**CS4001 Programming**

**Coursework: Gadget | Mobile | MP3 | GadgetShop GUI**

**Course Submission Cover Sheet**

***Module: CS4001 Programming Engineering***

***Component no: 003***

***Weighting: 60% of module mark***

***Deadline: 3rd of May 2024***

***Module Leader: Sandra Fernando | Student ID: 23032697***

<https://github.com/BelalUddin7?tab=repositories>

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:) Date:

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

Extracts from University *Regulations on*

Cheating, Plagiarism and Collusion

Section 2.3: "The following broad types of offence can be identified and are provided as indicative examples …..

1. **Cheating: including taking unauthorised material into an examination; consulting unauthorised material outside the examination hall during the examination; obtaining an unseen examination paper in advance of the examination; copying from another examinee; using an unauthorised calculator during the examination or storing unauthorised material in the memory of a programmable calculator which is taken into the examination; copying coursework.**
2. **Falsifying data in experimental results.**
3. Personation, where a substitute takes an examination or test on behalf of the candidate. Both candidate and substitute may be guilty of an offence under these Regulations.
4. **Bribery or attempted bribery of a person thought to have some influence on the candidate's assessment.**
5. Collusion to present joint work as the work solely of one individual.
6. Plagiarism, where the work or ideas of another are presented as the candidate's own.
7. Other conduct calculated to secure an advantage on assessment.
8. Assisting in any of the above.

Some notes on what this means for students:

1. Copying another student's work is an offence, whether from a copy on paper or from a computer file, and in whatever form the intellectual property being copied takes, including text, mathematical notation and computer programs.
2. Taking extracts from published sources *without attribution* is an offence. To quote ideas, sometimes using extracts, is generally to be encouraged. Quoting ideas is achieved by stating an author's argument and attributing it, perhaps by quoting, immediately in the text, his or her name and year of publication, e.g. " e = mc2 (Einstein 1905)". A *references* section at the end of your work should then list all such references in alphabetical order of authors' surnames. (There are variations on this referencing system which your tutors may prefer you to use.) If you wish to quote a paragraph or so from published work then indent the quotation on both left and right margins, using an italic font where practicable, and introduce the quotation with an attribution.

Table of Contents

[UML Class Diagram: 4](#_Toc165548037)

[Description of the class and following class and two sub – classes 5](#_Toc165548038)

[Pseudocode for the following button - handling methods 8](#_Toc165548039)

[Textboxes, input check with try/catch 10](#_Toc165548040)

[Explaining buttons and actions performed methods 13](#_Toc165548041)

[Running the programme in command prompt 14](#_Toc165548042)

[GUI: (Display, Add mobile, Make a call) (Display, Add MP3, Download music) 15](#_Toc165548043)

[Evidence of tests 17](#_Toc165548044)

[Section on error detection and correction 19](#_Toc165548045)

[Reflection and conclusion 20](#_Toc165548046)

## UML Class Diagram:

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generatedA screenshot of a phone code

