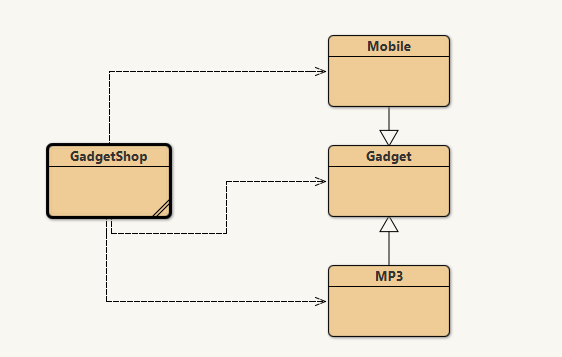
Gadget Shop System Report

Java GUI Project

**Class Diagram**



**Class and Method Descriptions**

**Gadget Class**

* **Gadget(String model, double price, int weight, String size)**: Constructor that initializes a gadget with the provided model, price, weight, and size.
* **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 string containing all the gadget details for display.

**Mobile Class (extends Gadget)**

* **Mobile(...)**: Constructor that initializes a mobile with model, price, weight, size, and credit.
* **addCredit(int)**: Adds calling credit to the mobile.
* **makeCall(String phoneNumber, int duration)**: Simulates making a call and deducts credit.
* **display()**: Overrides Gadget's display method to include credit info.

**MP3 Class (extends Gadget)**

* **MP3(...)**: Constructor that initializes an MP3 player with model, price, weight, size, and memory.
* **downloadMusic(int size)**: Simulates downloading music and decreases available memory.
* **display()**: Overrides Gadget's display method to include memory info.

**GadgetShop Class**

* Contains JavaFX UI components and an ArrayList to manage gadgets.
* **start(Stage)**: JavaFX entry point that sets up the GUI.
* Event handler methods for GUI actions.

**Pseudocode for Button-handling Methods**

**Add Mobile Button**

When Add Mobile button is clicked:

    Get input values for model, price, weight, size, and credit

    Validate the numerical inputs

    If validation is successful:

        Create a new Mobile object with the input values

        Add the Mobile object to the gadgets list

        Display success message

    Else:

        Display error message

**Add MP3 Button**

When Add MP3 button is clicked:

    Get input values for model, price, weight, size, and memory

    Validate the numerical inputs

    If validation is successful:

        Create a new MP3 object with the input values

        Add the MP3 object to the gadgets list

        Clear all input fields

        Display success message

    Else:

        Display error message

**Clear Button**

When Clear button is clicked:

    Clear all text fields in the GUI

**Make A Call Button**

When Make A Call button is clicked:

    Get the display number, phone number, and duration from input fields

    Validate the display number and duration are numerical and within range

    If validation is successful:

        Retrieve the gadget from the gadgets list using the display number

        If the gadget is a Mobile:

            Attempt to make a call with the phone number and duration

            If successful:

                Update the display with the remaining credit

                Display success message

            Else:

                Display error message for insufficient credit

        Else:

            Display error message for incorrect gadget type

    Else:

        Display error message for invalid input

**Download Music Button**

When Download Music button is clicked:

    Get the display number and download size from input fields

    Validate the display number and download size are numerical and within range

    If validation is successful:

        Retrieve the gadget from the gadgets list using the display number

        If the gadget is an MP3 player:

            Attempt to download music with the given size

            If successful:

                Update the display with the remaining memory

                Display success message

            Else:

                Display error message for insufficient memory

        Else:

            Display error message for incorrect gadget type

    Else:

        Display error message for invalid input

**Display All Button**

When Display All button is clicked:

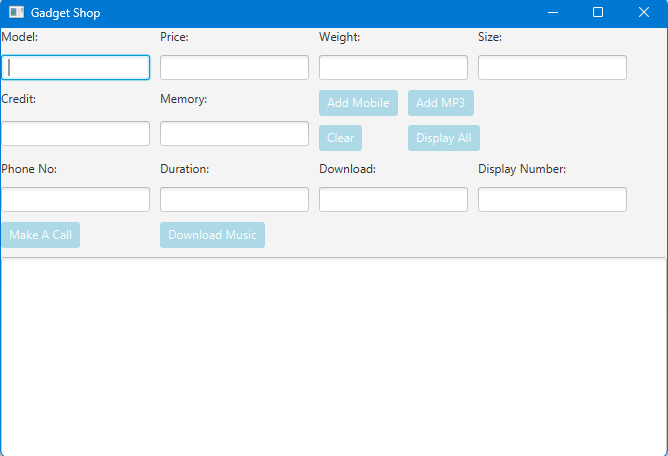
    Set an empty string for display

    For each gadget in gadgets list:

        Append gadget's display details to the string

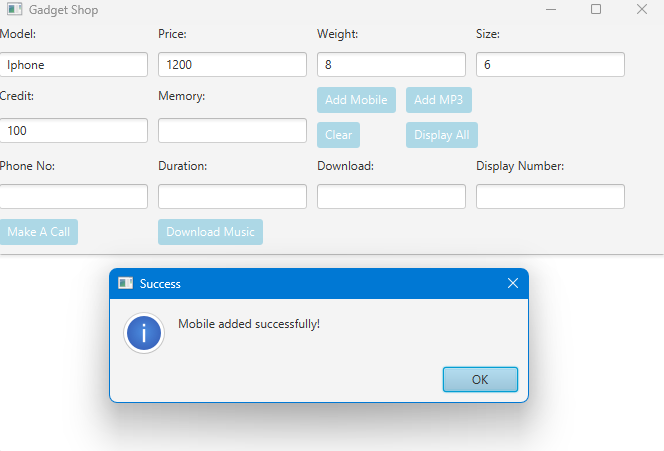
    Update the display area with the details string

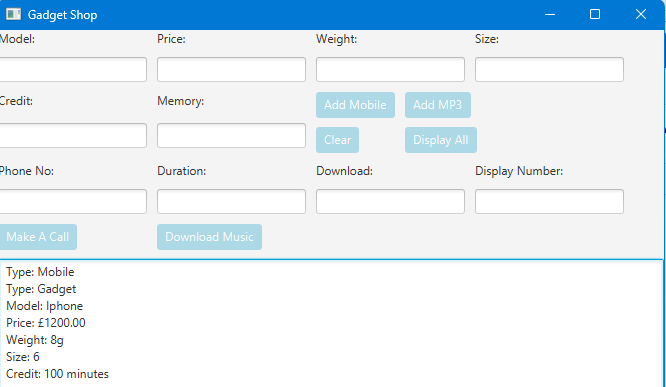
**GUI:**

****

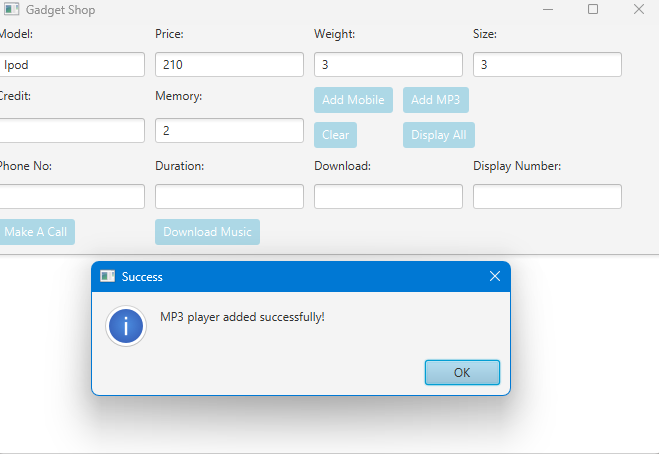
**Testing Evidence**

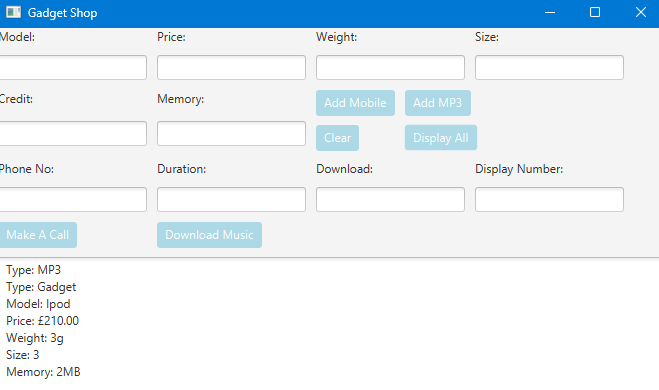
* ***Test Case 1: Adding a Mobile***
  + *Expected Outcome: Mobile is added to the list and displayed in the TextArea.*

