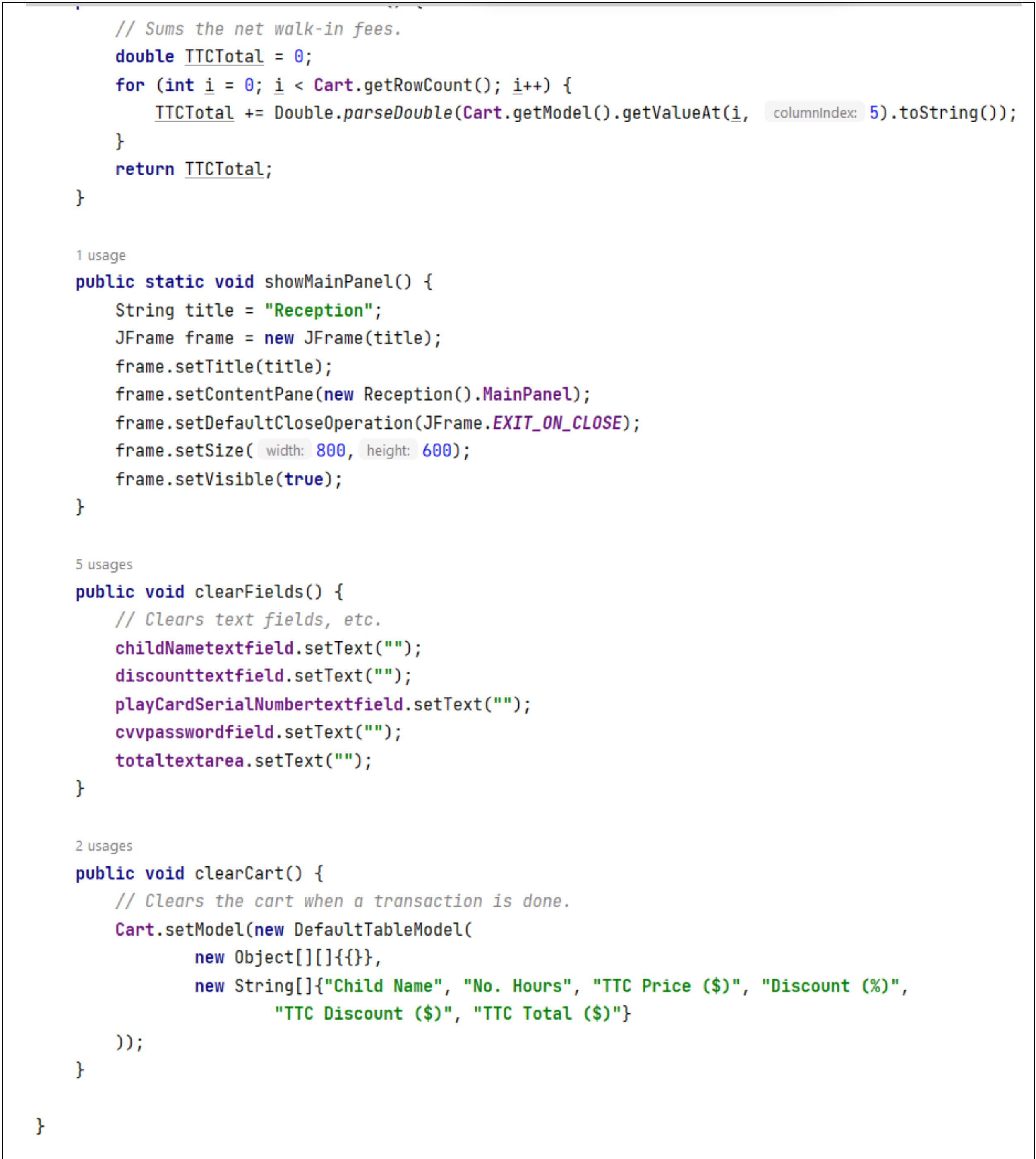


```
        refreshBalancebutton.addActionListener(e -> balancetextarea.setText("$" + netBalance));
        refreshReportbutton.addActionListener(e -> {
            for (WalkIn walkIn : walkIns) {
                tableModel2.addRow(new Object[]{
                        walkIn.getChildName(),
                        walkIn.getNumberOfHours(),
                        walkIn.getTTCPrice(),
                        walkIn.getDiscount(),
                        walkIn.getTTCDiscount(),
                        walkIn.getTTCTotal()

                });
            }
        });
    }
```

Refreshing the report traverses the walkIns Arraylist and displays its information in the walkInsReport JTable.