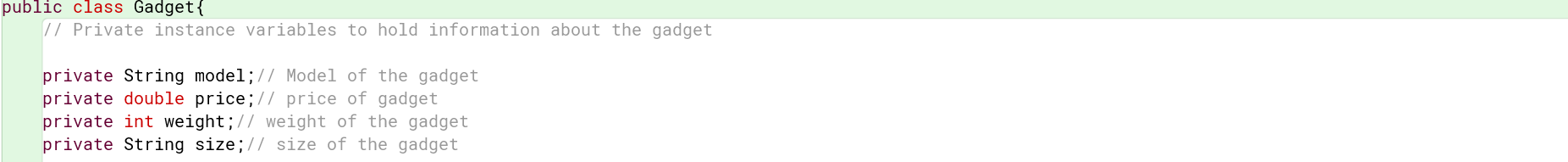
Description automatically generated

A screen shot of a computer program

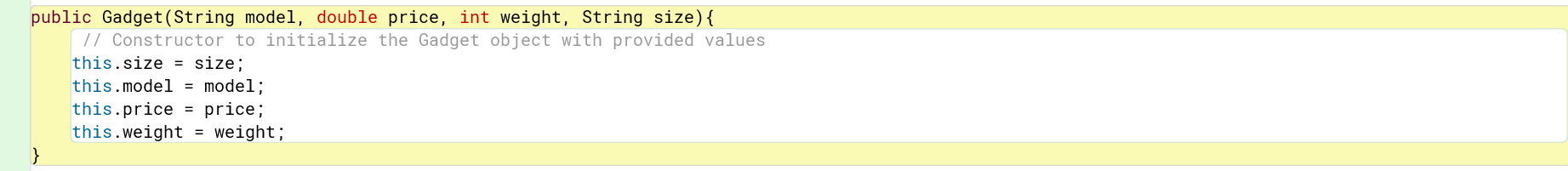
Description automatically generated

## Description of the class and following class and two sub – classes

***Gadget.java:***



The provided code defines a Java class named Gadget, which holds information about a gadget such as its model, price, weight, and size. This is all put into the field of the class. The 4 attributes are assigned with ***‘private’*** so it can be declared and accessed within the class.

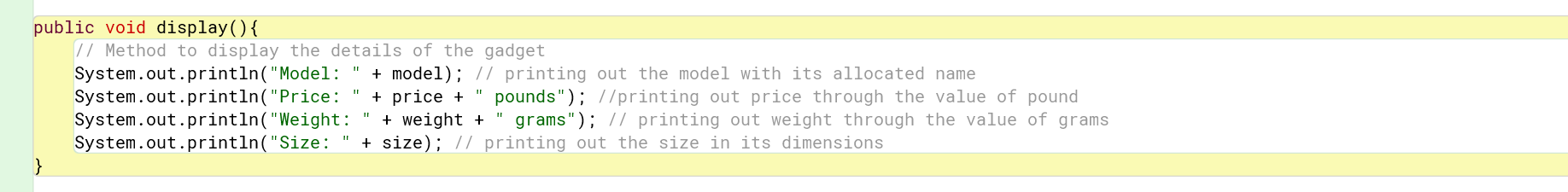


It includes a constructor method to initialize a new Gadget object with specific values for these attributes. They are assigned with ***‘this.’*** to allow the attributes to exist and function.

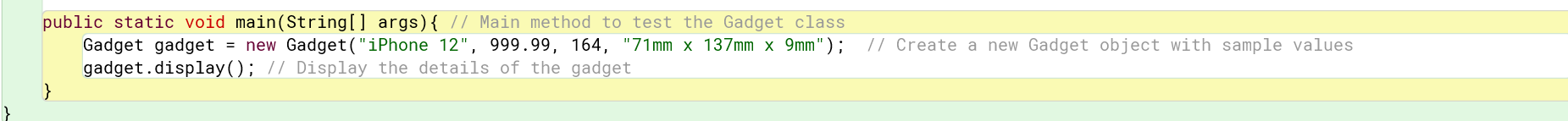
A screenshot of a computer

Description automatically generated

Additionally, there are getter methods for retrieving the model, price, weight, and size of the gadget. This accessor methods allows all the details to be moved and changed.



The display method is portrayed through the use of ***‘Public void display()’***. It prints out the details of the gadget, including its model, price (in pounds), weight (in grams), and size (specified in dimensions). ***‘System.out.println’*** is an instruction to specify which description to display.



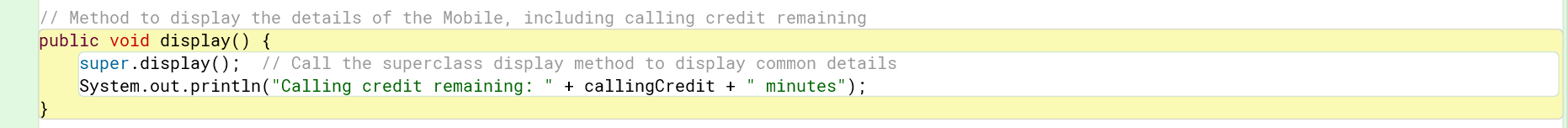
Finally, the main method serves as a test to the Gadget class by creating a new Gadget object with sample values and displaying its details using the display method. It displays all the final results of the whole code, and the main method***, ‘public static void main (string[] args){‘*** is used for this to function.

***Mobile.java:***

A screenshot of a computer

Description automatically generated

The Mobile class has additional features that are exclusive to mobile devices. The Mobile class introduces a new private instance variable named ***callingCredit***, which holds the mobile's remaining calling credit. Its constructor function initialises a new Mobile object by passing in values for model, price, weight, size, and calling credit. The superclass constructor is then called in order to create the inherited fields.