**

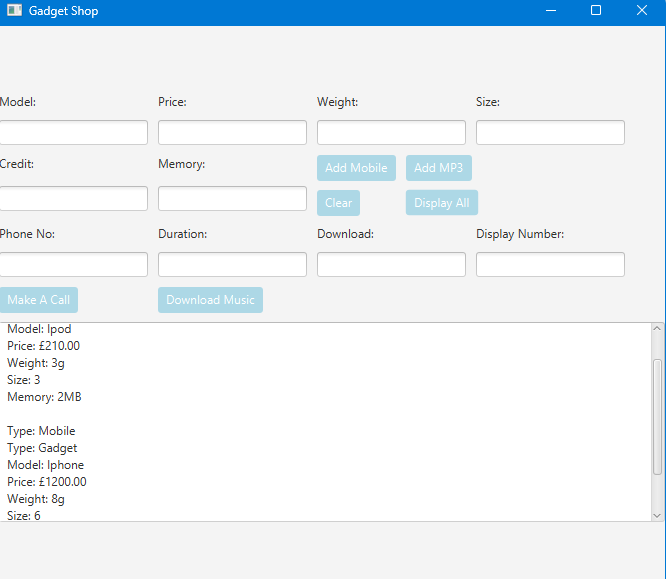
**

* ***Test Case 2: Adding an MP3 Player***
  + *Expected Outcome: MP3 player is added to the list and displayed in the TextArea.*

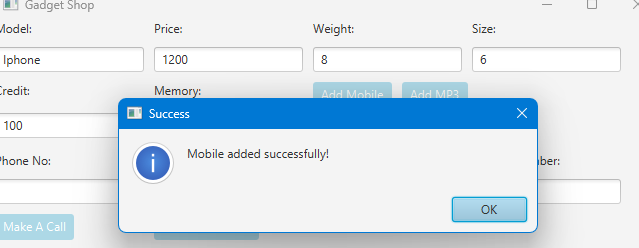
**

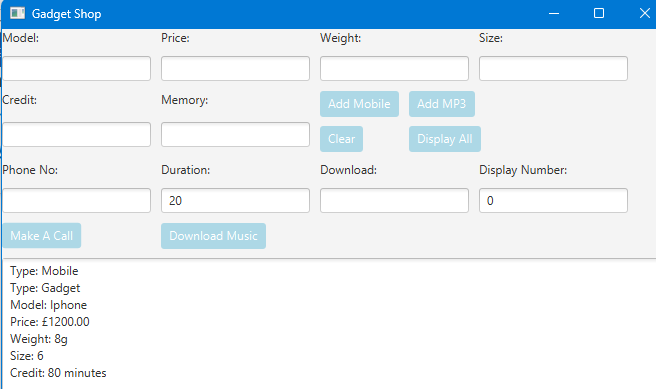
**

* ***Test Case 3: Displaying All Gadgets***
  + *Expected Outcome: All added gadgets are displayed in the TextArea with their details formatted.*

