**Coursework Report**

Table Of Contents

[**1.GUI** 1](#_Toc165209916)

[**2.UML Class Diagrams** 2](#_Toc165209917)

[**3.Functionality** 3](#_Toc165209918)

[**4.Pseudocode** 16](#_Toc165209919)

[**5.Running my program from the terminal** 19](#_Toc165209920)

[**6.Testing** 20](#_Toc165209921)

[**7.Conclusion** 27](#_Toc165209922)

[**8. Appendix** 28](#_Toc165209923)

# **1.GUI**

I wanted to design a simple GUI for this gadget shop I decided to use a flow layout as I felt it gave me the most creativity and helped me to achieve the look I was going for. I wanted to keep it as simple as possible yet still with all the functions needed, this is what I imagined.

A screenshot of a computer

Description automatically generated

For my JFrame I have used FlowLayout, my JFrame is not resizable, my FlowLayout consists of 5 rows and a Y axis to position the different elements.

The GUI is called Gadget Shop. Model, Weight, Size, Credit, Memory, Phone, Duration, Download and Display Number are all text fields that have been integrated into this GUI. Make A Call, Clear, Add Mobile, Add MP3, Display All, Download Music, Download Help are all of the buttons I have used.

# **A screenshot of a computer Description automatically generated2.UML Class Diagrams**

A screenshot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA close-up of a screen

Description automatically generated**

# **3.Functionality**

A screenshot of a computer

Description automatically generatedAfter I was happy with the GUI I tackled the next part of the GadgetShop

as it was time to implement the functionality, I used ActionListener for handling the events that are in my program which happen to be all the buttons. I created an ActionPerformed method to be responsible for which method to be carried out as per the button press.

Each method will be explained here :

**clearFields()**

A screenshot of a computer code

Description automatically generatedThis method is used to clear all the text fields within the GUI of any text.

**makeCall()**

A screenshot of a computer

Description automatically generatedThis method takes a phone number from the phoneField, validates it to make sure its exactly 10 digits, it then does the same for call duration from the durationField then prints the phone number and call duration and clears the input fields afterwards.

**addMobile()**

This method is used to add a mobile device to the gadgets ArrayList, it takes values , model, size, price, credit and weight from the respective text fields parsing them into appropriate data types, any errors will be caught and incorrectInput() will be called, afterwards it creates a new Mobile object and adds it to the gadgets ArrayList, displays a success message, updates the index map and clears the fields.

A screenshot of a computer

Description automatically generated

**AddMP3()**

This method is used for adding an MP3 player to an ArrayList, it takes values model, size, price and memory from the respective text fields parsing the into the appropriate data types, any errors will be caught and incorrectInput() will be called, afterwards it creates a new MP3Player object and adds it to the gadget ArrayList, displays a success message, updates the index map and clears the fields.

A close-up of a computer screen

Description automatically generated

**displayAllGadgets()**

This method prints all the information about all of the gadgets stored in the ArrayList, if it is empty it prints “No gadgets added”. Otherwise it iterates over each gadget in the ArrayList and prints its model, price, weight and size. If the gadget is a mobile it will print the credit along with the other information, if it is an mp3 player it will print the memory. It also adds a line for visual separation.

A close-up of a colorful background

Description automatically generated

**downloadMusic()**

This method is responsible for downloading musicusing an MP3 player based on user input. It takes the gadget index from the user and converts it to an integer. It then locates the gadget using the index, if the gadget is found and it is an MP3 player it takes the user input for download size and starts the downloading process printing the memory size and model. If the gadget is not an MP3 player or the index is invalid, error messages are displayed.

A screenshot of a computer

Description automatically generated

**Gadget findGadgetByIndex(int index)**

This method searches for a gadget in the ArrayList based on its index. If the index is valid it retrieves and returns the gadget located at that index, if the index is invalid it returns null, indicating that no gadget was found at that index.