The ***display()*** method displays details of the mobile, including its inherited properties and the remaining calling credit.

A screenshot of a computer

Description automatically generated

There's a method ***addCallingCredit(int creditToAdd)*** to add calling credit to the mobile, with confirmation to ensure only positive amounts are added. Another method ***getCallingCredit()*** retrieves the calling credit of the mobile.

A close-up of a computer screen

Description automatically generated

The ***makeCall(String phoneNumber, int duration)*** method simulates making a call from the mobile, deducting the call duration from the calling credit if there's sufficient credit available.

A close-up of a box

Description automatically generated

The ***main(String[] args)*** function is the result of the class, by creating a Mobile object with sample data, displaying the initial information, adding calling credit, making a call, and updating the information following each operation.

***MP3.java:***

A screen shot of a computer

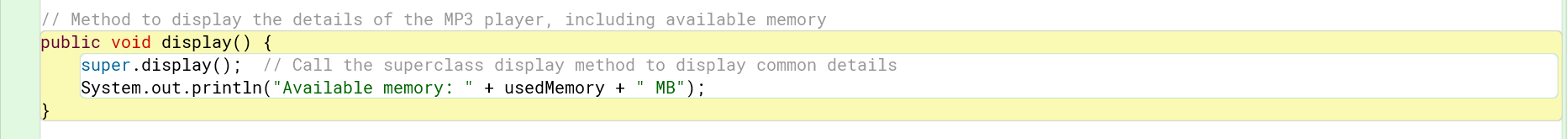
Description automatically generated

MP3 expands the Gadget class and adds functionality specific to MP3 players. The amount of memory utilised by the MP3 player is kept in a private instance variable called ***usedMemory*** within the MP3 class. Its constructor method initialises inherited properties and creates a new MP3 object with the model, price, weight, size, and used memory values specified by calling the superclass constructor.

A screenshot of a computer

Description automatically generated

There's a getter method, ***getUsedMemory()***, to retrieve the used memory of the MP3 player. Additionally, two methods are provided to manage the music library. One is ***downloadMusic(int memoryRequired***, allowing the user to download music to the MP3 player, reducing the used memory after downloading. The other is ***deleteMusic(int openMemory)***, which enables deleting music from the player, increasing the available memory.



The ***display()*** method shows the details of the MP3 player, including its inherited properties and the available memory.

A screenshot of a computer

Description automatically generated

***main(String[] args) method***, creates the MP3 object with sample values, showing initial details, downloading music, deleting music, and displaying updated details after each operation.

## Pseudocode for the following button - handling methods

***Adding a mobile:***

A screenshot of a computer program

Description automatically generated

***Adding an MP3:***

A computer program code with black text

Description automatically generated with medium confidence

***Displaying all gadgets in the array list:***

A screenshot of a computer code

Description automatically generated

***Getting the display number from the GUI:***

A white paper with black text

Description automatically generated

***Making a call:***

A black text on a white background

Description automatically generated

***Downloading music:***

A math equations on a white background

Description automatically generated

## Textboxes, input check with try/catch

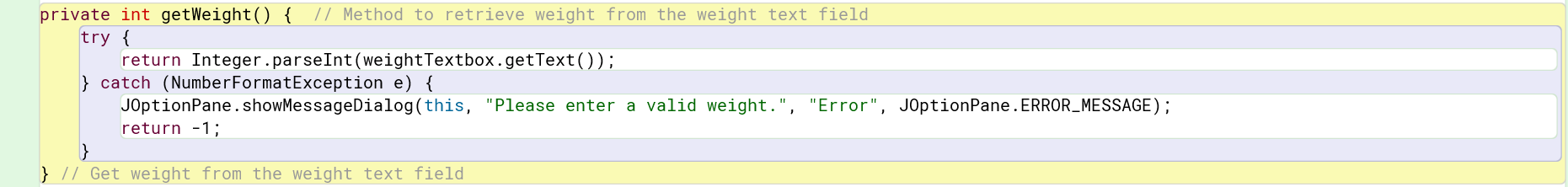
***Price Textbox:***

A screenshot of a computer

Description automatically generated

A user-inputted price from a text field component is retrieved, which may be referred to as ***priceTextbox.*** Within a try-catch block, the method attempts to extract the content from the text field and parse it into a double value using ***Double.parseDouble()***. If the method is successful, it returns the parsed price. However, in the event that the text cannot be correctly converted to a numerical value, for example, due to non-numeric characters or an empty field, ***a NumberFormatException*** is generated. Consequently, the function utilises a dialogue box provided by ***JOptionPane.showMessageDialog()*** to display an error message and asks the user to enter the proper pricing.