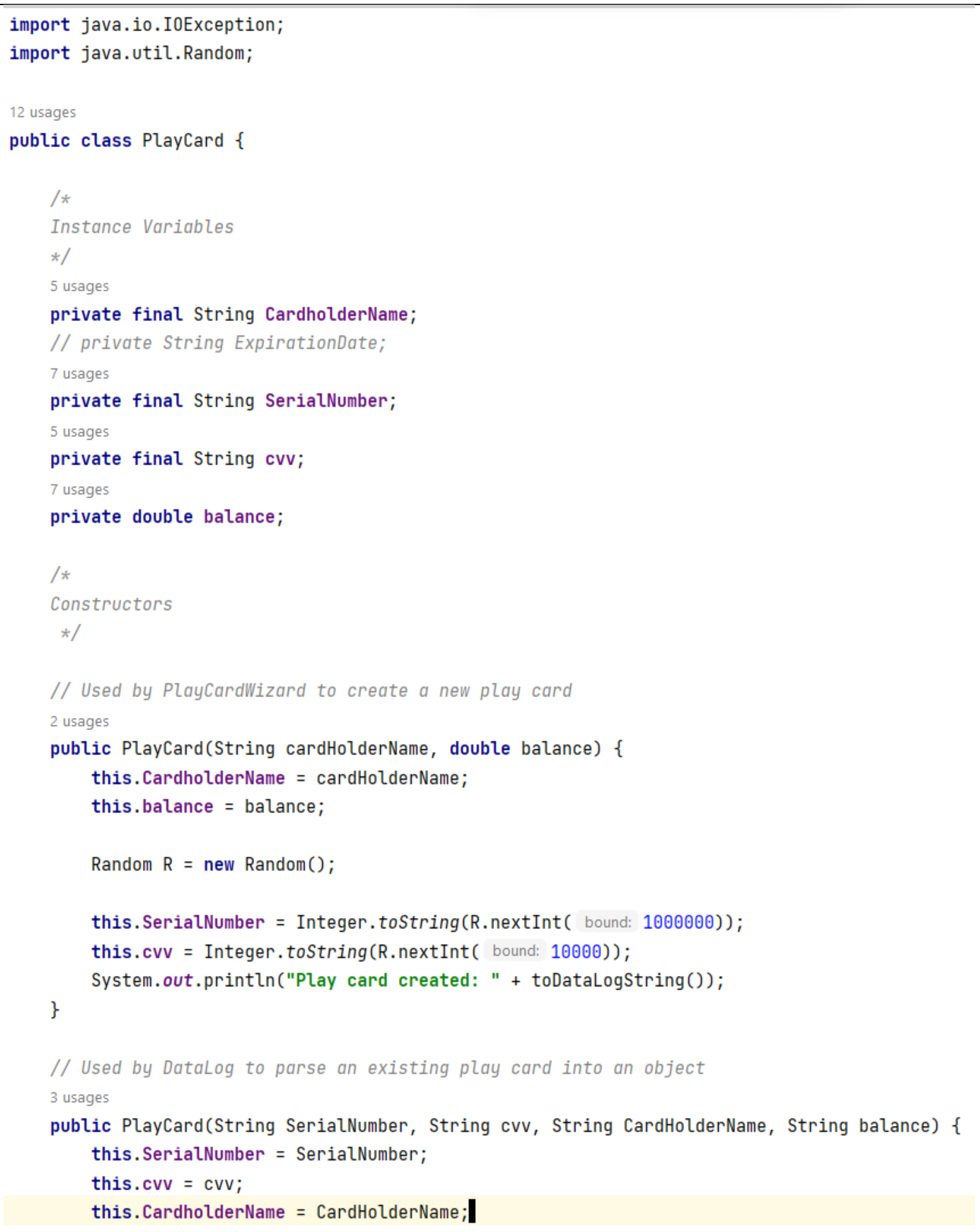


```java
/*
Accounting Methods
 */
3 usages
public double calculateWalkInTTCPrice() {
    double total = 0;
    /*
Prices & Other
 */
    double oneHourPrice = 8;
    if (numberOfHours == 1)
        total = oneHourPrice;
    double twoHoursPrice = 10;
    if (numberOfHours == 2)
        total = twoHoursPrice;
    return total;
}

2 usages
public double collectWalkInDiscount() {
    double discount = 0;
    // If a discount % is specified, its value is returned
    if (!(discounttextfield.getText().isEmpty())) {
        discount = Double.parseDouble(discounttextfield.getText());
    }
    return discount;
}

2 usages
public double calculateWalkInTTCDiscount() {
    // Calculates the discount amount
    return (collectWalkInDiscount() / 100) * calculateWalkInTTCPrice();
}

1 usage
public double calculateWalkInTTCTotal() {
    // Calculates the net walk-in fee: TTCPrice - TTCDiscount
    return calculateWalkInTTCPrice() - calculateWalkInTTCDiscount();
}
```