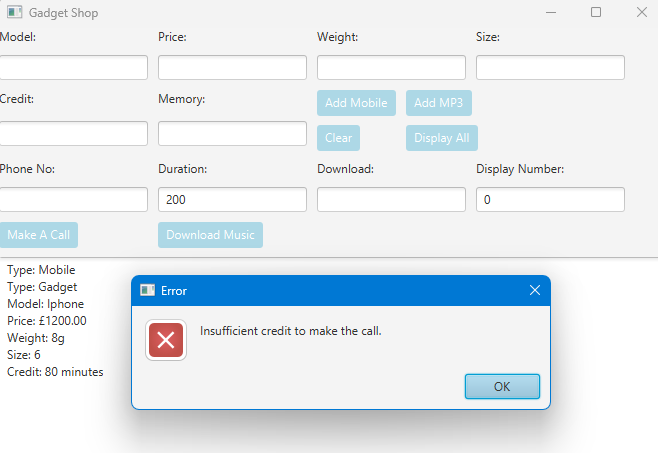
**

* ***Test Case 4: Making a Call with Sufficient Credit***
  + *Expected Outcome: Call is made, credit is deducted, and updated details are shown.*

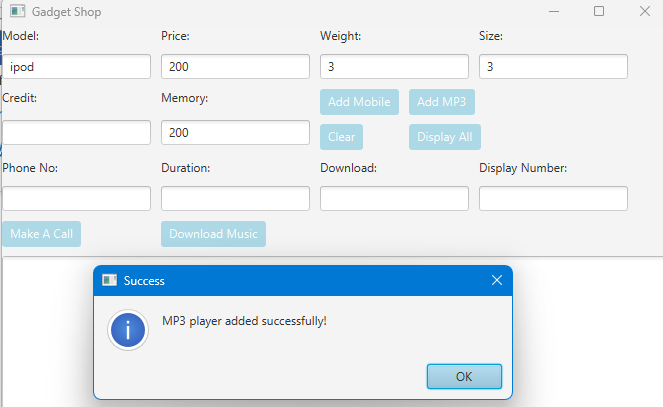
**

*Credits reduced to 80, initialing there were 100 but after making a ca:* **

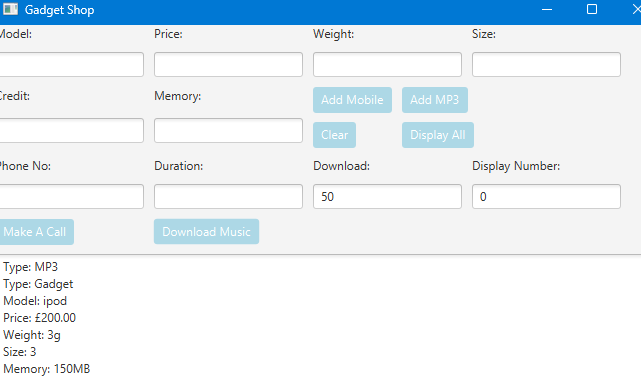
* ***Test Case 5: Making a Call with Insufficient Credit***
  + *Expected Outcome: Error message is displayed for insufficient credit*

**

* ***Test Case 6: Downloading Music with Sufficient Memory***
  + *Expected Outcome: Music is downloaded, memory is deducted, and updated details are shown.*

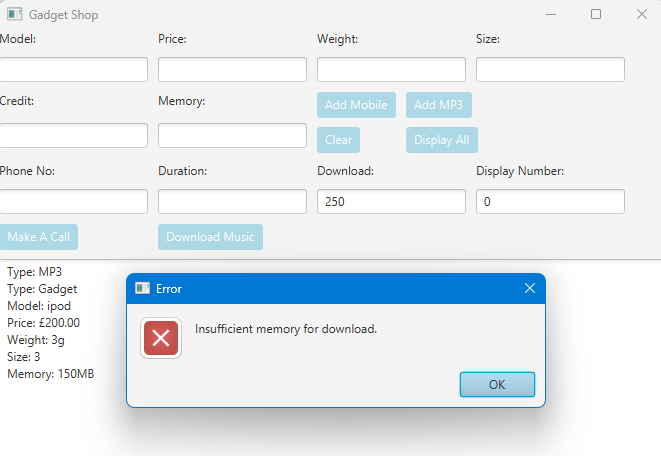
**

*Memory is 200, but after using 50:*

**

* ***Test Case 7: Downloading Music with Insufficient Memory***
  + *Expected Outcome: Error message is displayed for insufficient memory.*

*When 200:*

**

**Evaluation**

Throughout the assignment, I learned about JavaFX, a powerful platform for developing desktop apps with sophisticated graphical user interfaces. I was able to create a user-friendly interface without sacrificing application speed.