***Weight Textbox:***



Similarly, this method obtains a numerical weight input from a text field, which may be named ***weightTextbox***. The function looks for text in the weight text field and uses ***Integer.parseInt()*** to convert it to an integer value inside a try-catch block. The method, upon successful conversion, returns the weight value that was received. However, if the text cannot be converted into a proper number, an empty field or non-numeric characters are used as an indication that something is wrong and a ***NumberFormatException*** is produced. Therefore, the process prompts the user to enter the proper weight and uses ***JOptionPane.showMessageDialog()*** to generate a dialogue box to display an error message.

***Credit Textbox:***

A screenshot of a computer

Description automatically generated

A private function called getCredit() accepts an integer number as input and returns a text field named ***creditTextbox***. It attempts to transform the text field's contents into an integer using ***Integer.parseInt()***. If a ***NumberFormatException*** occurs during parsing, which often occurs when the input is not a valid integer, it describes the exception. In response, it asks the user to enter the correct credit amount and displays an error message using ***JOptionPane.showMessageDialog()***.

***Memory Textbox:***

A screenshot of a computer

Description automatically generated

This source code includes a private function called ***getCredit()*** which delivers a text field called creditTextbox in exchange for an integer value as input. It makes an effort to use Integer to convert the text field's contents into an ***integer.parseInt()***. It catches the exception whenever a ***NumberFormatException*** arises during parsing, which frequently happens when the input is not a valid integer. In response, it uses ***JOptionPane.showMessageDialog()*** to display an error message and prompts the user to provide the right credit amount.

***Duration Textbox:***

A screenshot of a computer

Description automatically generated

***getDuration()*** aims to extract an integer value from a text field labelled ***durationTextbox***, representing the duration. The method attempts to parse the content of the text field into an integer using ***Integer.parseInt()***. If this parsing fails due to a ***NumberFormatException***, typically indicating an invalid integer input, the method catches the exception. It then displays an error message via ***JOptionPane.showMessageDialog()*** to inform the user about the necessity for a valid duration entry.

***Download size Textbox:***

A screenshot of a computer

Description automatically generated

***getDownloadSize()*** designed to retrieve an integer value from a text field identified ***downloadSizeTextbox,*** presumably representing the size of a download. Within a try-catch block, it attempts to convert the content of the text field into an integer using ***Integer.parseInt()***. If this conversion fails due to a ***NumberFormatException***, indicating an invalid integer input, the method catches the exception. It then displays an error message using ***JOptionPane.showMessageDialog()*** to prompt the user for a valid download size entry.

***Display number Textbox:***

A screenshot of a computer

Description automatically generated

***getDisplayNumber()*** tasked with retrieving a display number from a combo box. It initializes **displayNumber** to -1 and attempts to parse the selected item from the combo box into an integer using ***Integer.parseInt()***. If successful, it checks whether the obtained display number falls within a valid range based on the gadgets list's size. If not within the range, it displays an error message, resets ***displayNumber*** to -1, and proceeds to catch the ***NumberFormatException***. In the catch block, it displays an error message notifying the user of the necessity for a valid display number.

## Explaining buttons and actions performed methods

A screenshot of a computer

Description automatically generated

This function is called each time a GUI action, such as clicking a button, takes place. Using **e.getSource()**, it determines which component initiated the activity and then takes the following actions in response:

* If the source of the action is **addMobileButton**, it retrieves the values from the text fields, creates a new **Mobile** object, adds it to the **gadgets** list, and clears the text fields.
* If the source is **addMP3Button**, it does similar operations but creates an **MP3** object instead.
* If the source is **clearButton**, it clears all the text fields.
* If the source is **displayButton**, it calls the **displayAll** method to display all gadgets in the text area.
* If the source is **makeCallButton**, it calls the **makeCall** method to initiate a call using the selected mobile gadget.
* If the source is **downloadMusicButton**, it calls the **downloadMusic** method to download music using the selected MP3 gadget.