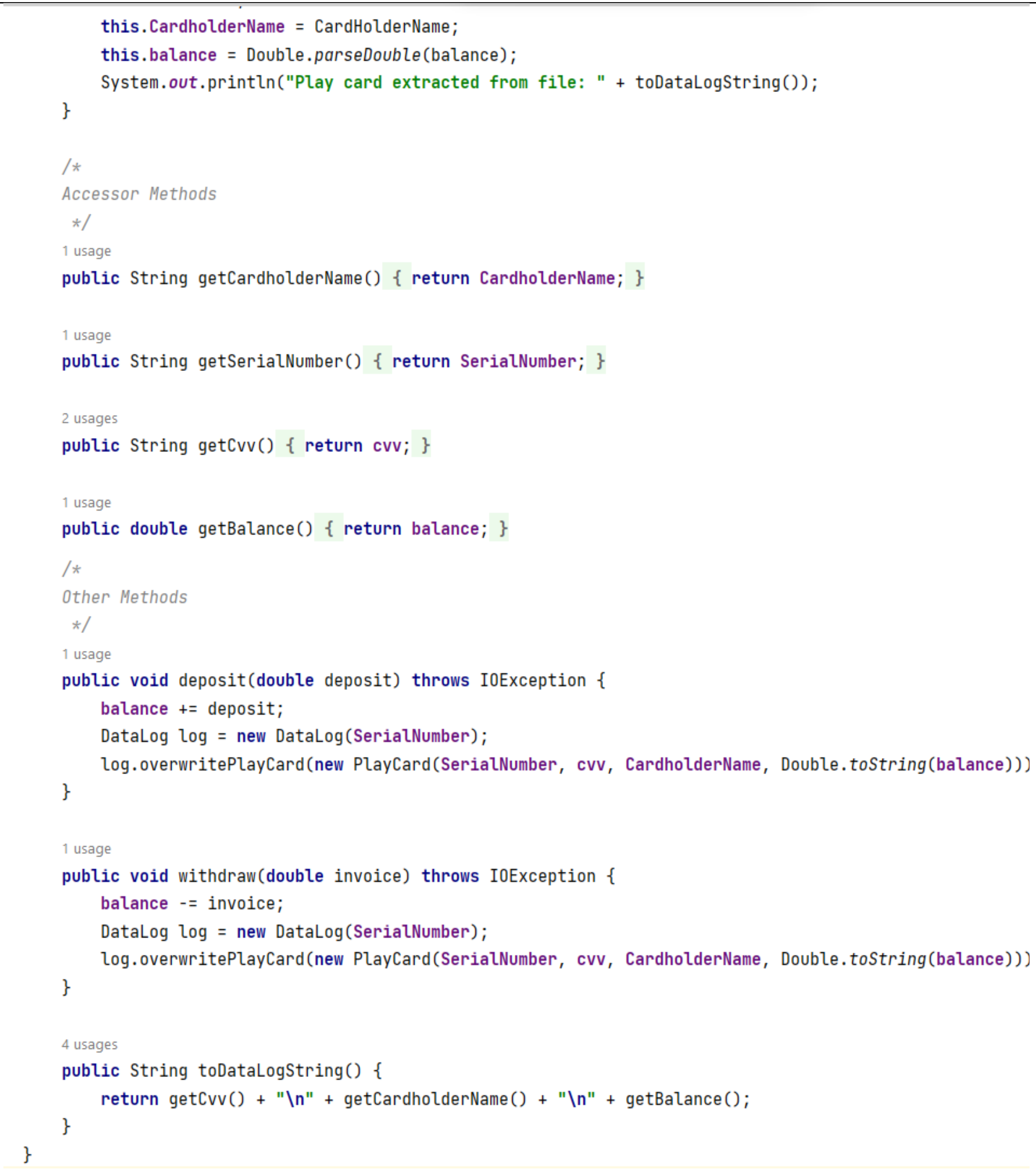
```java
        // Sums the net walk-in fees.
        double TTCTotal = 0;
        for (int i = 0; i < Cart.getRowCount(); i++) {
            TTCTotal += Double.parseDouble(Cart.getModel().getValueAt(i,  columnIndex: 5).toString());
        }
        return TTCTotal;
    }

    1 usage
    public static void showMainPanel() {
        String title = "Reception";
        JFrame frame = new JFrame(title);
        frame.setTitle(title);
        frame.setContentPane(new Reception().MainPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize( width: 800,  height: 600);
        frame.setVisible(true);
    }

    5 usages
    public void clearFields() {
        // Clears text fields, etc.
        childNametextfield.setText("");
        discounttextfield.setText("");
        playCardSerialNumbertextfield.setText("");
        cvvpasswordfield.setText("");
        totaltextarea.setText("");
    }

    2 usages
    public void clearCart() {
        // Clears the cart when a transaction is done.
        Cart.setModel(new DefaultTableModel(
                new Object[][]{{}},
                new String[]{"Child Name", "No. Hours", "TTC Price ($)", "Discount (%)",
                        "TTC Discount ($)", "TTC Total ($)"}
        ));
    }

}
```

II.1.4. Play Card



```
import java.io.IOException;
import java.util.Random;

12 usages
public class PlayCard {

    /*
    Instance Variables
    */
    5 usages
    private final String CardholderName;
    // private String ExpirationDate;
    7 usages
    private final String SerialNumber;
    5 usages
    private final String cvv;
    7 usages
    private double balance;

    /*
    Constructors
     */

    // Used by PlayCardWizard to create a new play card
    2 usages
    public PlayCard(String cardHolderName, double balance) {
        this.CardholderName = cardHolderName;
        this.balance = balance;

        Random R = new Random();

        this.SerialNumber = Integer.toString(R.nextInt( bound: 1000000));
        this.cvv = Integer.toString(R.nextInt( bound: 10000));
        System.out.println("Play card created: " + toDataLogString());
    }

    // Used by DataLog to parse an existing play card into an object
    3 usages
    public PlayCard(String SerialNumber, String cvv, String CardHolderName, String balance) {
        this.SerialNumber = SerialNumber;
        this.cvv = cvv;
        this.CardholderName = CardHolderName;
```

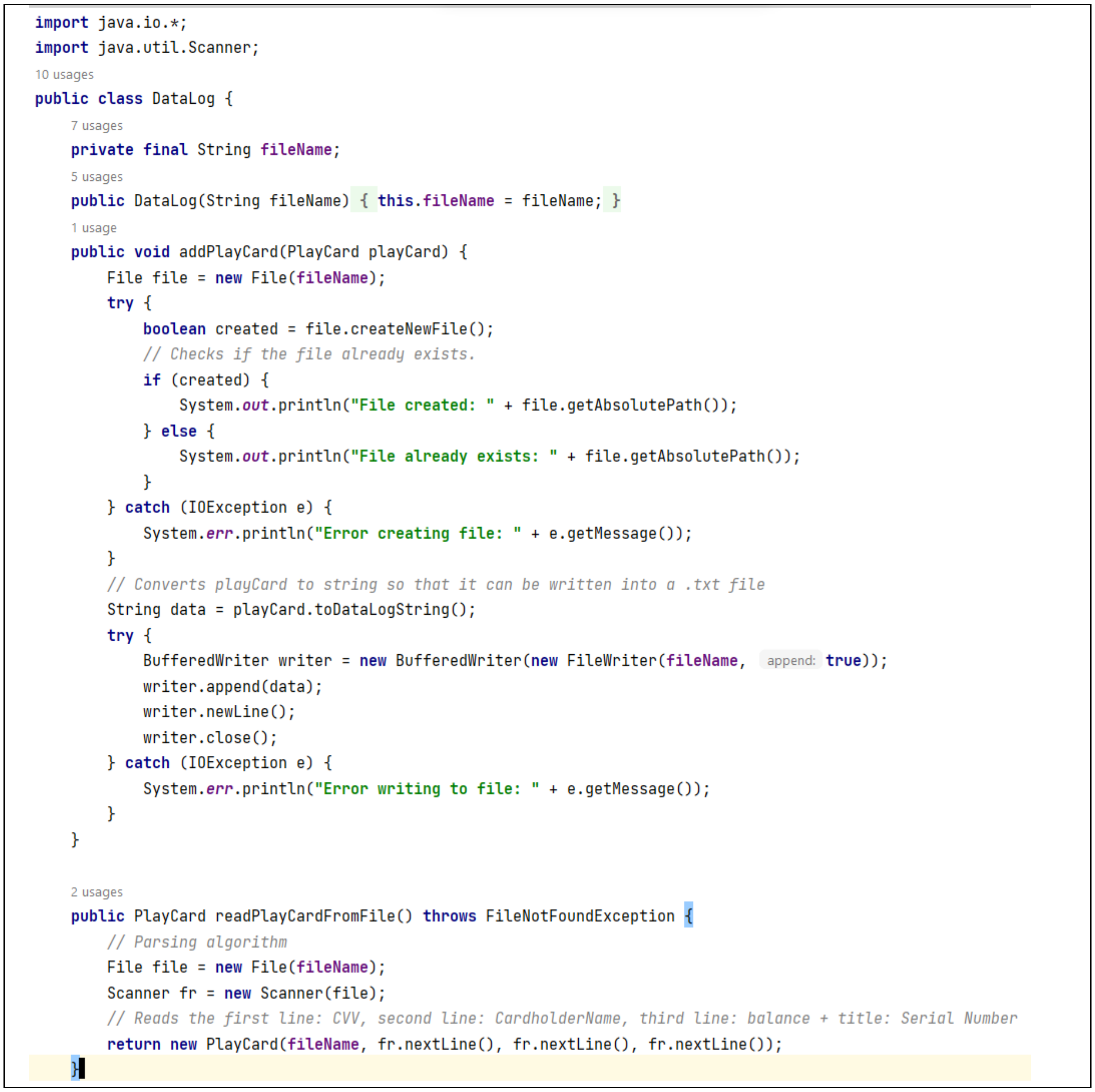
PlayCardWizard uses the first constructor to generate a new playCard object (uses a random number generator to create a new serial number), whereas DataLog uses the second constructor to parse an already-existing from a file into a playCard object.