A crucial lesson was to strictly adhere to object-oriented concepts. The inheritance hierarchy of Gadget, Mobile, and MP3 required careful consideration of class design and method overrides.

This project helped me not only improve my technical abilities, but also create a problem-solving attitude, which is crucial for every software engineer.

**Appendix: Code Listings**

**GadgetShop**

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import javafx.scene.control.Label;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;

import javafx.geometry.Pos;

import javafx.scene.control.Alert;

import javafx.scene.control.Alert.AlertType;

import java.util.ArrayList;

import javafx.scene.control.TextArea;

public class GadgetShop extends Application {

      private ArrayList<Gadget> gadgets = new ArrayList<>();

    private TextField modelField = new TextField();

    private TextField priceField = new TextField();

    private TextField weightField = new TextField();

    private TextField sizeField = new TextField();

    private TextField creditField = new TextField();

    private TextField memoryField = new TextField();

    private TextField phoneNumberField = new TextField();

    private TextField durationField = new TextField();

    private TextField downloadField = new TextField();

    private TextField displayNumberField = new TextField();

  private TextArea displayArea = new TextArea();

    @Override

    public void start(Stage primaryStage) {

        primaryStage.setTitle("Gadget Shop");

        GridPane grid = new GridPane();

        grid.setAlignment(Pos.CENTER);

        grid.setHgap(10);

        grid.setVgap(10);

        // First Row

        grid.add(new Label("Model:"), 0, 0);

        grid.add(modelField, 0, 1);

        grid.add(new Label("Price:"), 1, 0);

        grid.add(priceField, 1, 1);

        grid.add(new Label("Weight:"), 2, 0);

        grid.add(weightField, 2, 1);

        grid.add(new Label("Size:"), 3, 0);

        grid.add(sizeField, 3, 1);

        // Second Row

        grid.add(new Label("Credit:"), 0, 2);

        grid.add(creditField, 0, 3);

        grid.add(new Label("Memory:"), 1, 2);

        grid.add(memoryField, 1, 3);

        // Button Grid

        GridPane buttonGrid = new GridPane();

        buttonGrid.setHgap(10);

        buttonGrid.setVgap(10);

        Button addMobileButton = new Button("Add Mobile");

        buttonGrid.add(addMobileButton, 0, 0);

        Button addMP3Button = new Button("Add MP3");

        buttonGrid.add(addMP3Button, 1, 0);

        Button clearButton = new Button("Clear");

        buttonGrid.add(clearButton, 0, 1);

        Button displayAllButton = new Button("Display All");

        buttonGrid.add(displayAllButton, 1, 1);

        grid.add(buttonGrid, 2, 2, 2, 2); // Span 2 columns and 2 rows for the button grid

        // Third Row

        grid.add(new Label("Phone No:"), 0, 4);

        grid.add(phoneNumberField, 0, 5);

        grid.add(new Label("Duration:"), 1, 4);

        grid.add(durationField, 1, 5);

        grid.add(new Label("Download:"), 2, 4);

        grid.add(downloadField, 2, 5);

        grid.add(new Label("Display Number:"), 3, 4);

        grid.add(displayNumberField, 3, 5);

        // Bottom Buttons

        Button makeCallButton = new Button("Make A Call");

        grid.add(makeCallButton, 0, 6);

        Button downloadMusicButton = new Button("Download Music");

        grid.add(downloadMusicButton, 1, 6);

         // Initialize the TextArea

        displayArea.setEditable(false);

        displayArea.setPrefHeight(200); // Set preferred height for the TextArea

        grid.add(displayArea, 0, 7, 8, 1); // Span 8 columns, update as necessary

        Scene scene = new Scene(grid); // Let the scene size be determined by the layout

        primaryStage.setScene(scene);

        primaryStage.show();