A screenshot of a computer

Description automatically generated

This method retrieves the combo box's ***(cmbDisplayNumber)*** display number. It makes the selected item into an integer. If it is successful, it finds the appropriate range, which is between 0 and ***gadgets.size()*** - 1. An error notification is displayed otherwise. If the parsing fails, an error message is also displayed. In the end, the display number is given back. When combined, these methods ensure that user interactions are managed, that appropriate decisions are made in response to those interactions, and that any issues that may emerge are handled with grace.

## Running the programme in command prompt

A screenshot of a computer

Description automatically generated

In order for me to run the programme, through the command prompt, a JDK installation kit was required for my system to operate the java files. Once doing so, and opening the java files on command prompt, the following commands were inputted for the programme to successfully run.

## GUI: (Display, Add mobile, Make a call) (Display, Add MP3, Download music)

***Mobile:***

A screenshot of a computer

Description automatically generated

The mobile has been made from the GUI through inputting the model, price, size, weight and its available calling credit. 150 minutes of calling credit was assigned to it.

A screenshot of a computer

Description automatically generated

From the GUI a call has been inputted, for the mobile to make a call. The duration that the user has applied is for 50 minutes, therefore 50 minutes has been removed the initial 100 minutes.

***MP3:***

A screenshot of a computer

Description automatically generated

The MP3 has been added with the following attributes given to it. The model, price, weight, size and available memory. It was given 100MB worth of memory.

A screenshot of a computer

Description automatically generated

The music has been downloaded using 50MB of memory, as a result, 50MB of memory has been taken away from the initial 100MB.

## Evidence of tests

***Adding a mobile to the array list:***

A screenshot of a computer

Description automatically generated

***Adding an MP3 player to the array list:***

A screenshot of a computer

Description automatically generated

***Displaying the details of all of the gadgets in the array list:***

A screenshot of a computer

Description automatically generated

This screenshot consists of all the details of both the mobile class and MP3 class when they are being functioned. The mobile class has had 100 minutes of calling credit, but once a call worth 50 minutes was made, the total amount of minutes decreased to 50 minutes.

In regards to the MP3, it had 100MB worth of available memory, then once the music was downloaded and inputted from the GUI, 20MB was taken away from it.

***Test that appropriate dialog boxes appear when unsuitable values are entered for the display number:***

A screenshot of a computer

Description automatically generated

This screenshot above, displays an error message, when the wrong values are inputted for the display number. The code is set so that the display number is required to be more than minus one, however in the image above, the display number set for – 1.

## Section on error detection and correction

**1st error detection: Syntax error**

When lines of code are written incorrectly, a syntax error occurs. The majority of syntax mistakes are misspelt names or missing punctuation. The code will not function if a syntax mistake occurs in a programming language that is compiled or interpreted.

A white rectangular object with yellow border

Description automatically generated

In this section of the code, the word ‘duration’ is not spelt correctly and is not in accordance to how it was registered initially. Due to this, the code will not understand and not recognise what it is. This is proof of a syntax error occurring within the code.

**2nd error detection: Logical error**

When code in Java compiles and runs without issue but fails to provide the intended or expected result because of faulty programme logic, this is known as a logical error.

A screenshot of a phone

Description automatically generated

The display number in this code should have been -1, however due to mistakes, 4 was inputted. The programme ran the code but when the programme was being used, errors occurred and it was not possible to function the display number feature of the GUI.

**3rd error detection: Runtime error**

A runtime error occurs when a Java programme is executing and encounters an unexpected situation that prevents it from continuing. Unlike syntax errors, which the compiler detects, and logical errors, which result in incorrect output, runtime errors occur while the programme is executing.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

When converting the Gadget object to MP3 or Mobile in the makeCall() and downloadMusic() operations, a runtime error occurs. The casting assumes that the item in the gadgets list at the specified index is, in fact, an MP3 or mobile instance. But now there's a new device in the list, and that means the casting won't function in runtime.

## Reflection and conclusion

***Evaluation on work:***

Overall, from this project, the work I have carried out, it has ensured that I have been able to effectively follow the necessary steps needed to make a java programme on BLUEJ. The 4 classes which I coded are the gadget class, mobile class, mp3 class and a GUI, where I had to make sure they all have their unique set of codes and functions. For each code, It was necessary that they all had to interact with one another so they all had aspects in common which enabled the overall application to function as an entirety.