A screenshot of a computer code

Description automatically generated

**getGadgetByIndex(int index)**

This method gets a gadget from the ArrayList based on its index and displays the model. It calls findGadgetByIndex(index) to locate the gadget, if found it prints the model, if not it displays an error message.

A screenshot of a computer

Description automatically generated

**GadgetShop()**

This is the constructor, it initializes a GadgetShop object, it calls makeFrame() to setup the GUI then initializes the ArrayList gadgets to store all of the gadgets and a HashMap named “indexMap” to map gadget models to their indexes in the ArrayList.

A screenshot of a computer code

Description automatically generated

**makeFrame()**

This method is responsible for creating and configuring the GUI of the gadget shop. It sets the title of the frame to “Gadget Shop” and initializes the content pane with a vertical BoxLayout. It then adds

several rows of components and a button panel to the content pane using methods like addRow(). After it associated action listeners with all the buttons to handle the user interactions. Finally it packs the frame, sets it visible and configures it to exit the application when closed.

A screenshot of a computer program

Description automatically generated

**addRow1(Container contentPane)**

A screenshot of a code

Description automatically generatedThis method adds the first row of the components to the content pane of the frame. It creates a JPanel named “row1” and sets its layout to a left-aligned FlowLayout. It adds labels and text fields for model, price and weight each with a width to the “row1” panel. It then adds the “row1” panel to the content pane.

**addRow2(Container contentPane)**

This method adds the secind row of the components to the content pane of the frame. It creates a JPanel named “row2” and sets its layout to a left-aligned FlowLayout. It adds labels and text fields for size, credit and memory each with a width to the “row2” panel. It then adds the “row2” panel to the content pane.

A screenshot of a computer code

Description automatically generated

**addRow3(Container contentPane)**

This method adds the third row of the components to the content pane of the frame. It creates a JPanel named “row3” and sets its layout to a left-aligned FlowLayout. It adds labels and text fields for a phone number entry, along with a button labeled “Make A Call” to the row3 panel each with a width to the “row3” panel. It then adds the “row3” panel to the content pane.

A screenshot of a computer code

Description automatically generated

**addRow4(Container contentPane)**

This method adds the fourth row of components to the content pane of the frame. It creates a JPanel named “row4” and sets its layout to a left-aligned FlowLayout. It adds labels and text fields for duration and download each with a width to the “row4” panel. It then adds the “row4” panel to the content pane.

A screenshot of a computer program

Description automatically generated

**addRow5(Container contentPane)**

This method adds the fifth row of components to the content pane of the frame. It creates a JPanel named “row5” and sets its layout to a left-aligned FlowLayout. It adds labels and text fields for displaying a number to the “row5” panel. It then adds the “row5”panel to the content pane.

A screenshot of a computer code

Description automatically generated

**addButtonPanel(Container contentPane)**

A screenshot of a computer code

Description automatically generatedThis method adds a panel containing buttons to the content pane of the frame. It creates a JPanel named “buttonPanel” and sets its layout to a centered FlowLayout. It adds buttons labeled “Clear”, “Add Mobile”, “Add MP3”, “Display All”, “Download Music” and “Download Help” to the buttonPanel. It adds the buttonPanel to the content pane.

**Main(String[] args)**

This method is he entry point of the program. It creates a new GadgetShop object to be executed on the event dispatch thread using SwingUtilities.invokeLater().

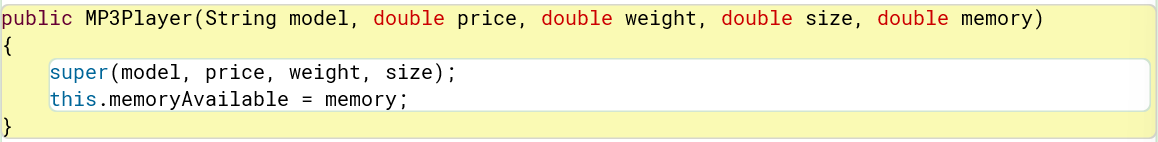
A screenshot of a computer program

Description automatically generated

As for GadgetShop.java that was all the methods, I will now move on to MP3Player

**MP3Player(String model, double price, double weight, double size, double memory)**

This is the constructor of the MP3Player class, it initializes an instance of the MP3Player class. It takes the parameters: model, price, weight, size and memory. The super() method is used to call the constructor of the superclass passing the model, price, weight and size to initialize the inherited properties, it also initializes the memoryAvailable property specific to the MP3Player class with the value of the memory parameter.