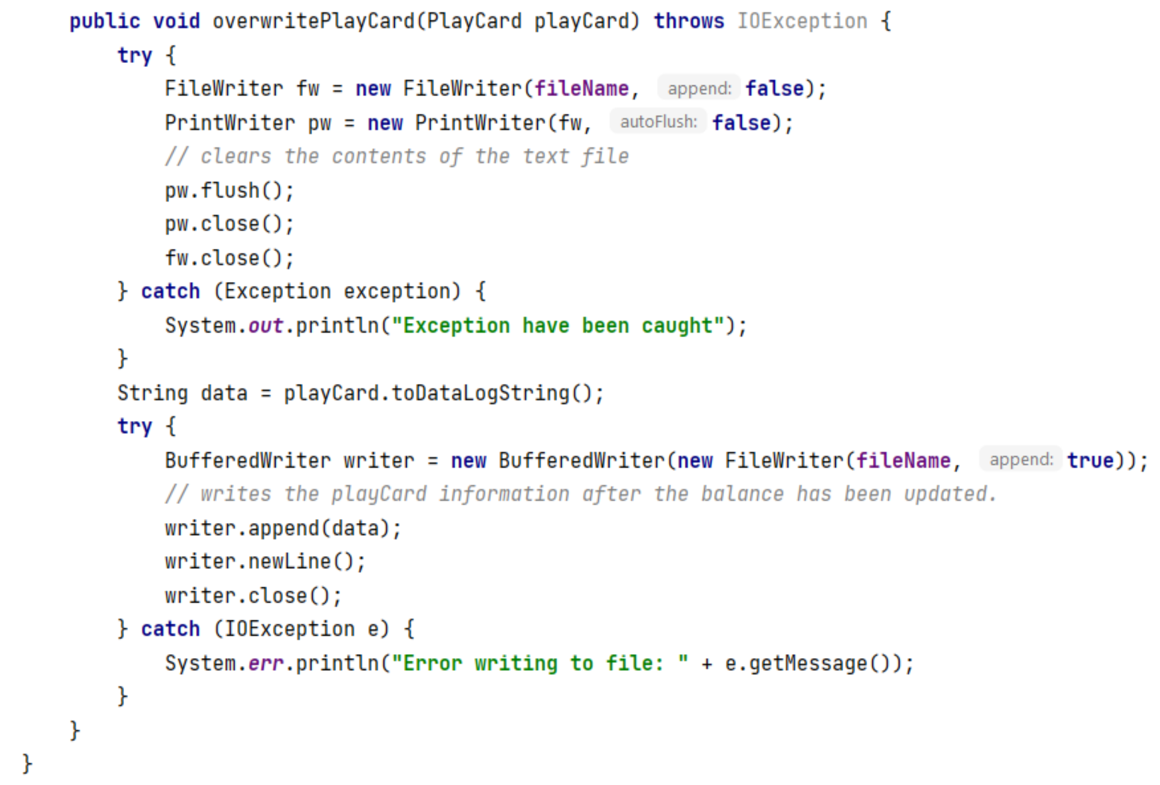
```java
        this.CardholderName = CardHolderName;
        this.balance = Double.parseDouble(balance);
        System.out.println("Play card extracted from file: " + toDataLogString());
    }

    /*
    Accessor Methods
     */
    1 usage
    public String getCardholderName() { return CardholderName; }

    1 usage
    public String getSerialNumber() { return SerialNumber; }

    2 usages
    public String getCvv() { return cvv; }

    1 usage
    public double getBalance() { return balance; }

    /*
    Other Methods
     */
    1 usage
    public void deposit(double deposit) throws IOException {
        balance += deposit;
        DataLog log = new DataLog(SerialNumber);
        log.overwritePlayCard(new PlayCard(SerialNumber, cvv, CardholderName, Double.toString(balance)))
    }

    1 usage
    public void withdraw(double invoice) throws IOException {
        balance -= invoice;
        DataLog log = new DataLog(SerialNumber);
        log.overwritePlayCard(new PlayCard(SerialNumber, cvv, CardholderName, Double.toString(balance)))
    }

    4 usages
    public String toDataLogString() {
        return getCvv() + "\n" + getCardholderName() + "\n" + getBalance();
    }
}
```

deposit() and withdraw() carry out the same operation (with the exception of adding / subtracting) which is calling DataLog.overwritePlayCard() to update the card's balance on the .txt file accordingly.