I engaged myself in having to check and test many alternatives to make sure the whole application is running at its optimum. The codes needed to successfully connect with one another for other functions to occur and move forward with the application. Also, the project had grew and codes became more prolonged as it progressed, so the quality and quantity intensified as I was doing more.

## References:

(kentdjb, 2020)

## Appendix:

***Gadget:***

public class Gadget{

// Private instance variables to hold information about the gadget

private String model;// Model of the gadget

private double price;// price of gadget

private int weight;// weight of the gadget

private String size;// size of the gadget

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

// Constructor to initialize the Gadget object with provided values

this.size = size;

this.model = model;

this.price = price;

this.weight = weight;

}

public String getModel(){

// Getter method to retrieve the model of the gadget

return model;

}

public double getPrice(){

// Getter method to retrieve the price of the gadget

return price;

}

public int getWeight(){

// Getter method to retrieve the weight of the gadget

return weight;

}

public String getSize(){

// Getter method to retrieve the size of the gadget

return size;

}

public void display(){

// Method to display the details of the gadget

System.out.println("Model: " + model); // printing out the model with its allocated name

System.out.println("Price: " + price + " pounds"); //printing out price through the value of pound

System.out.println("Weight: " + weight + " grams"); // printing out weight through the value of grams

System.out.println("Size: " + size); // printing out the size in its dimensions

}

public static void main(String[] args){ // Main method to test the Gadget class

Gadget gadget = new Gadget("iPhone 12", 999.99, 164, "71mm x 137mm x 9mm"); // Create a new Gadget object with sample values

gadget.display(); // Display the details of the gadget

}

}

Mobile:

public class Mobile extends Gadget {

// Additional private instance variable to hold the calling credit of the mobile

private int callingCredit;

// Constructor to initialize the Mobile object with provided values, including calling credit

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

// Call the superclass constructor to initialize inherited properties

super(model, price, weight, size);

this.callingCredit = callingCredit;

}

// Method to display the details of the Mobile, including calling credit remaining

public void display() {

super.display(); // Call the superclass display method to display common details

System.out.println("Calling credit remaining: " + callingCredit + " minutes");

}

// Method to add calling credit to the mobile

public void addCallingCredit(int creditToAdd) {

if (creditToAdd > 0) {

callingCredit += creditToAdd; // Add the provided credit to the existing calling credit

} else {

System.out.println("Please enter a positive amount to add calling credit.");

}

}

// Getter method to retrieve the calling credit of the mobile

public int getCallingCredit() {

return callingCredit;

}

// Method to make a call from the mobile

public void makeCall(String phoneNumber, int duration) {

if (duration <= callingCredit) { // Check if there is sufficient calling credit

System.out.println("Making call to " + phoneNumber + " for " + duration + " minutes.");

callingCredit -= duration; // Deduct the duration from the calling credit

} else {

System.out.println("Insufficient calling credit.");

}

}

// Main method to test the Mobile class

public static void main(String[] args) {

// Create a new Mobile object with sample values

Mobile mobile = new Mobile("iPhone 5s", 45.6, 134, "89mm x 120mm x 12mm", 100);

mobile.display(); // Display initial details

mobile.addCallingCredit(50); // Add calling credit and display updated details

mobile.makeCall("123456789", 30); // Make a call and display updated details

mobile.display();

}

}

MP3:

public class MP3 extends Gadget {

// Additional private instance variable to hold the used memory of the MP3 player

private int usedMemory;

// Constructor to initialize the MP3 object with provided values, including used memory

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

// Call the superclass constructor to initialize inherited properties

super(model, price, weight, size);

this.usedMemory = usedMemory;

}

// Getter method to retrieve the used memory of the MP3 player

public int getUsedMemory() {

return usedMemory;

}

// Method to download music to the MP3 player

public void downloadMusic(int memoryRequired) {

if (memoryRequired <= usedMemory) { // Check if there is enough memory

usedMemory -= memoryRequired; // Reduce the used memory after downloading

System.out.println("Music downloaded.");

} else {

System.out.println("Insufficient memory to download the music.");

}

}

// Method to delete music from the MP3 player

public void deleteMusic(int openMemory) {

usedMemory += openMemory; // Increase the available memory after deleting music

System.out.println("Music deleted.");

}

// Method to display the details of the MP3 player, including available memory

public void display() {

super.display(); // Call the superclass display method to display common details

System.out.println("Available memory: " + usedMemory + " MB");