**downloadMusic(double downloadSize)**

This method allows downloading music to the MP3 player. It takes a parameter downloadSize, if the downloadSize is less than or equal to the memoryAvailable on the MP3 player, it then does the appropriate mathematics and prints “Music downloaded successfully”. Otherwise it prints an error message.

A screenshot of a computer

Description automatically generated

**getMemoryAvailable()**

This method returns the amount of available memory on the MP3 player.

A yellow and black sign

Description automatically generated

**setMemoryAvailable(double memoryAvailable)**

This method sets the amount of available memory on the MP3 player to the specified value, it allows updating the available memory of the MP3 player.

A yellow background with black text

Description automatically generated

**getdownloadSize()**

This method returns the size of the last music download performed.

A yellow box with black text

Description automatically generated

**setdownloadSize(double downloadSize)**

This method sets the size of the latest music download performed.

A yellow background with black text and black text

Description automatically generated

**getMemoryUsed()**

This method returns the amount of memory used on the MP3 player.

A yellow and black text

Description automatically generated

**setMemoryUsed(double memoryUsed)**

This method sets the amount of memory used on the MP3 player.

A yellow background with black text and black letters

Description automatically generated

**deleteMusic(double memoryToFree)**

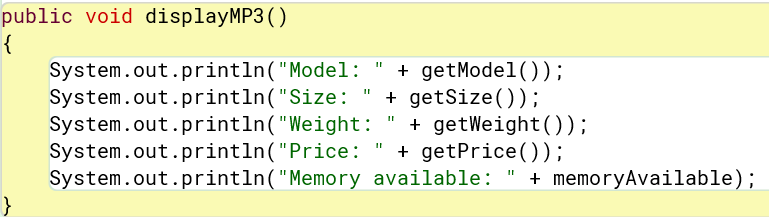
This method is used to delete music from the MP3 player. If memoryToFree is greater than 0 and less than or equal to the memoryUsed it subtracts memoryToFree from memoryUsed and adds it to memoryAvailable. If successful it prints a message indicating so.

A screenshot of a computer

Description automatically generated

**DisplayMP3()**

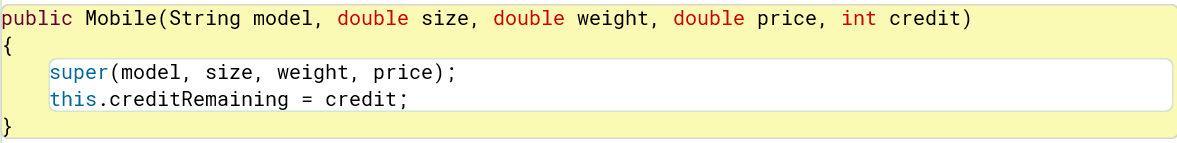
This method is used to display information about the MP3 player. It prints model, size, weight, price and memory available on the MP3 player.



As for the MP3Player.java that is all the methods, I will now move on to the Mobile.java

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

This is the constructor for the Mobile class, it initializes an instance of the Mobile class, it takes the parameters: model, size, weight, price and credit. The super() method is used to call the constructor of the superclass passing model, size, weight and price to initialize inherited properties. It also initializes the creditRemaining property specific to the Mobile class with the value of the credit parameter.



**getCreditRemaining()**

This method returns the amount of credit remaining on the mobile device.

A yellow and black text

Description automatically generated

For Mobile.java that is all the methods, I will know be moving on to the Gadget.java method.

**Gadget(String model, double price, double weight, double size)**

This is the constructor of the Gadget class it takes the parameters: model, price, weight and size. It assigns the values of these parameters to the corresponding properties of the Gadget class, model, price, weight and size.