II.1.5. DataLog



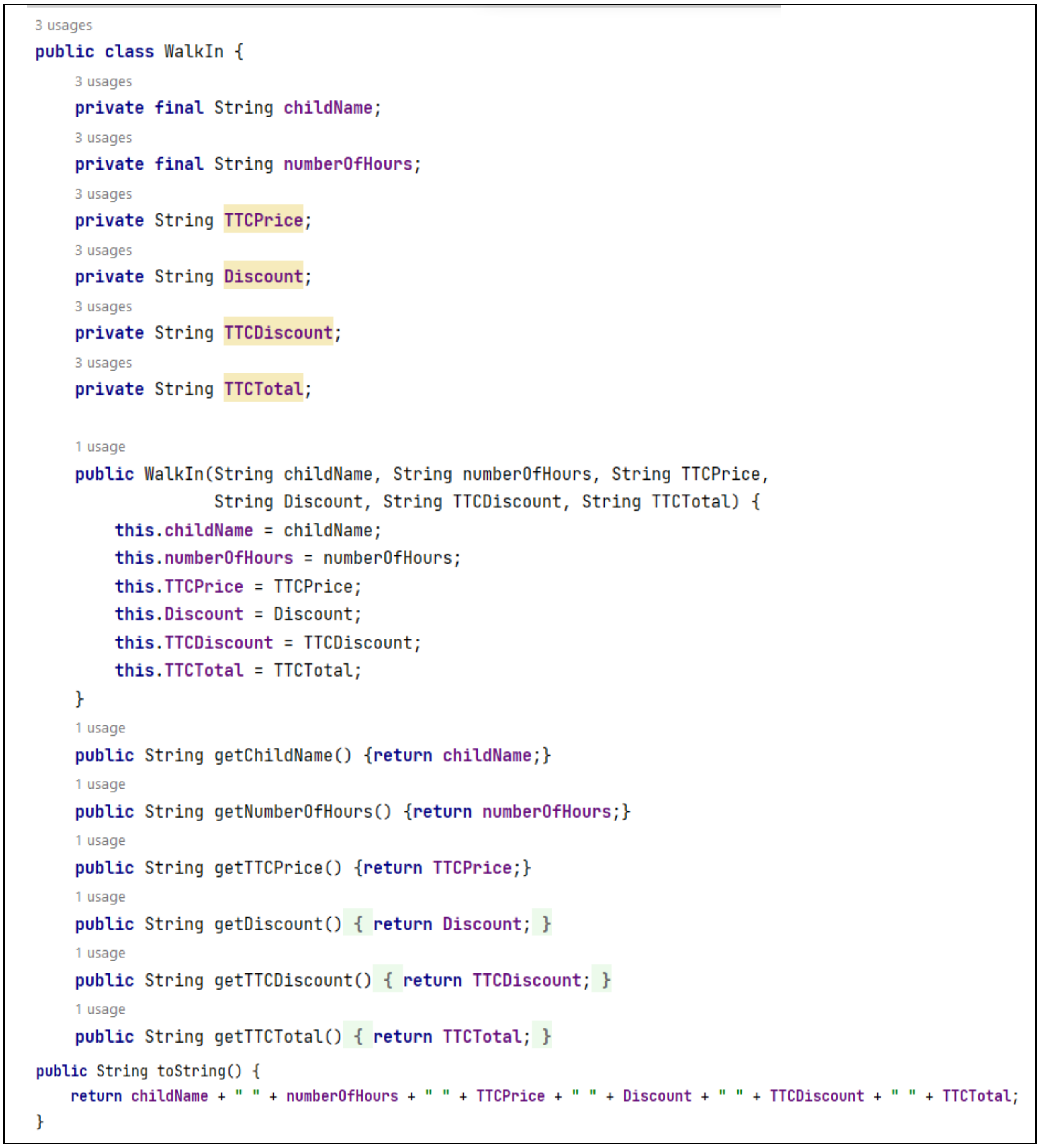
```java
import java.io.*;
import java.util.Scanner;
10 usages
public class DataLog {
    7 usages
    private final String fileName;
    5 usages
    public DataLog(String fileName) { this.fileName = fileName; }
    1 usage
    public void addPlayCard(PlayCard playCard) {
        File file = new File(fileName);
        try {
            boolean created = file.createNewFile();
            // Checks if the file already exists.
            if (created) {
                System.out.println("File created: " + file.getAbsolutePath());
            } else {
                System.out.println("File already exists: " + file.getAbsolutePath());
            }
        } catch (IOException e) {
            System.err.println("Error creating file: " + e.getMessage());
        }
        // Converts playCard to string so that it can be written into a .txt file
        String data = playCard.toDataLogString();
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter(fileName,  append: true));
            writer.append(data);
            writer.newLine();
            writer.close();
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }
    }

    2 usages
    public PlayCard readPlayCardFromFile() throws FileNotFoundException {
        // Parsing algorithm
        File file = new File(fileName);
        Scanner fr = new Scanner(file);
        // Reads the first line: CVV, second line: CardholderName, third line: balance + title: Serial Number
        return new PlayCard(fileName, fr.nextLine(), fr.nextLine(), fr.nextLine());
    }
```



```
        public void overwritePlayCard(PlayCard playCard) throws IOException {
            try {
                FileWriter fw = new FileWriter(fileName,  append: false);
                PrintWriter pw = new PrintWriter(fw,  autoFlush: false);
                // clears the contents of the text file
                pw.flush();
                pw.close();
                fw.close();
            } catch (Exception exception) {
                System.out.println("Exception have been caught");
            }
            String data = playCard.toDataLogString();
            try {
                BufferedWriter writer = new BufferedWriter(new FileWriter(fileName,  append: true));
                // writes the playCard information after the balance has been updated.
                writer.append(data);
                writer.newLine();
                writer.close();
            } catch (IOException e) {
                System.err.println("Error writing to file: " + e.getMessage());
            }
        }
    }
```

overwritePlayCard() clears the content of the .txt file specified and rewrites the new playCard information after the balance has been updated.

