Course Submission Cover Sheet
Module: CC4001 Programming Engineering
Component no: 003
Weighting: 60% of module mark
Deadline: 1st of May 2024
Module Leader: Sandra Fernando   Student ID: SANDRO ZAKAIDZE

PLAGIARISM
You are reminded that there exist regulations concerning plagiarism. Extracts from these regulations are printed below. Please sign below to say that you have read and understand these extracts:

(Signature:) Sandro Zakaidze   Date: 16/04/2024

This header sheet should be attached to the work you submit. No work will be accepted without it.

GadgetShop Project Documentation

Table of Contents

1. Introduction

The GadgetShop project is a Java application that simulates a gadget store management system. It allows users to add gadgets (Mobile phones and MP3 players), manage their details, and perform operations such as making calls and downloading music via a graphical user interface (GUI). The application employs object-oriented programming principles and ensures an interactive user experience through various GUI components.

2. Class Overview

Gadget Class

Purpose: The `Gadget` class is the superclass that represents a generic gadget with attributes for model, price, weight, and size. It provides the basic structure and methods that are inherited by the `Mobile` and `MP3` subclasses.

Attributes:

- `model` (String): The model name of the gadget.
- `price` (double): The price of the gadget.
- `weight` (int): The weight of the gadget in grams.
- `size` (String): The dimensions of the gadget.

Methods:

- `Gadget(String model, double price, int weight, String size)`: Constructor to initialize the gadget attributes.
- `getModel()`: Returns the model of the gadget.
- `getPrice()`: Returns the price of the gadget.
- `getWeight()`: Returns the weight of the gadget.
- `getSize()`: Returns the size of the gadget.
- `display()`: Returns a formatted string with the gadget details.

Mobile Class

Purpose: The `Mobile` class extends `Gadget` to represent a mobile phone with additional functionality to manage calling credit and make phone calls.

Attributes:

- `callingCreditRemaining` (int): The remaining calling credit in minutes.

Methods:

- `Mobile(String model, double price, int weight, String size, int callingCreditRemaining)`: Constructor to initialize the mobile attributes.
- `getCredit()`: Returns the remaining calling credit.
- `insertCredit(int amount)`: Adds credit to the remaining calling credit if the amount is positive.
- `makePhoneCall(String phoneNumber, int durationOfCall)`: Makes a phone call if sufficient credit is available, otherwise prompts the user to add more credit.
- `display()`: Returns a formatted string with the mobile details, including the remaining calling credit.

MP3 Class

Purpose: The `MP3` class extends `Gadget` to represent an MP3 player with additional functionality to manage available memory and download music.

Attributes:

- `memory` (double): The available memory in megabytes.

Methods:

- `MP3(String model, double price, int weight, String size, double memory)`: Constructor to initialize the MP3 attributes.
- `getMemory()`: Returns the available memory.
- `downloadMusic(double amount)`: Downloads music if sufficient memory is available, otherwise prompts the user to free up memory.
- `display()`: Returns a formatted string with the MP3 details, including the available memory.

GadgetShop Class

Purpose: The `GadgetShop` class manages the GUI and the interaction between the user and the gadget store. It handles user inputs, displays gadget details, and performs operations such as adding gadgets, making calls, and downloading music.

Attributes:

- `gadgets` (ArrayList<Gadget>): A list to store all gadgets.
- Various GUI components (JFrame, JTextField, JButton): GUI elements for user interaction.

Methods:

- `GadgetShop()`: Constructor to initialize the gadget list and set up the GUI components.
- `setLookAndFeel()`: Sets the Nimbus look and feel for the GUI.
- `initializeFields()`: Initializes the text fields and buttons for the GUI.
- `setupFrame()`: Sets up the main frame and adds components to it.
- `addLabeledField(Container parent, GridBagConstraints constraints, String label, JTextField field, int row)`: Adds a labeled text field to the specified container.

- `actionPerformed(ActionEvent event)`: Handles button click events.
- `displayAllGadgets()`: Displays all gadgets in the list in a message dialog.
- `clearFields()`: Clears all input fields in the GUI.
- `readDouble(JTextField field, String fieldName)`: Reads a double value from a text field.
- `readInt(JTextField field, String fieldName)`: Reads an integer value from a text field.
- `readDisplayNumber()`: Reads and validates the display number from the text field.
- `main(String[] args)`: The main method to launch the GadgetShop application.

3. Method Descriptions

Gadget Class Methods

9. **Gadget(String model, double price, int weight, String size)**

   - **Description:** Constructor to initialize the gadget attributes.
   - **Parameters:**
     - `model`: The model name of the gadget.
     - `price`: The price of the gadget.
     - `weight`: The weight of the gadget.
     - `size`: The dimensions of the gadget.
   - **Returns:** None.

10. **getModel()**

   - **Description:** Returns the model of the gadget.
   - **Parameters:** None.
   - **Returns:** `String` - The model of the gadget.

11. **getPrice()**

   - **Description:** Returns the price of the gadget.
   - **Parameters:** None.
   - **Returns:** `double` - The price of the gadget.

12. **getWeight()**

   - **Description:** Returns the weight of the gadget.
   - **Parameters:** None.
   - **Returns:** `int` - The weight of the gadget.

13. **getSize()**

   - **Description:** Returns the size of the gadget.
   - **Parameters:** None.
   - **Returns:** `String` - The size of the gadget.

14. **display()**

   - **Description:** Returns a formatted string with the gadget details.
   - **Parameters:** None.
   - **Returns:** `String` - The formatted gadget details.

Mobile Class Methods

15. **Mobile(String model, double price, int weight, String size, int callingCreditRemaining)**

- **Description:** Constructor to initialize the mobile attributes.
- **Parameters:**
  - `model`: The model name of the mobile.
  - `price`: The price of the mobile.
  - `weight`: The weight of the mobile.
  - `size`: The dimensions of the mobile.
  - `callingCreditRemaining`: The remaining calling credit.
- **Returns:** None.

16. **getCredit()**

- **Description:** Returns the remaining calling credit.
- **Parameters:** None.
- **Returns:** `int` - The remaining calling credit.

17. **insertCredit(int amount)**

- **Description:** Adds credit to the remaining calling credit if the amount is positive.
- **Parameters:**
  - `amount`: The amount of credit to add.
- **Returns:** None.

18. **makePhoneCall(String phoneNumber, int durationOfCall)**

- **Description:** Makes a phone call if sufficient credit is available, otherwise prompts the user to add more credit.
- **Parameters:**
  - `phoneNumber`: The phone number to call.
  - `durationOfCall`: The duration of the call in minutes.
- **Returns:** None.

19. **display()**

- **Description:** Returns a formatted string with the mobile details, including the remaining calling credit.
- **Parameters:** None.
- **Returns:** `String` - The formatted mobile details.

MP3 Class Methods

20. **MP3(String model, double price, int weight, String size, double memory)**

- **Description:** Constructor to initialize the MP3 attributes.
- **Parameters:**
  - `model`: The model name of the MP3.
  - `price`: The price of the MP3.
  - `weight`: The weight of the MP3.
  - `size`: The dimensions of the MP3.
  - `memory`: The available memory.
- **Returns:** None.

21. getMemory()

- **Description:** Returns the available memory.
- **Parameters:** None.
- **Returns:** `double` - The available memory.

22. downloadMusic(double amount)

- **Description:** Downloads music if sufficient memory is available, otherwise prompts the user to free up memory.
- **Parameters:**
  - `amount`: The amount of memory required for the download.
- **Returns:** None.

23. display()

- **Description:** Returns a formatted string with the MP3 details, including the available memory.
- **Parameters:** None.
- **Returns:** `String` - The formatted MP3 details.