              // Set button color to blue

        setButtonStyle(addMobileButton);

        setButtonStyle(addMP3Button);

        setButtonStyle(clearButton);

        setButtonStyle(displayAllButton);

        setButtonStyle(makeCallButton);

        setButtonStyle(downloadMusicButton);

            addMobileButton.setOnAction(event -> handleAddMobile());

        addMP3Button.setOnAction(event -> handleAddMP3());

        clearButton.setOnAction(event -> handleClear());

        displayAllButton.setOnAction(event -> handleDisplayAll());

        makeCallButton.setOnAction(event -> handleMakeCall());

        downloadMusicButton.setOnAction(event -> handleDownloadMusic());

    }

      private void setButtonStyle(Button button) {

       button.setStyle("-fx-background-color: #ADD8E6; -fx-text-fill: white;");

    }

      private void handleAddMobile() {

        try {

            String model = modelField.getText();

            double price = Double.parseDouble(priceField.getText());

            int weight = Integer.parseInt(weightField.getText());

            String size = sizeField.getText();

            int credit = Integer.parseInt(creditField.getText());

            Mobile mobile = new Mobile(model, price, weight, size, credit);

            gadgets.add(mobile);

            showAlert(AlertType.INFORMATION, "Success", "Mobile added successfully!");

            handleClear();

        } catch (NumberFormatException e) {

            showAlert(AlertType.ERROR, "Error", "Invalid input. Please enter valid numbers for price, weight, and credit.");

        }

    }

    private void handleAddMP3() {

        try {

            String model = modelField.getText();

            double price = Double.parseDouble(priceField.getText());

            int weight = Integer.parseInt(weightField.getText());

            String size = sizeField.getText();

            int memory = Integer.parseInt(memoryField.getText());

            MP3 mp3 = new MP3(model, price, weight, size, memory);

            gadgets.add(mp3);

            showAlert(AlertType.INFORMATION, "Success", "MP3 player added successfully!");

            handleClear();

        } catch (NumberFormatException e) {

            showAlert(AlertType.ERROR, "Error", "Invalid input. Please enter valid numbers for price, weight, and memory.");

        }

    }

    private void handleClear() {

        modelField.clear();

        priceField.clear();

        weightField.clear();

        sizeField.clear();

        creditField.clear();

        memoryField.clear();

        phoneNumberField.clear();

        durationField.clear();

        downloadField.clear();

        displayNumberField.clear();

    }

    // Method to display details of all gadgets

    private void handleDisplayAll() {

        StringBuilder displayText = new StringBuilder();

        for (Gadget gadget : gadgets) {

            displayText.append(gadget.display()).append("\n");

        }

        displayArea.setText(displayText.toString());

    }

    private void handleMakeCall() {

    try {

        int displayNumber = Integer.parseInt(displayNumberField.getText());

        String phoneNumber = phoneNumberField.getText();

        int duration = Integer.parseInt(durationField.getText());

        if (displayNumber >= 0 && displayNumber < gadgets.size()) {

            Gadget gadget = gadgets.get(displayNumber);

            if (gadget instanceof Mobile) {

                Mobile mobile = (Mobile) gadget;

                if (mobile.getCredit() >= duration) {

                    mobile.makeCall(phoneNumber, duration);

                    displayArea.setText(mobile.display()); // Update TextArea with new details

                } else {

                    showAlert(Alert.AlertType.ERROR, "Error", "Insufficient credit to make the call.");

                }

            } else {

                showAlert(Alert.AlertType.ERROR, "Error", "Selected gadget is not a Mobile.");

            }

        } else {

            showAlert(Alert.AlertType.ERROR, "Error", "Invalid display number.");

        }

    } catch (NumberFormatException e) {

        showAlert(Alert.AlertType.ERROR, "Error", "Invalid input. Please enter valid numbers.");

    }

}

    // Code to simulate downloading music

private void handleDownloadMusic() {

    try {

        int displayNumber = Integer.parseInt(displayNumberField.getText());

        int downloadSize = Integer.parseInt(downloadField.getText());

        if (displayNumber >= 0 && displayNumber < gadgets.size()) {

            Gadget gadget = gadgets.get(displayNumber);

            if (gadget instanceof MP3) {

                MP3 mp3 = (MP3) gadget;

                if (mp3.getMemory() >= downloadSize) {

                    mp3.downloadMusic(downloadSize);

                    displayArea.setText(mp3.display()); // Update TextArea with new details

                } else {

                    showAlert(Alert.AlertType.ERROR, "Error", "Insufficient memory for download.");