}

// Main method to test the MP3 class

public static void main(String[] args) {

// Create a new MP3 object with sample values

MP3 mp3 = new MP3("iPod", 199.99, 38, "2.4in x 1.6in x 0.45in", 5700);

mp3.display(); // Display initial details

mp3.downloadMusic(1500); // Download music and display updated details

mp3.display();

mp3.deleteMusic(700); // Delete music and display updated details

mp3.display();

}

}

GUI:

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.ArrayList;

public class GadgetShopGUI extends JFrame implements ActionListener{ // Class representing the GUI for a Gadget Shop

private ArrayList<Gadget> gadgets; // ArrayList to store gadgets

private JTextField modelTextbox, priceTextbox, weightTextbox, sizeTextbox, creditTextbox, memoryTextbox, phoneNumberTextbox, durationTextbox, downloadSizeTextbox;

// Text fields for various gadget attributes

private JTextArea txtDisplay; // Text area to display gadget information

private JComboBox<String> cmbDisplayNumber; // Combo box to select display number

private JButton addMobileButton, addMP3Button, clearButton, displayButton, makeCallButton, downloadMusicButton; // Buttons for different actions

public GadgetShopGUI(){ // Constructor for initializing the GUI

gadgets = new ArrayList<>(); // Initialize gadgets ArrayList

setTitle("Gadget Shop"); // Set title of the frame

setSize(500, 600); // Set size of the frame

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Set default close operation

setLayout(new GridLayout(7, 2)); // Set layout for the frame

// Add labels and text fields for various gadget attributes

JLabel lblModel = new JLabel("Model:");

add(lblModel);

modelTextbox = new JTextField();

add(modelTextbox);

JLabel lblPrice = new JLabel("Price:");

add(lblPrice);

priceTextbox = new JTextField();

add(priceTextbox);

JLabel lblWeight = new JLabel("Weight:");

add(lblWeight);

weightTextbox = new JTextField();

add(weightTextbox);

JLabel lblSize = new JLabel("Size:");

add(lblSize);

sizeTextbox = new JTextField();

add(sizeTextbox);

JLabel lblCredit = new JLabel("Credit:");

add(lblCredit);

creditTextbox = new JTextField();

add(creditTextbox);

JLabel lblMemory = new JLabel("Memory:");

add(lblMemory);

memoryTextbox = new JTextField();

add(memoryTextbox);

JLabel lblPhoneNumber = new JLabel("Phone Number:");

add(lblPhoneNumber);

phoneNumberTextbox = new JTextField();

add(phoneNumberTextbox);

JLabel lblDuration = new JLabel("Duration:");

add(lblDuration);

durationTextbox = new JTextField();

add(durationTextbox);

JLabel lblDownloadSize = new JLabel("Download Size:");

add(lblDownloadSize);

downloadSizeTextbox = new JTextField();

add(downloadSizeTextbox);

JLabel lblDisplayNumber = new JLabel("Display Number:");

add(lblDisplayNumber);

String[] displayNumbers = {"-1", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"};

cmbDisplayNumber = new JComboBox<>(displayNumbers);

add(cmbDisplayNumber);

txtDisplay = new JTextArea();

add(txtDisplay);

// Add buttons for different actions

addMobileButton = new JButton("Add Mobile");

addMobileButton.addActionListener(this);

add(addMobileButton);

addMP3Button = new JButton("Add MP3");

addMP3Button.addActionListener(this);

add(addMP3Button);

clearButton = new JButton("Clear");

clearButton.addActionListener(this);

add(clearButton);

displayButton = new JButton("Display All");

displayButton.addActionListener(this);

add(displayButton);

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

makeCallButton.addActionListener(this);

add(makeCallButton);

downloadMusicButton = new JButton("Download Music");

downloadMusicButton.addActionListener(this);

add(downloadMusicButton);

setVisible(true); // Set visibility of the frame

}

@Override // Method called when an action is performed

public void actionPerformed(ActionEvent e){ // Handle different actions based on the event source

if (e.getSource() == addMobileButton) {

String model = getModel();

double price = getPrice();

int weight = getWeight();

String size = getGadgetsSize();

int credit = getCredit();

gadgets.add(new Mobile(model, price, weight, size, credit));

clearFields();

} else if (e.getSource() == addMP3Button) {

String model = getModel();

double price = getPrice();

int weight = getWeight();