GadgetShop Class Methods

24. GadgetShop()

- **Description:** Constructor to initialize the gadget list and set up the GUI components.
- **Parameters:** None.
- **Returns:** None.

25. setLookAndFeel()

- **Description:** Sets the Nimbus look and feel for the GUI.
- **Parameters:** None.
- **Returns:** None.

26. initializeFields()

- **Description:** Initializes the text fields and buttons for the GUI.
- **Parameters:** None.
- **Returns:** None.

27. setupFrame()

- **Description:** Sets up the main frame and adds components to it.
- **Parameters:** None.
- **Returns:** None.

28. addLabeledField(Container parent, GridBagConstraints constraints, String label, JTextField field, int row)

- **Description:** Adds a labeled text field to the specified container.
- **Parameters:**
  - `parent`: The container to which the label and field are added.
  - `constraints`: The layout constraints for the component.
  - `label`: The text for the label.
  - `field`: The text field to add.

        - `row`: The row position for the component.
- **Returns:** None.

29. actionPerformed(ActionEvent event)

- **Description:** Handles button click events.
- **Parameters:**
    - `event`: The ActionEvent triggered by button clicks.
- **Returns:** None.

30. displayAllGadgets()

- **Description:** Displays all gadgets in the list in a message dialog.
- **Parameters:** None.
- **Returns:** None.

31. clearFields()

- **Description:** Clears all input fields in the GUI.
- **Parameters:** None.
- **Returns:** None.

32. readDouble(JTextField field, String fieldName)

- **Description:** Reads a double value from a text field.
- **Parameters:**
    - `field`: The text field to read from.
    - `fieldName`: The name of the field (for error messages).
- **Returns:** `double` - The double value from the field.
- **Throws:** IllegalArgumentException if the input is not a valid double.

33. readInt(JTextField field, String fieldName)

- **Description:** Reads an integer value from a text field.
- **Parameters:**
    - `field`: The text field to read from.
    - `fieldName`: The name of the field (for error messages).
- **Returns:** `int` - The integer value from the field.
- **Throws:** IllegalArgumentException if the input is not a valid integer.

34. readDisplayNumber()

- **Description:** Reads and validates the display number from the text field.
- **Parameters:** None.
- **Returns:** `int` - The display number, or -1 if invalid.

35. main(String[] args)

- **Description:** The main method to launch the GadgetShop application.
- **Parameters:**
    - `args`: Command line arguments.
- **Returns:** None.

4. GUI Design and Functionality

The GUI of the GadgetShop application is designed using Java Swing components. The main frame contains various text fields for user input, labeled appropriately, and buttons for performing actions such as adding gadgets, clearing fields, displaying all gadgets, making calls, and downloading music.

- **Add Mobile Button:** Adds a new mobile gadget to the list using the values entered in the text fields.
- **Add MP3 Button:** Adds a new MP3 gadget to the list using the values entered in the text fields.
- **Clear Fields Button:** Clears all the text fields in the GUI.
- **Display All Gadgets Button:** Displays all gadgets in the list in a message dialog.
- **Make Call Button:** Makes a call using the selected mobile gadget if sufficient credit is available.
- **Download Music Button:** Downloads music to the selected MP3 gadget if sufficient memory is available.

5. Error Handling

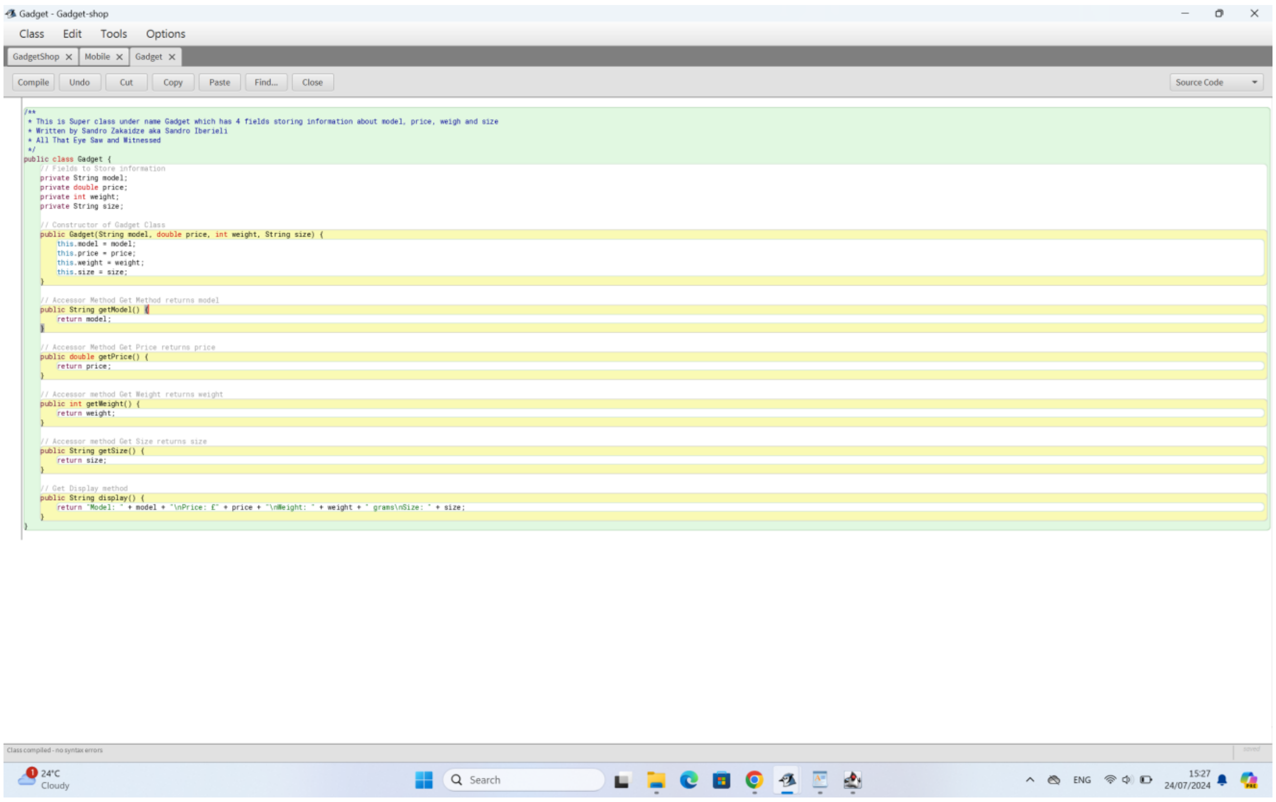
The GadgetShop application employs robust error handling to ensure smooth user interaction. Input values are validated before processing, and appropriate error messages are displayed in message dialogs for invalid inputs. The application also checks for valid gadget indices and ensures the selected gadget type matches the operation being performed (e.g., making a call with a mobile gadget).

6. Conclusion

The GadgetShop project demonstrates the application of object-oriented programming principles and GUI design using Java Swing. The application allows users to manage gadgets efficiently, with a user-friendly interface and robust error handling. The project highlights the importance of clear method definitions, proper input validation, and informative user feedback.