II.1.6. WalkIn



```
3 usages
public class WalkIn {
    3 usages
    private final String childName;
    3 usages
    private final String numberOfHours;
    3 usages
    private String TTCPrice;
    3 usages
    private String Discount;
    3 usages
    private String TTCDiscount;
    3 usages
    private String TTCTotal;

    1 usage
    public WalkIn(String childName, String numberOfHours, String TTCPrice,
                  String Discount, String TTCDiscount, String TTCTotal) {
        this.childName = childName;
        this.numberOfHours = numberOfHours;
        this.TTCPrice = TTCPrice;
        this.Discount = Discount;
        this.TTCDiscount = TTCDiscount;
        this.TTCTotal = TTCTotal;
    }
    1 usage
    public String getChildName() {return childName;}
    1 usage
    public String getNumberOfHours() {return numberOfHours;}
    1 usage
    public String getTTCPrice() {return TTCPrice;}
    1 usage
    public String getDiscount() { return Discount; }
    1 usage
    public String getTTCDiscount() { return TTCDiscount; }
    1 usage
    public String getTTCTotal() { return TTCTotal; }
public String toString() {
    return childName + " " + numberOfHours + " " + TTCPrice + " " + Discount + " " + TTCDiscount + " " + TTCTotal;
}
```

II.2. Call Diagram

To better illustrate how classes interact:

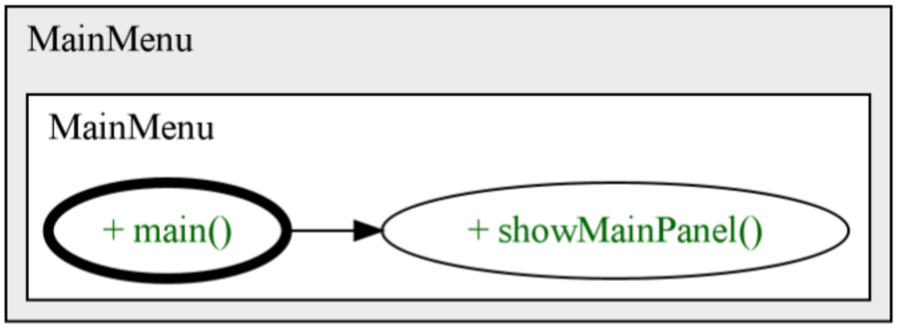


Figure 9 MainMenu.main() Calls

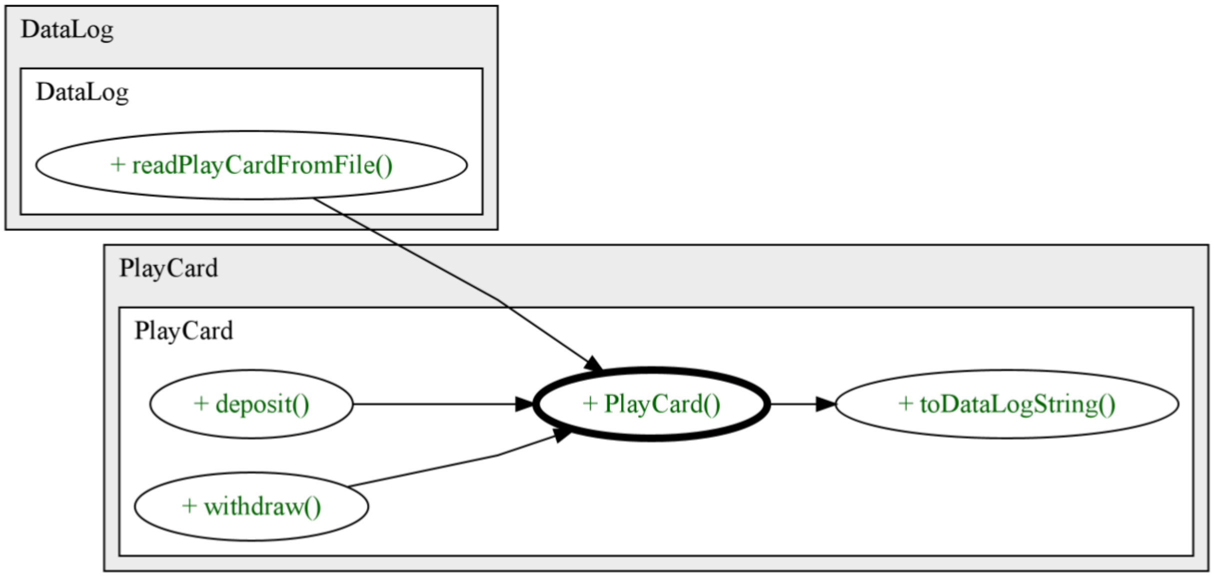


Figure 10 PlayCard() Calls

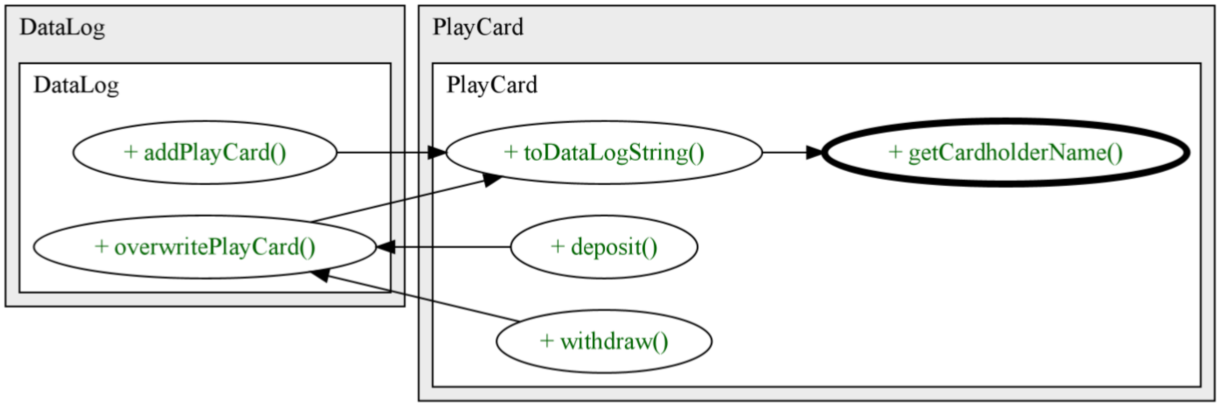


Figure 11 PlayCard.getCardholderName() Calls

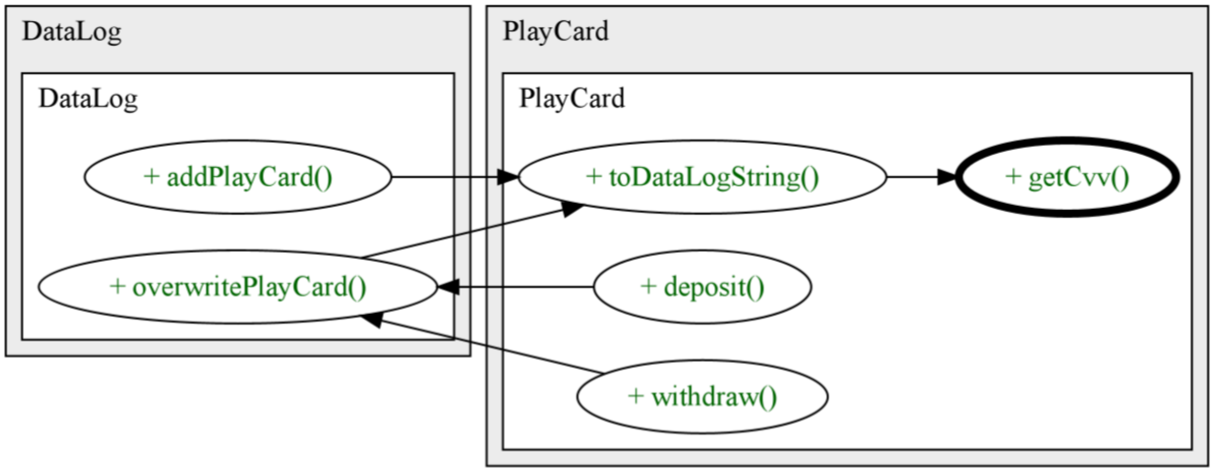


Figure 12 PlayCard.getCvv() Calls

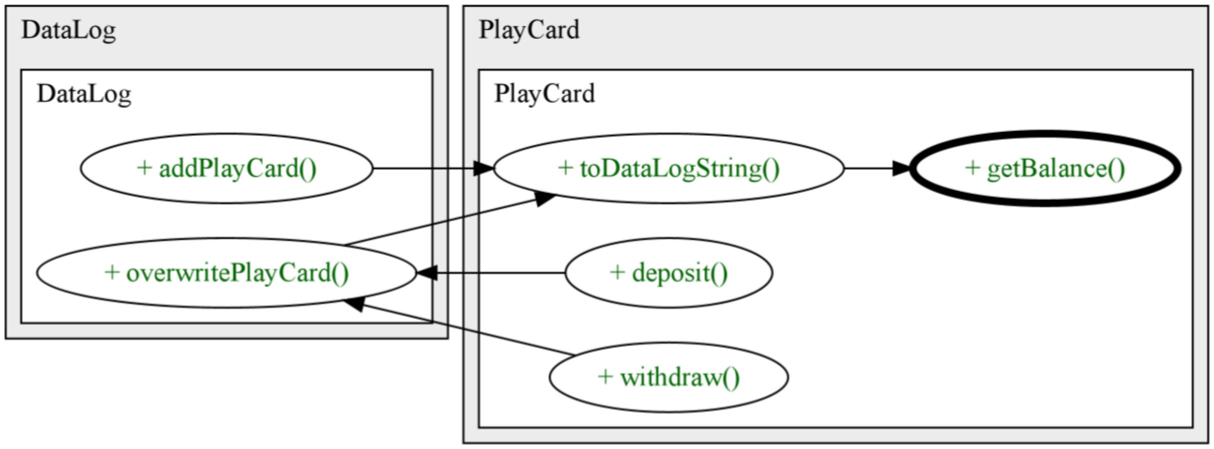


Figure 13 PlayCard.getBalance() Calls

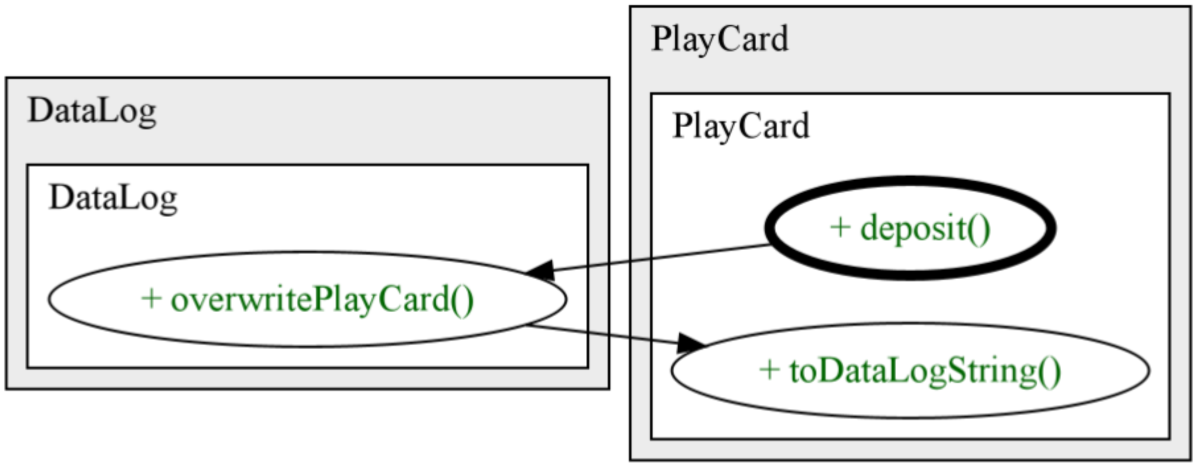


Figure 14 PlayCard.deposit() Calls

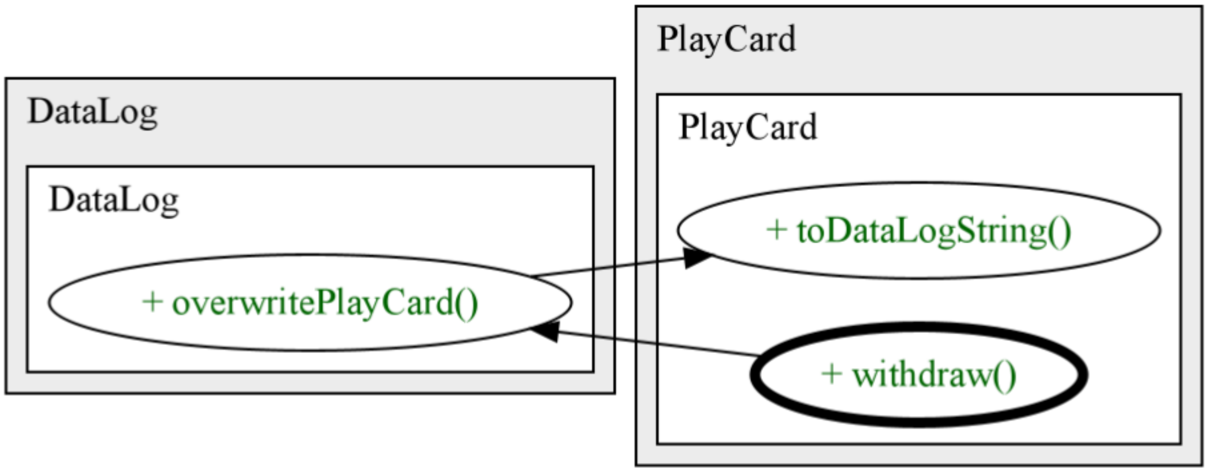


Figure 15 PlayCard.withdraw() Calls

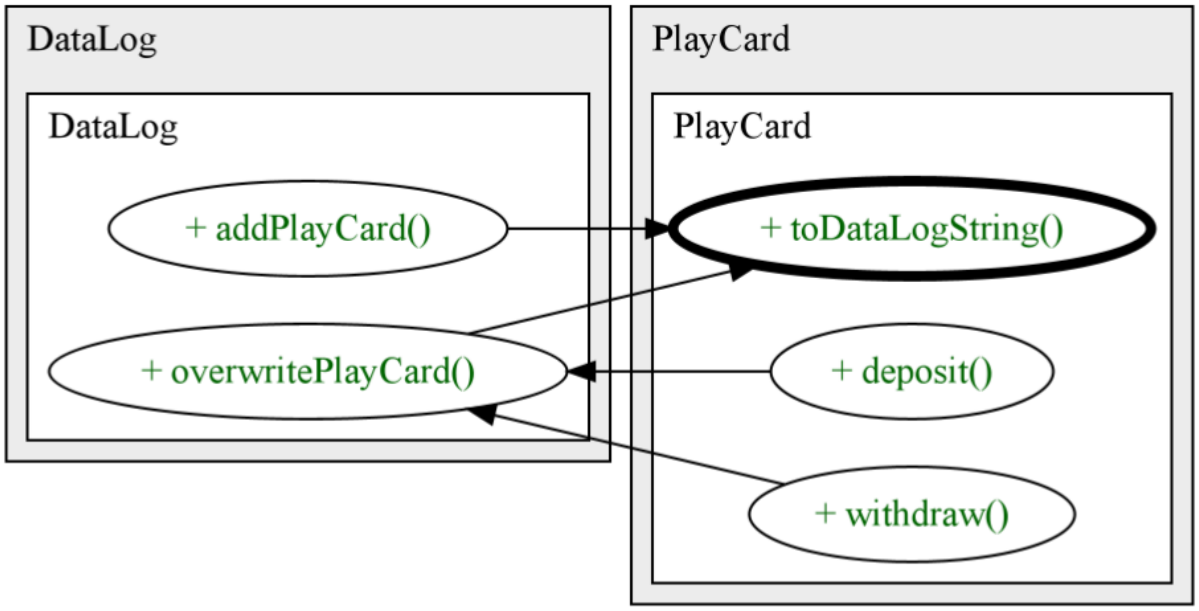


Figure 16 PlayCard.toDataLogString() Calls

*Note that all the methods of the WalkIn class are invoked internally.

II.3. Parsing

Parsing is the process of converting data between formats so that the program handling the data can comprehend it and operate on it. In this application, parsing is prominent in the DataLog class in the readPlayCardFromFile() method that requires converting from text to a PlayCard object. That is in order to benefit from the method of the PlayCard class, notable deposit() and withdraw() in order to be able to update the balance of the play card.

II.4. Exception Handling

Sometimes, the user inputs data whose format does not comply with the methods that handle said input and cause malfunction. To avoid this outcome, the candidate adds “throws Exception” when defining some methods and uses try / catch statements to assert boundaries at the level of the compiler. Throwing exceptions outside of the method keeps the application from crashing.