String size = getGadgetsSize();

int memory = getMemory();

gadgets.add(new MP3(model, price, weight, size, memory));

clearFields();

} else if (e.getSource() == clearButton) {

clearFields();

} else if (e.getSource() == displayButton) {

displayAll();

} else if (e.getSource() == makeCallButton) {

makeCall();

} else if (e.getSource() == downloadMusicButton) {

downloadMusic();

}

}

private String getModel() { // Method to retrieve model from the model text field

return modelTextbox.getText();

} // Get model from the model text field

private double getPrice() { // Method to retrieve price from the price text field

try {

return Double.parseDouble(priceTextbox.getText());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Please enter a valid price.", "Error", JOptionPane.ERROR\_MESSAGE);

return -1;

}

} // Get price from the price text field

private int getWeight() { // Method to retrieve weight from the weight text field

try {

return Integer.parseInt(weightTextbox.getText());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Please enter a valid weight.", "Error", JOptionPane.ERROR\_MESSAGE);

return -1;

}

} // Get weight from the weight text field

private String getGadgetsSize(){ // Method to retrieve size from the size text field

return sizeTextbox.getText();

} // Get size from the size text field

private int getCredit() { // Method to retrieve credit from the credit text field

try {

return Integer.parseInt(creditTextbox.getText());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Please enter a valid credit amount.", "Error", JOptionPane.ERROR\_MESSAGE);

return -1;

}

} // Get credit from the credit text field

private int getMemory() { // Method to retrieve memory from the memory text field

try {

return Integer.parseInt(memoryTextbox.getText());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Please enter a valid memory size.", "Error", JOptionPane.ERROR\_MESSAGE);

return -1;

}

} // Get memory from the memory text field

private String getPhoneNumber() { // Method to retrieve phone number from the phone number text field

return phoneNumberTextbox.getText();

} // Get phone number from the phone number text field

private int getDuration() { // Method to retrieve duration from the duration text field

try {

return Integer.parseInt(durationTextbox.getText());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Please enter a valid duration.", "Error", JOptionPane.ERROR\_MESSAGE);

return -1;

}

} // Get duration from the duration text field

private int getDownloadSize() { // Method to retrieve download size from the download size text field

try {

return Integer.parseInt(downloadSizeTextbox.getText());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Please enter a valid download size.", "Error", JOptionPane.ERROR\_MESSAGE);

return -1;

}

} // Get download size from the download size text field

private int getDisplayNumber() { // Method to retrieve display number from the combo box

int displayNumber = -1;

try {

displayNumber = Integer.parseInt((String) cmbDisplayNumber.getSelectedItem());

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

JOptionPane.showMessageDialog(this, "Please enter a valid display number.", "Error", JOptionPane.ERROR\_MESSAGE);

displayNumber = -1;

}

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Please enter a valid display number.", "Error", JOptionPane.ERROR\_MESSAGE);

}

return displayNumber;

} // Get display number from the combo box

private void clearFields() { // Method to clear all text fields

modelTextbox.setText("");

priceTextbox.setText("");

weightTextbox.setText("");

sizeTextbox.setText("");

creditTextbox.setText("");

memoryTextbox.setText("");

phoneNumberTextbox.setText("");

durationTextbox.setText("");

downloadSizeTextbox.setText("");

} // Clear all text fields

private void displayAll() { // Method to display information of all gadgets in the text area

txtDisplay.setText("");

for (Gadget gadget : gadgets) {

gadget.display();

txtDisplay.append("\n");

}

} // Display information of all gadgets in the text area

private void makeCall() { // Method to make a call using the selected mobile gadget

int displayNumber = getDisplayNumber();

if (displayNumber != -1) {

Mobile mobile = (Mobile) gadgets.get(displayNumber);

String phoneNumber = getPhoneNumber();

int duration = getDuration();

mobile.makeCall(phoneNumber, duration);

}

} // Make a call using the selected mobile gadget

private void downloadMusic() { // Method to download music using the selected MP3 gadget

int displayNumber = getDisplayNumber();

if (displayNumber != -1) {

MP3 mp3 = (MP3) gadgets.get(displayNumber);

int downloadSize = getDownloadSize();

mp3.downloadMusic(downloadSize);

}

} // Download music using the selected MP3 gadget

public static void main(String[] args) { // Main method to create an instance of GadgetShopGUI

new GadgetShopGUI();

}

}