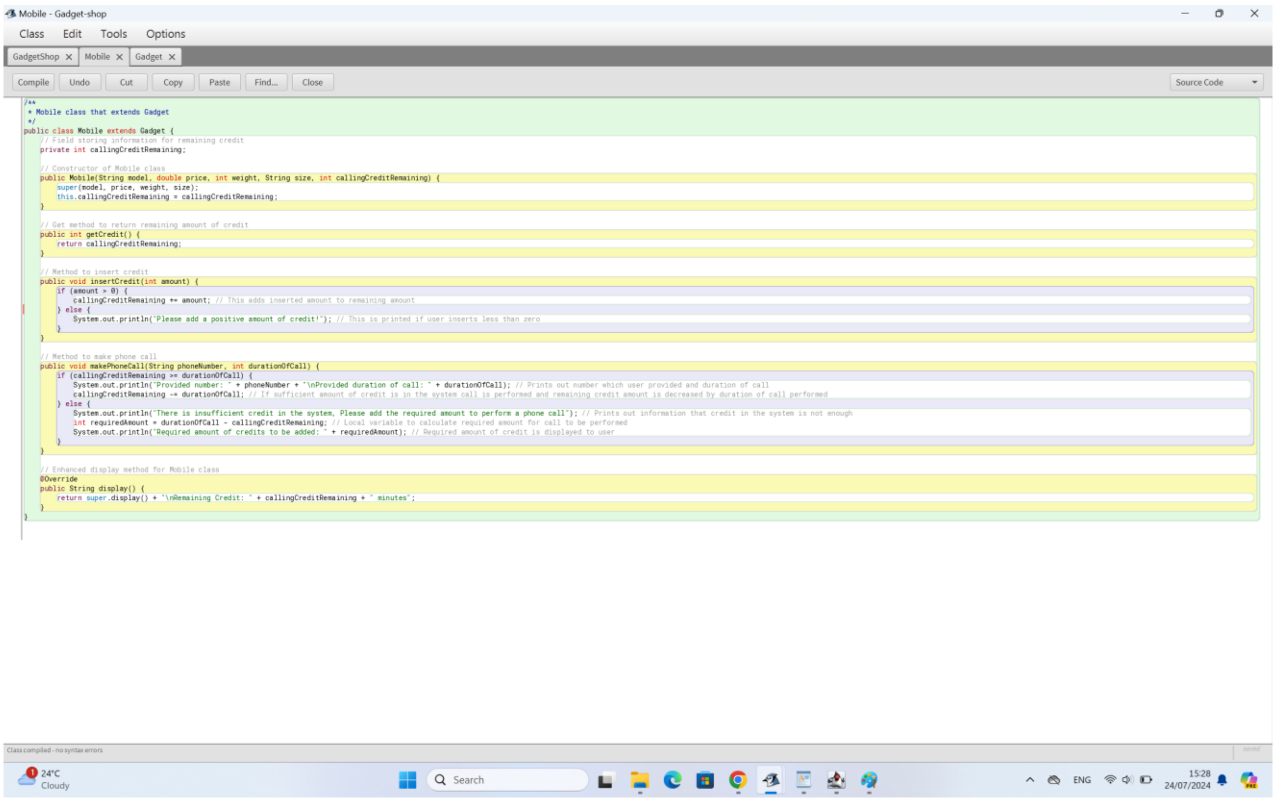
7. APENDIX

Gadget CLASS:



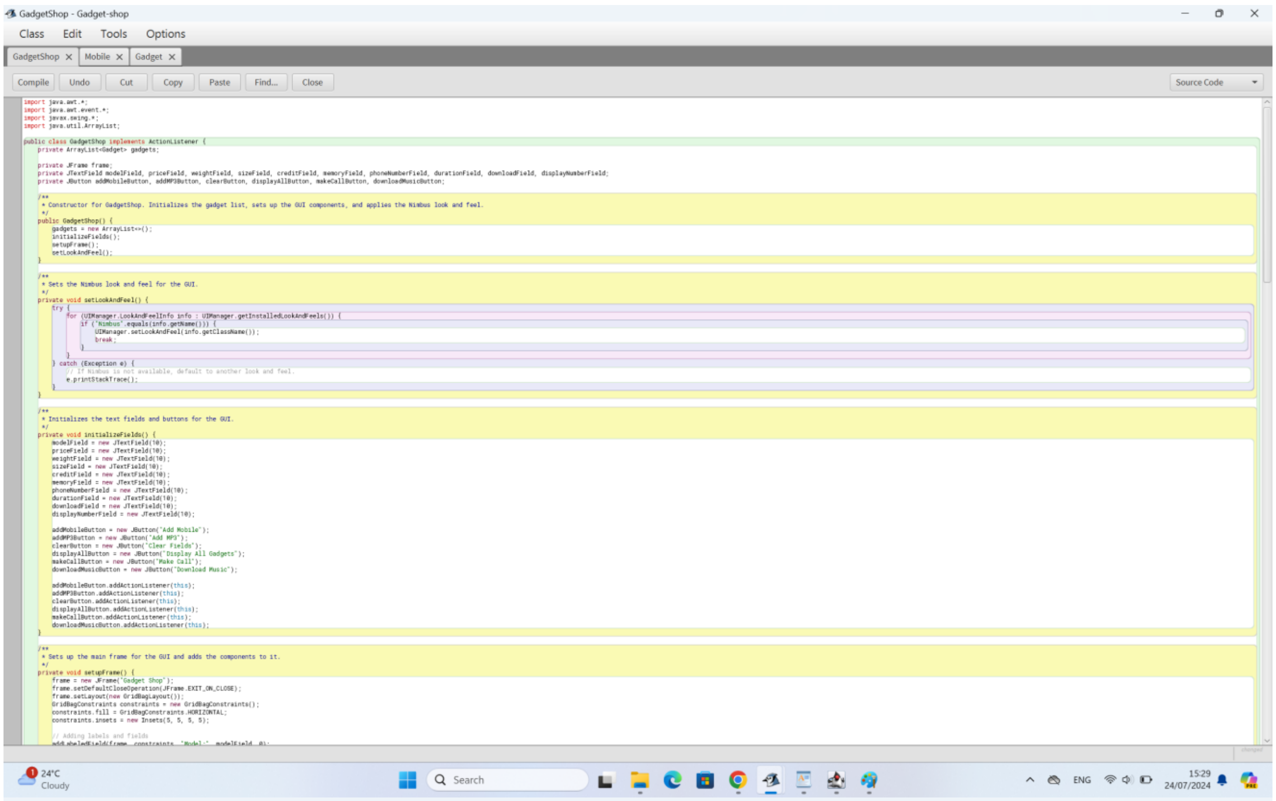
```java
/**
 * This is Super class under name Gadget which has 4 fields storing information about model, price, weigh and size
 * Written by Sandro Zakaidze aka Sandro Iberieli
 * All That Eye Saw and Witnessed
 */
public class Gadget {
    // Fields to Store information
    private String model;
    private double price;
    private int weight;
    private String size;

    // Constructor of Gadget Class
    public Gadget(String model, double price, int weight, String size) {
        this.model = model;
        this.price = price;
        this.weight = weight;
        this.size = size;
    }

    // Accessor Method Get Method returns model
    public String getModel() {
        return model;
    }

    // Accessor Method Get Price returns price
    public double getPrice() {
        return price;
    }

    // Accessor method Get Weight returns weight
    public int getWeight() {
        return weight;
    }

    // Accessor method Get Size returns size
    public String getSize() {
        return size;
    }

    // Get Display method
    public String display() {
        return "Model: " + model + "\nPrice: £" + price + "\nWeight: " + weight + " grams\nSize: " + size;
    }
}
```

Mobile Class:

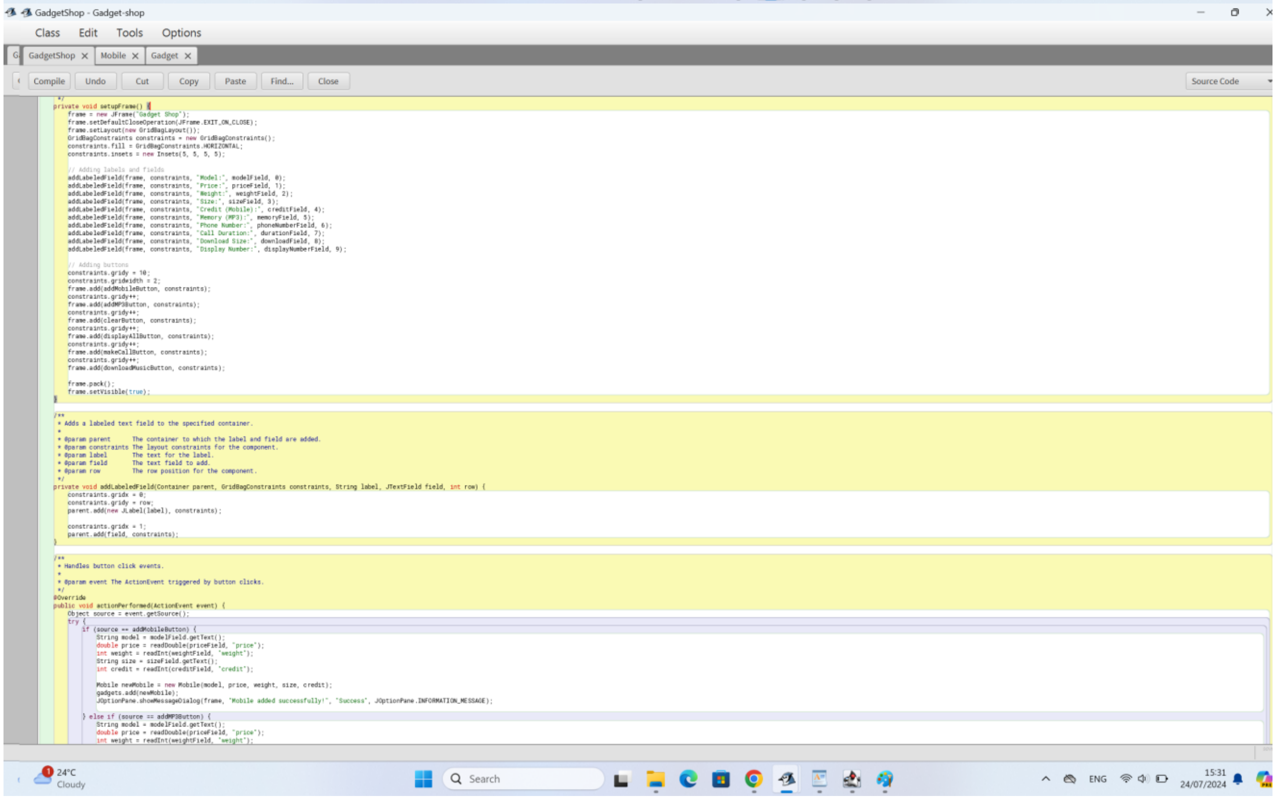


```java
/**
 * Mobile class that extends Gadget
 */
public class Mobile extends Gadget {
    // Field storing information for remaining credit
    private int callingCreditRemaining;

    // Constructor of Mobile class
    public Mobile(String model, double price, int weight, String size, int callingCreditRemaining) {
        super(model, price, weight, size);
        this.callingCreditRemaining = callingCreditRemaining;
    }

    // Get method to return remaining amount of credit
    public int getCredit() {
        return callingCreditRemaining;
    }

    // Method to insert credit
    public void insertCredit(int amount) {
        if (amount > 0) {
            callingCreditRemaining += amount; // This adds inserted amount to remaining amount
        } else {
            System.out.println("Please add a positive amount of credit!"); // This is printed if user inserts less than zero
        }
    }

    // Method to make phone call
    public void makePhoneCall(String phoneNumber, int durationOfCall) {
        if (callingCreditRemaining >= durationOfCall) {
            System.out.println("Provided number: " + phoneNumber + "\nProvided duration of call: " + durationOfCall); // Prints out number which user provided and duration of call
            callingCreditRemaining -= durationOfCall; // If sufficient amount of credit is in the system call is performed and remaining credit amount is decreased by duration of call performed
        } else {
            System.out.println("There is insufficient credit in the system, Please add the required amount to perform a phone call"); // Prints out information that credit in the system is not enough
            int requiredAmount = durationOfCall - callingCreditRemaining; // Local variable to calculate required amount for call to be performed
            System.out.println("Required amount of credits to be added: " + requiredAmount); // Required amount of credit is displayed to user
        }
    }

    // Enhanced display method for Mobile class
    @Override
    public String display() {
        return super.display() + "\nRemaining Credit: " + callingCreditRemaining + " minutes";
    }
}
```

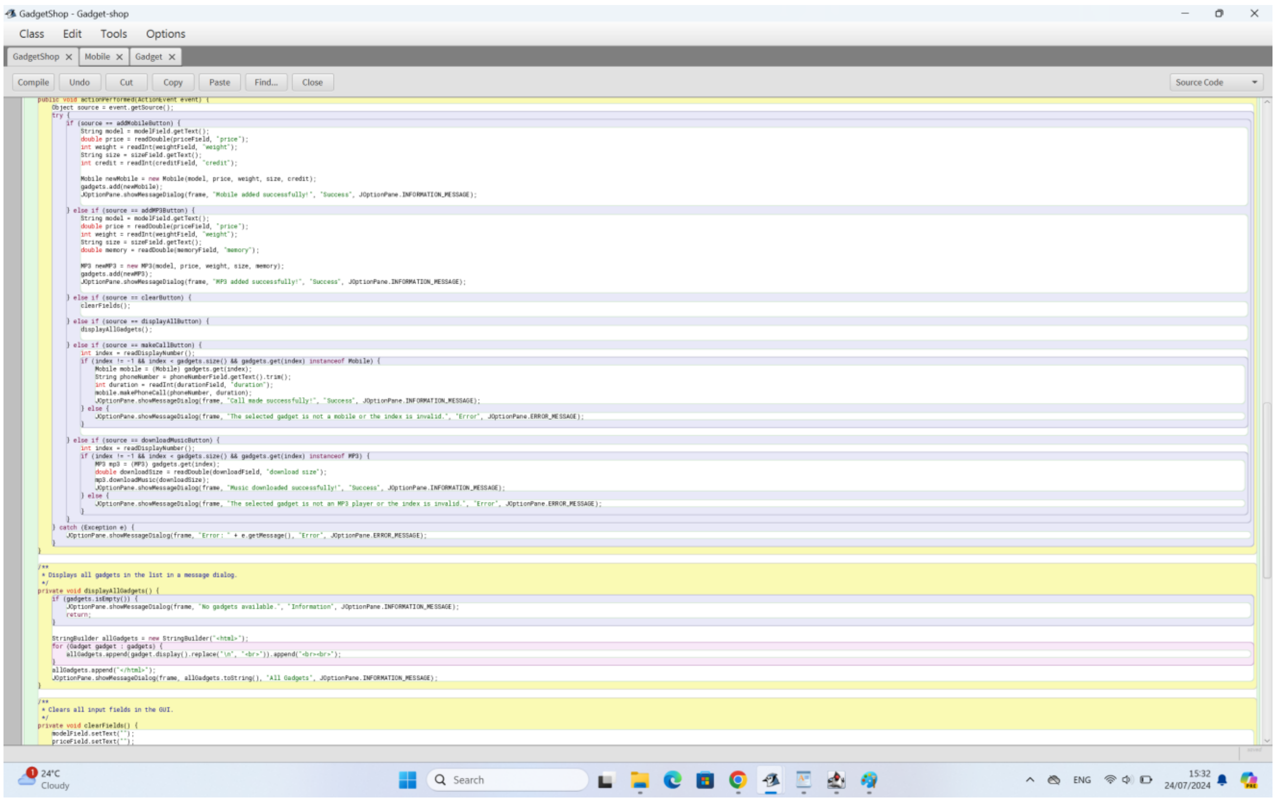
Gadget Shop GUI:



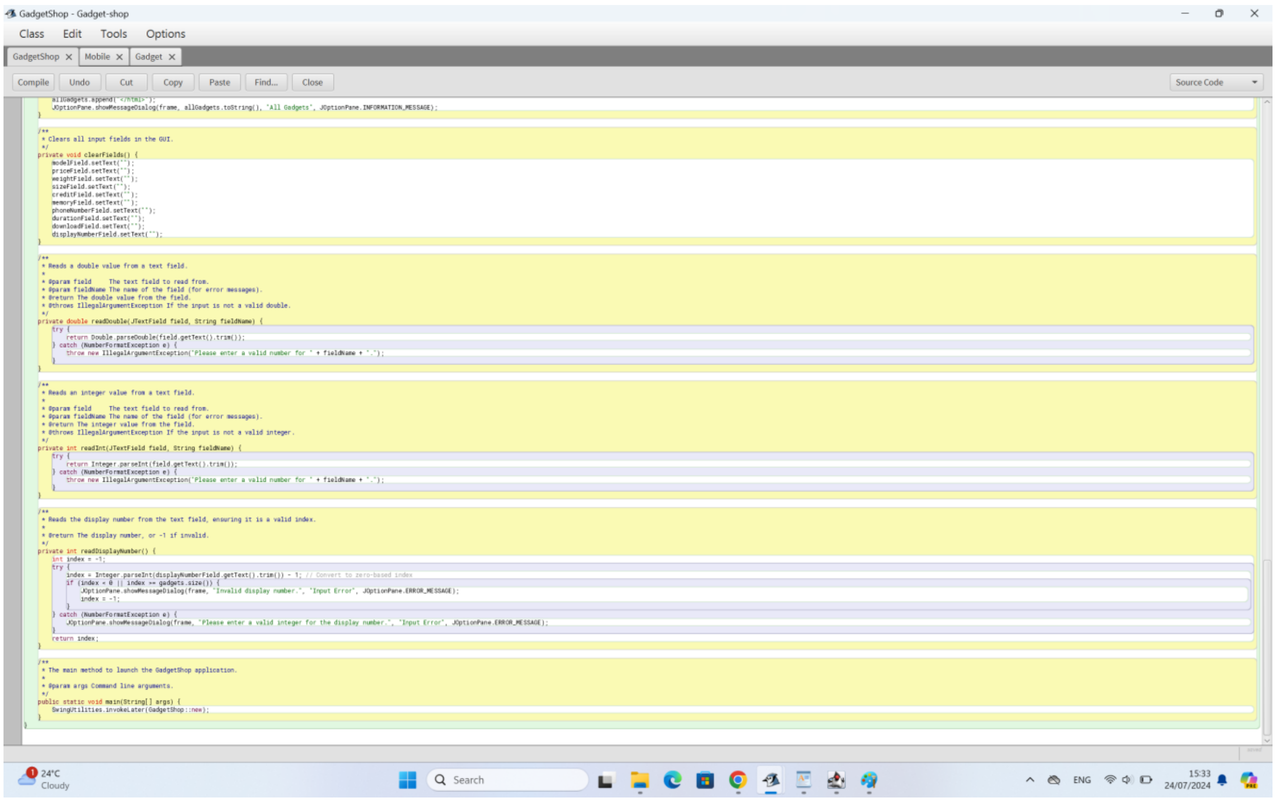
```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.ArrayList;

public class GadgetShop implements ActionListener {
    private ArrayList<Gadget> gadgets;

    private JFrame frame;
    private JTextField modelField, priceField, weightField, sizeField, creditField, memoryField, phoneNumberField, durationField, downloadField, displayNumberField;
    private JButton addMobileButton, addMP3Button, clearButton, displayAllButton, makeCallButton, downloadMusicButton;

    /**
     * Constructor for GadgetShop. Initializes the gadget list, sets up the GUI components, and applies the Nimbus look and feel.
     */
    public GadgetShop() {
        gadgets = new ArrayList<>();
        initializeFields();
        setupFrame();
        setLookAndFeel();
    }

    /**
     * Sets the Nimbus look and feel for the GUI.
     */
    private void setLookAndFeel() {
        try {
            for (UIManager.LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (Exception e) {
            // If Nimbus is not available, default to another look and feel.
            e.printStackTrace();
        }
    }

    /**
     * Initializes the text fields and buttons for the GUI.
     */
    private void initializeFields() {
        modelField = new JTextField(10);
        priceField = new JTextField(10);
        weightField = new JTextField(10);
        sizeField = new JTextField(10);
        creditField = new JTextField(10);
        memoryField = new JTextField(10);
        phoneNumberField = new JTextField(10);
        durationField = new JTextField(10);
        downloadField = new JTextField(10);
        displayNumberField = new JTextField(10);

        addMobileButton = new JButton("Add Mobile");
        addMP3Button = new JButton("Add MP3");
        clearButton = new JButton("Clear Fields");
        displayAllButton = new JButton("Display All Gadgets");
        makeCallButton = new JButton("Make Call");
        downloadMusicButton = new JButton("Download Music");

        addMobileButton.addActionListener(this);
        addMP3Button.addActionListener(this);
        clearButton.addActionListener(this);
        displayAllButton.addActionListener(this);
        makeCallButton.addActionListener(this);
        downloadMusicButton.addActionListener(this);
    }

    /**
     * Sets up the main frame for the GUI and adds the components to it.
     */
    private void setupFrame() {
        frame = new JFrame("Gadget Shop");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new GridBagLayout());
        GridBagConstraints constraints = new GridBagConstraints();
        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.insets = new Insets(5, 5, 5, 5);

        // Adding labels and fields
        addLabeledField(frame, constraints, "Model:", modelField, 0);
```