                }

            } else {

                showAlert(Alert.AlertType.ERROR, "Error", "Selected gadget is not an MP3 player.");

            }

        } else {

            showAlert(Alert.AlertType.ERROR, "Error", "Invalid display number.");

        }

    } catch (NumberFormatException e) {

        showAlert(Alert.AlertType.ERROR, "Error", "Invalid input. Please enter valid numbers.");

    } catch (IllegalStateException e) {

        showAlert(Alert.AlertType.ERROR, "Error", e.getMessage());

    }

    private boolean validateNumberInput(String input) {

        // Implement number validation logic, return true if valid

        try {

            Double.parseDouble(input);

            return true;

        } catch (NumberFormatException e) {

            return false;

        }

    }

    //Show ALerts

       private void showAlert(AlertType alertType, String title, String message) {

        Alert alert = new Alert(alertType);

        alert.setTitle(title);

        alert.setHeaderText(null);

        alert.setContentText(message);

        alert.showAndWait();

    }

    //Main Method

    public static void main(String[] args) {

        launch(args);

    }

}

**Gadget:**

public class Gadget {

    private String model;

    private double price;

    private int weight;

    private String size;

    // Constructor for Gadget

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

        this.model = model;

        this.price = price;

        this.weight = weight;

        this.size = size;

    }

  // Method to get the current credit

    public int getCredit() {

        return credit;

    }

    // Accessor for model

    public String getModel() {

        return model;

    }

    // Accessor for price

    public double getPrice() {

        return price;

    }

    // Accessor for weight

    public int getWeight() {

        return weight;

    }

    // Accessor for size

    public String getSize() {

        return size;

    }

    // Display method for gadget details

    public String display() {

        return String.format("Type: Gadget\nModel: %s\nPrice: £%.2f\nWeight: %dg\nSize: %s\n",

                             model, price, weight, size);

    }

}

**Mobile:**

public class Mobile extends Gadget {

    private int credit;

    // Constructor for Mobile, extending Gadget

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

        super(model, price, weight, size);

        this.credit = credit;

    }

    // Adds calling credit

    public void addCredit(int amount) {

        if (amount > 0) {

            credit += amount;

        }

    }

    // Method to simulate making a call

    public void makeCall(String phoneNumber, int duration) {

        if (credit >= duration) {

            credit -= duration;

        }

    }

    // Overridden display method for mobile

    @Override

    public String display() {

        return String.format("Type: Mobile\n%sCredit: %d minutes\n",

                             super.display(), credit);

    }

}

**MP3:**

public class MP3 extends Gadget {

    private int memory;

    // Constructor for MP3, extending Gadget

    public MP3(String model, double price, int weight, String size, int memory) {

        super(model, price, weight, size);

        this.memory = memory;

    }

      // Method to simulate downloading music

    public void downloadMusic(int size) {

        if (size <= memory) {

            memory -= size;

        } else {

            throw new IllegalStateException("Insufficient memory for download.");

        }

    }

    // Get current memory

    public int getMemory() {

        return memory;

    }

    // Overridden display method for MP3

    @Override

    public String display() {

        return String.format("Type: MP3\n%sMemory: %dMB\n",

                             super.display(), memory);

    }

}

**Adherence to Style Guide**

My code strictly follows the recommended style guide. I ensured that:

* Class and variable names are in camelCase and are indicative of their purpose.
* Methods are named with action verbs and are kept short and focused on a single task.
* The code is properly indented, and blocks are consistently spaced to enhance readability.
* Comments are used to explain the intention behind complex logic.
* Exception handling is employed to gracefully manage runtime errors and provide feedback to the user.

By adhering to these guidelines, I aimed to make the code maintainable and easy to understand, thereby facilitating future enhancements and collaborative development.