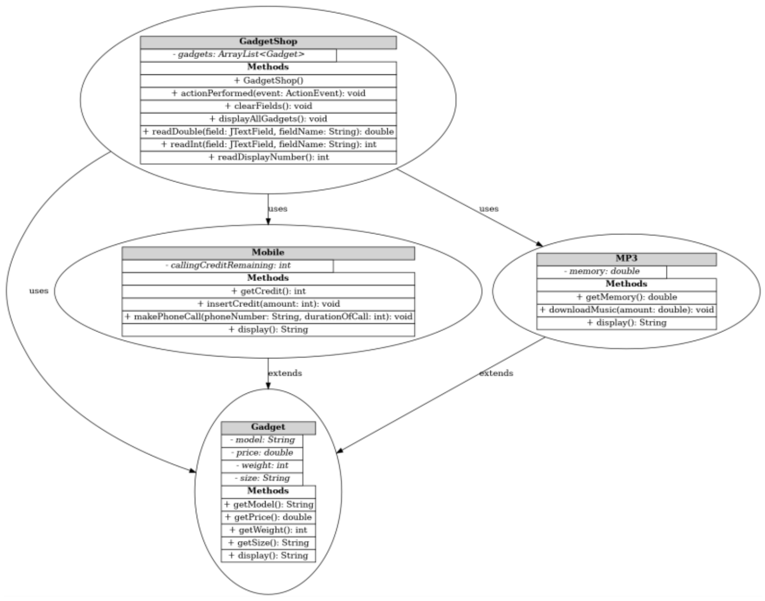
```java
     */
    private void setupFrame() {
        frame = new JFrame("Gadget Shop");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new GridBagLayout());
        GridBagConstraints constraints = new GridBagConstraints();
        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.insets = new Insets(5, 5, 5, 5);

        // Adding labels and fields
        addLabeledField(frame, constraints, "Model:", modelField, 0);
        addLabeledField(frame, constraints, "Price:", priceField, 1);
        addLabeledField(frame, constraints, "Weight:", weightField, 2);
        addLabeledField(frame, constraints, "Size:", sizeField, 3);
        addLabeledField(frame, constraints, "Credit (Mobile):", creditField, 4);
        addLabeledField(frame, constraints, "Memory (MP3):", memoryField, 5);
        addLabeledField(frame, constraints, "Phone Number:", phoneNumberField, 6);
        addLabeledField(frame, constraints, "Call Duration:", durationField, 7);
        addLabeledField(frame, constraints, "Download Size:", downloadField, 8);
        addLabeledField(frame, constraints, "Display Number:", displayNumberField, 9);

        // Adding buttons
        constraints.gridy = 10;
        constraints.gridwidth = 2;
        frame.add(addMobileButton, constraints);
        constraints.gridy++;
        frame.add(addMP3Button, constraints);
        constraints.gridy++;
        frame.add(clearButton, constraints);
        constraints.gridy++;
        frame.add(displayAllButton, constraints);
        constraints.gridy++;
        frame.add(makeCallButton, constraints);
        constraints.gridy++;
        frame.add(downloadMusicButton, constraints);

        frame.pack();
        frame.setVisible(true);
    }

    /**
     * Adds a labeled text field to the specified container.
     *
     * @param parent      The container to which the label and field are added.
     * @param constraints The layout constraints for the component.
     * @param label       The text for the label.
     * @param field       The text field to add.
     * @param row         The row position for the component.
     */
    private void addLabeledField(Container parent, GridBagConstraints constraints, String label, JTextField field, int row) {
        constraints.gridx = 0;
        constraints.gridy = row;
        parent.add(new JLabel(label), constraints);

        constraints.gridx = 1;
        parent.add(field, constraints);
    }

    /**
     * Handles button click events.
     *
     * @param event The ActionEvent triggered by button clicks.
     */
    @Override
    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();
        try {
            if (source == addMobileButton) {
                String model = modelField.getText();
                double price = readDouble(priceField, "price");
                int weight = readInt(weightField, "weight");
                String size = sizeField.getText();
                int credit = readInt(creditField, "credit");

                Mobile newMobile = new Mobile(model, price, weight, size, credit);
                gadgets.add(newMobile);
                JOptionPane.showMessageDialog(frame, "Mobile added successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);

            } else if (source == addMP3Button) {
                String model = modelField.getText();
                double price = readDouble(priceField, "price");
                int weight = readInt(weightField, "weight");
```



```java
public void actionPerformed(ActionEvent event) {
    Object source = event.getSource();
    try {
        if (source == addMobileButton) {
            String model = modelField.getText();
            double price = readDouble(priceField, "price");
            int weight = readInt(weightField, "weight");
            String size = sizeField.getText();
            int credit = readInt(creditField, "credit");

            Mobile newMobile = new Mobile(model, price, weight, size, credit);
            gadgets.add(newMobile);
            JOptionPane.showMessageDialog(frame, "Mobile added successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);

        } else if (source == addMP3Button) {
            String model = modelField.getText();
            double price = readDouble(priceField, "price");
            int weight = readInt(weightField, "weight");
            String size = sizeField.getText();
            double memory = readDouble(memoryField, "memory");

            MP3 newMP3 = new MP3(model, price, weight, size, memory);
            gadgets.add(newMP3);
            JOptionPane.showMessageDialog(frame, "MP3 added successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);

        } else if (source == clearButton) {
            clearFields();

        } else if (source == displayAllButton) {
            displayAllGadgets();

        } else if (source == makeCallButton) {
            int index = readDisplayNumber();
            if (index != -1 && index < gadgets.size() && gadgets.get(index) instanceof Mobile) {
                Mobile mobile = (Mobile) gadgets.get(index);
                String phoneNumber = phoneNumberField.getText().trim();
                int duration = readInt(durationField, "duration");
                mobile.makePhoneCall(phoneNumber, duration);
                JOptionPane.showMessageDialog(frame, "Call made successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(frame, "The selected gadget is not a mobile or the index is invalid.", "Error", JOptionPane.ERROR_MESSAGE);
            }

        } else if (source == downloadMusicButton) {
            int index = readDisplayNumber();
            if (index != -1 && index < gadgets.size() && gadgets.get(index) instanceof MP3) {
                MP3 mp3 = (MP3) gadgets.get(index);
                double downloadSize = readDouble(downloadField, "download size");
                mp3.downloadMusic(downloadSize);
                JOptionPane.showMessageDialog(frame, "Music downloaded successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(frame, "The selected gadget is not an MP3 player or the index is invalid.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(frame, "Error: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * Displays all gadgets in the list in a message dialog.
 */
private void displayAllGadgets() {
    if (gadgets.isEmpty()) {
        JOptionPane.showMessageDialog(frame, "No gadgets available.", "Information", JOptionPane.INFORMATION_MESSAGE);
        return;
    }

    StringBuilder allGadgets = new StringBuilder("<html>");
    for (Gadget gadget : gadgets) {
        allGadgets.append(gadget.display().replace("\n", "<br>")).append("<br><br>");
    }
    allGadgets.append("</html>");
    JOptionPane.showMessageDialog(frame, allGadgets.toString(), "All Gadgets", JOptionPane.INFORMATION_MESSAGE);
}

/**
 * Clears all input fields in the GUI.
 */
private void clearFields() {
    modelField.setText("");
    priceField.setText("");
```


```java
        allGadgets.append("</html>");
        JOptionPane.showMessageDialog(frame, allGadgets.toString(), "All Gadgets", JOptionPane.INFORMATION_MESSAGE);
    }

    /**
     * Clears all input fields in the GUI.
     */
    private void clearFields() {
        modelField.setText("");
        priceField.setText("");
        weightField.setText("");
        sizeField.setText("");
        creditField.setText("");
        memoryField.setText("");
        phoneNumberField.setText("");
        durationField.setText("");
        downloadField.setText("");
        displayNumberField.setText("");
    }

    /**
     * Reads a double value from a text field.
     *
     * @param field     The text field to read from.
     * @param fieldName The name of the field (for error messages).
     * @return The double value from the field.
     * @throws IllegalArgumentException If the input is not a valid double.
     */
    private double readDouble(JTextField field, String fieldName) {
        try {
            return Double.parseDouble(field.getText().trim());
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("Please enter a valid number for " + fieldName + ".");
        }
    }

    /**
     * Reads an integer value from a text field.
     *
     * @param field     The text field to read from.
     * @param fieldName The name of the field (for error messages).
     * @return The integer value from the field.
     * @throws IllegalArgumentException If the input is not a valid integer.
     */
    private int readInt(JTextField field, String fieldName) {
        try {
            return Integer.parseInt(field.getText().trim());
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("Please enter a valid number for " + fieldName + ".");
        }
    }

    /**
     * Reads the display number from the text field, ensuring it is a valid index.
     *
     * @return The display number, or -1 if invalid.
     */
    private int readDisplayNumber() {
        int index = -1;
        try {
            index = Integer.parseInt(displayNumberField.getText().trim()) - 1; // Convert to zero-based index
            if (index < 0 || index >= gadgets.size()) {
                JOptionPane.showMessageDialog(frame, "Invalid display number.", "Input Error", JOptionPane.ERROR_MESSAGE);
                index = -1;
            }
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(frame, "Please enter a valid integer for the display number.", "Input Error", JOptionPane.ERROR_MESSAGE);
        }
        return index;
    }

    /**
     * The main method to launch the GadgetShop application.
     *
     * @param args Command line arguments.
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(GadgetShop::new);
    }
}
```

UML DIAGRAM:


GadgetShop
- gadgets: ArrayList<Gadget>
Methods
+ GadgetShop()
+ actionPerformed(event: ActionEvent): void
+ clearFields(): void
+ displayAllGadgets(): void
+ readDouble(field: JTextField, fieldName: String): double
+ readInt(field: JTextField, fieldName: String): int
+ readDisplayNumber(): int
uses
uses
uses
Mobile
- callingCreditRemaining: int
Methods
+ getCredit(): int
+ insertCredit(amount: int): void
+ makePhoneCall(phoneNumber: String, durationOfCall: int): void
+ display(): String
MP3
- memory: double
Methods
+ getMemory(): double
+ downloadMusic(amount: double): void
+ display(): String
extends
extends
Gadget
- model: String
- price: double
- weight: int
- size: String
Methods
+ getModel(): String
+ getPrice(): double
+ getWeight(): int
+ getSize(): String
+ display(): String

CODE OF THE PROJECT:

GADGET CLASS:

```
/**
* This is Super class under name Gadget which has 4 fields storing information about model, price, weigh and size
* Written by Sandro Zakaidze aka Sandro Iberieli
* All That Eye Saw and Witnessed
*/
public class Gadget {
    // Fields to Store information
    private String model;
    private double price;
    private int weight;
    private String size;
```

```
// Constructor of Gadget Class

public Gadget(String model, double price, int weight, String size) {

    this.model = model;

    this.price = price;

    this.weight = weight;

    this.size = size;

}


// Accessor Method Get Method returns model

public String getModel() {

    return model;

}


// Accessor Method Get Price returns price

public double getPrice() {

    return price;

}


// Accessor method Get Weight returns weight

public int getWeight() {

    return weight;

}


// Accessor method Get Size returns size

public String getSize() {

    return size;
    }

    // Get Display method
    public String display() {
        return "Model: " + model + "\nPrice: £" + price + "\nWeight: " + weight + " grams\nSize: " + size;
    }
}
```

MOBILE CLASS:

```
/**
* Mobile class that extends Gadget
*/
public class Mobile extends Gadget {
    // Field storing information for remaining credit
    private int callingCreditRemaining;

    // Constructor of Mobile class
    public Mobile(String model, double price, int weight, String size, int callingCreditRemaining) {
        super(model, price, weight, size);
        this.callingCreditRemaining = callingCreditRemaining;
    }

    // Get method to return remaining amount of credit
    public int getCredit() {
        return callingCreditRemaining;

    }



    // Method to insert credit

    public void insertCredit(int amount) {

        if (amount > 0) {

            callingCreditRemaining += amount; // This adds inserted amount to remaining
amount

        } else {

            System.out.println("Please add a positive amount of credit!"); // This is printed if
user inserts less than zero

        }

    }



    // Method to make phone call

    public void makePhoneCall(String phoneNumber, int durationOfCall) {

        if (callingCreditRemaining >= durationOfCall) {

            System.out.println("Provided number: " + phoneNumber + "\nProvided duration of
call: " + durationOfCall); // Prints out number which user provided and duration of call

            callingCreditRemaining -= durationOfCall; // If sufficient amount of credit is in the
system call is performed and remaining credit amount is decreased by duration of call
performed

        } else {

            System.out.println("There is insufficient credit in the system, Please add the
required amount to perform a phone call"); // Prints out information that credit in the system
is not enough

            int requiredAmount = durationOfCall - callingCreditRemaining; // Local variable to
calculate required amount for call to be performed

            System.out.println("Required amount of credits to be added: " + requiredAmount);
// Required amount of credit is displayed to user

        }

    }
    // Enhanced display method for Mobile class
    @Override
    public String display() {
        return super.display() + "\nRemaining Credit: " + callingCreditRemaining + " minutes";
    }
}
```

MP3 CLASS:

```
/**
* MP3 class that extends Gadget
*/
public class MP3 extends Gadget {
    // Field storing available memory
    private double memory;

    // Constructor of MP3 class
    public MP3(String model, double price, int weight, String size, double memory) {
        super(model, price, weight, size);
        this.memory = memory;
    }

    // Get method to return available memory
    public double getMemory() {

        return memory;

    }



    // Method to download music

    public void downloadMusic(double amount) {

        if (memory >= amount) {

            memory -= amount;

            System.out.println("Starting to download your files, please wait!");

            System.out.println("File successfully downloaded!");

        } else {

            System.out.println("You do not have enough available memory to download the desired file");

            System.out.println("Please increase the amount of memory to download the desired file");

        }

    }



    // Enhanced display method for MP3 class

    @Override

    public String display() {

        return super.display() + "\nAvailable Memory: " + memory + " MB";

    }

}
```

GADGET SHOP GUI:

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

public class GadgetShop implements ActionListener {
    private ArrayList<Gadget> gadgets;

    private JFrame frame;
    private JTextField modelField, priceField, weightField, sizeField, creditField, memoryField, phoneNumberField, durationField, downloadField, displayNumberField;
    private JButton addMobileButton, addMP3Button, clearButton, displayAllButton, makeCallButton, downloadMusicButton;

    /**
     * Constructor for GadgetShop. Initializes the gadget list, sets up the GUI components, and applies the Nimbus look and feel.
     */
    public GadgetShop() {
        gadgets = new ArrayList<>();
        initializeFields();
        setupFrame();
        setLookAndFeel();
    }

    /**
     * Sets the Nimbus look and feel for the GUI.
```

```
 */
private void setLookAndFeel() {
    try {
        for (UIManager.LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (Exception e) {
        // If Nimbus is not available, default to another look and feel.
        e.printStackTrace();
    }
}

/**
 * Initializes the text fields and buttons for the GUI.
 */
private void initializeFields() {
    modelField = new JTextField(10);
    priceField = new JTextField(10);
    weightField = new JTextField(10);
    sizeField = new JTextField(10);
    creditField = new JTextField(10);
    memoryField = new JTextField(10);
    phoneNumberField = new JTextField(10);
    durationField = new JTextField(10);
```

```java
    downloadField = new JTextField(10);
    displayNumberField = new JTextField(10);


    addMobileButton = new JButton("Add Mobile");
    addMP3Button = new JButton("Add MP3");
    clearButton = new JButton("Clear Fields");
    displayAllButton = new JButton("Display All Gadgets");
    makeCallButton = new JButton("Make Call");
    downloadMusicButton = new JButton("Download Music");


    addMobileButton.addActionListener(this);
    addMP3Button.addActionListener(this);
    clearButton.addActionListener(this);
    displayAllButton.addActionListener(this);
    makeCallButton.addActionListener(this);
    downloadMusicButton.addActionListener(this);
}


/**
 * Sets up the main frame for the GUI and adds the components to it.
 */
private void setupFrame() {
    frame = new JFrame("Gadget Shop");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLayout(new GridBagLayout());
    GridBagConstraints constraints = new GridBagConstraints();
    constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.insets = new Insets(5, 5, 5, 5);


// Adding labels and fields

addLabeledField(frame, constraints, "Model:", modelField, 0);

addLabeledField(frame, constraints, "Price:", priceField, 1);

addLabeledField(frame, constraints, "Weight:", weightField, 2);

addLabeledField(frame, constraints, "Size:", sizeField, 3);

addLabeledField(frame, constraints, "Credit (Mobile):", creditField, 4);

addLabeledField(frame, constraints, "Memory (MP3):", memoryField, 5);

addLabeledField(frame, constraints, "Phone Number:", phoneNumberField, 6);

addLabeledField(frame, constraints, "Call Duration:", durationField, 7);

addLabeledField(frame, constraints, "Download Size:", downloadField, 8);

addLabeledField(frame, constraints, "Display Number:", displayNumberField, 9);


// Adding buttons

constraints.gridy = 10;

constraints.gridwidth = 2;

frame.add(addMobileButton, constraints);

constraints.gridy++;

frame.add(addMP3Button, constraints);

constraints.gridy++;

frame.add(clearButton, constraints);

constraints.gridy++;

frame.add(displayAllButton, constraints);

constraints.gridy++;

frame.add(makeCallButton, constraints);

constraints.gridy++;
```

```
        frame.add(downloadMusicButton, constraints);


        frame.pack();

        frame.setVisible(true);

    }


    /**

     * Adds a labeled text field to the specified container.

     *

     * @param parent      The container to which the label and field are added.

     * @param constraints The layout constraints for the component.

     * @param label       The text for the label.

     * @param field       The text field to add.

     * @param row         The row position for the component.

     */

    private void addLabeledField(Container parent, GridBagConstraints constraints, String
label, JTextField field, int row) {

        constraints.gridx = 0;

        constraints.gridy = row;

        parent.add(new JLabel(label), constraints);


        constraints.gridx = 1;

        parent.add(field, constraints);

    }


    /**

     * Handles button click events.

     *
     * @param event The ActionEvent triggered by button clicks.
     */
    @Override
    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();
        try {
            if (source == addMobileButton) {
                String model = modelField.getText();
                double price = readDouble(priceField, "price");
                int weight = readInt(weightField, "weight");
                String size = sizeField.getText();
                int credit = readInt(creditField, "credit");

                Mobile newMobile = new Mobile(model, price, weight, size, credit);
                gadgets.add(newMobile);
                JOptionPane.showMessageDialog(frame, "Mobile added successfully!",
"Success", JOptionPane.INFORMATION_MESSAGE);

            } else if (source == addMP3Button) {
                String model = modelField.getText();
                double price = readDouble(priceField, "price");
                int weight = readInt(weightField, "weight");
                String size = sizeField.getText();
                double memory = readDouble(memoryField, "memory");

                MP3 newMP3 = new MP3(model, price, weight, size, memory);
                gadgets.add(newMP3);
                JOptionPane.showMessageDialog(frame, "MP3 added successfully!", "Success",
JOptionPane.INFORMATION_MESSAGE);

            } else if (source == clearButton) {
                clearFields();

            } else if (source == displayAllButton) {
                displayAllGadgets();

            } else if (source == makeCallButton) {
                int index = readDisplayNumber();
                if (index != -1 && index < gadgets.size() && gadgets.get(index) instanceof Mobile)
{
                    Mobile mobile = (Mobile) gadgets.get(index);
                    String phoneNumber = phoneNumberField.getText().trim();
                    int duration = readInt(durationField, "duration");
                    mobile.makePhoneCall(phoneNumber, duration);
                    JOptionPane.showMessageDialog(frame, "Call made successfully!",
"Success", JOptionPane.INFORMATION_MESSAGE);
                } else {
                    JOptionPane.showMessageDialog(frame, "The selected gadget is not a mobile
or the index is invalid.", "Error", JOptionPane.ERROR_MESSAGE);
                }

            } else if (source == downloadMusicButton) {
                int index = readDisplayNumber();
                if (index != -1 && index < gadgets.size() && gadgets.get(index) instanceof MP3) {
                    MP3 mp3 = (MP3) gadgets.get(index);
                    double downloadSize = readDouble(downloadField, "download size");
                    mp3.downloadMusic(downloadSize);
                JOptionPane.showMessageDialog(frame, "Music downloaded successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);

            } else {

                JOptionPane.showMessageDialog(frame, "The selected gadget is not an MP3 player or the index is invalid.", "Error", JOptionPane.ERROR_MESSAGE);

            }

        }

    } catch (Exception e) {

        JOptionPane.showMessageDialog(frame, "Error: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);

    }

}



/**

 * Displays all gadgets in the list in a message dialog.

 */

private void displayAllGadgets() {

    if (gadgets.isEmpty()) {

        JOptionPane.showMessageDialog(frame, "No gadgets available.", "Information", JOptionPane.INFORMATION_MESSAGE);

        return;

    }



    StringBuilder allGadgets = new StringBuilder("<html>");

    for (Gadget gadget : gadgets) {

        allGadgets.append(gadget.display().replace("\n", "<br>")).append("<br><br>");

    }

    allGadgets.append("</html>");

    JOptionPane.showMessageDialog(frame, allGadgets.toString(), "All Gadgets", JOptionPane.INFORMATION_MESSAGE);
```

```
}

/**
 * Clears all input fields in the GUI.
 */
private void clearFields() {
    modelField.setText("");
    priceField.setText("");
    weightField.setText("");
    sizeField.setText("");
    creditField.setText("");
    memoryField.setText("");
    phoneNumberField.setText("");
    durationField.setText("");
    downloadField.setText("");
    displayNumberField.setText("");
}

/**
 * Reads a double value from a text field.
 *
 * @param field     The text field to read from.
 * @param fieldName The name of the field (for error messages).
 * @return The double value from the field.
 * @throws IllegalArgumentException If the input is not a valid double.
 */
private double readDouble(JTextField field, String fieldName) {
        try {
            return Double.parseDouble(field.getText().trim());
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("Please enter a valid number for " + fieldName
+ ".");
        }
    }

    /**
     * Reads an integer value from a text field.
     *
     * @param field     The text field to read from.
     * @param fieldName The name of the field (for error messages).
     * @return The integer value from the field.
     * @throws IllegalArgumentException If the input is not a valid integer.
     */
    private int readInt(JTextField field, String fieldName) {
        try {
            return Integer.parseInt(field.getText().trim());
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("Please enter a valid number for " + fieldName
+ ".");
        }
    }

    /**
     * Reads the display number from the text field, ensuring it is a valid index.
     *
     * @return The display number, or -1 if invalid.
     */
    private int readDisplayNumber() {
        int index = -1;
        try {
            index = Integer.parseInt(displayNumberField.getText().trim()) - 1; // Convert to zero-based index
            if (index < 0 || index >= gadgets.size()) {
                JOptionPane.showMessageDialog(frame, "Invalid display number.", "Input Error", JOptionPane.ERROR_MESSAGE);
                index = -1;
            }
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(frame, "Please enter a valid integer for the display number.", "Input Error", JOptionPane.ERROR_MESSAGE);
        }
        return index;
    }


    /**
     * The main method to launch the GadgetShop application.
     *
     * @param args Command line arguments.
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(GadgetShop::new);
    }
}
```

References

36. Deitel, P. J., & Deitel, H. M., *Java How to Program*, 11th end (Pearson, 2017).

37. Eckel, B., *Thinking in Java*, 4th end (Prentice Hall, 2006).

38. Schildt, H., *Java: The Complete Reference*, 11th (McGraw-Hill Education, 2018).

39. Robinson, R., Vorobiev, P., & Vorobiev, V., *Swing: A Beginner's Guide* (McGraw-Hill/Osborne, 2002).

40. Fowler, M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd (Addison-Wesley Professional, 2004).

41. Booch, G., Rumbaugh, J., & Jacobson, I., *The Unified Modeling Language User Guide*, 2nd (Addison-Wesley Professional, 2005).

42. Gaddis, T., *Starting Out with Java: From Control Structures through Objects*, 7th (Pearson, 2019).

43. Zukowski, J., *The Definitive Guide to Swing for Java 2* (2005).

44. Kolling, M., & Barnes, D., *BlueJ: Programming with Java*, 5th (Pearson, 2016).

45. Sommerville, I., *Software Engineering*, 10th edn (Pearson, 2015).

46. Pressman, R. S., *Software Engineering: A Practitioner's Approach*, 8th (McGraw-Hill Education, 2014).

47. Oracle Documentation: Java SE Documentation. Available at: https://docs.oracle.com/javase/8/docs/ [Accessed: 24 July 2024].

48. Stack Overflow. Available at: https://stackoverflow.com/ [Accessed: 24 July 2024].

49. Graphviz - Graph Visualization Software. Available at: https://graphviz.org/ [Accessed: 24 July

2024].

50. IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains. Available at: https://www.jetbrains.com/idea/ [Accessed: 24 July 2024].