A yellow and black text

Description automatically generated with medium confidence

**getModel()**

This method returns the model of the gadget.

A yellow box with black text

Description automatically generated

**setModel(String model)**

This method is used to set the model of the gadget.

A yellow box with black text

Description automatically generated

**getPrice()**

This method returns the price of the gadget.

A yellow box with black text

Description automatically generated

**setPrice(double price)**

This method is used to set the price of the gadget.

A yellow background with black text and black letters

Description automatically generated

**getWeight()**

This method returns the weight of the gadget.

A yellow background with black and red words

Description automatically generated

**setWeight(double weight)**

This method is used to set the weight of the gadget.

A yellow and black text

Description automatically generated

**getSize()**

This method returns the size of the gadget.

A yellow and black text

Description automatically generated

**setSize(double size)**

This method sets the size of the gadget.

A yellow background with black text and black text

Description automatically generated

# **4.Pseudocode**

**clearButton()**

A white text with black text

Description automatically generated

**makeCall()**

A white screen with black text

Description automatically generated

**addMobile()**

**A screenshot of a computer code

Description automatically generated**

**addMP3()**

**A screenshot of a computer program

Description automatically generated**

**displayAllGadgets()**

**A screenshot of a computer program

Description automatically generated**

**downloadMusic()**

**A white text with black text

Description automatically generated**

# **5.Running my program from the terminal**

A screenshot of a computer

Description automatically generated

The first step was to change the directory to the location of my project.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generatedI then compiled it.

I then successfully ran the program through the terminal.

# **6.Testing**

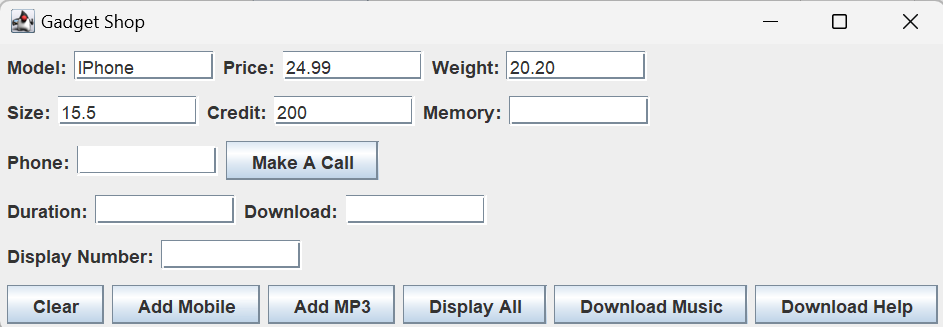
Now I move on to the testing phase to make sure everything works as intended.

Firstly the GUI displays as it should.

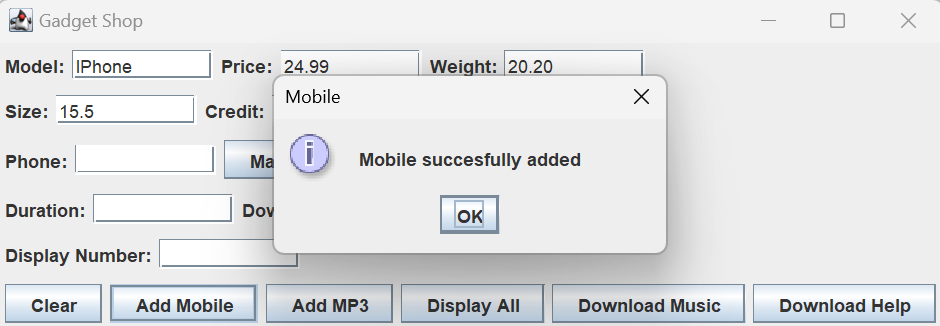
A screenshot of a computer

Description automatically generated

**addMobile()**

****

Firstly I input the specifications.

****

I then click Add Mobile and receive a pop up message confirming a mobile has been added to the array list.

**A screenshot of a computer

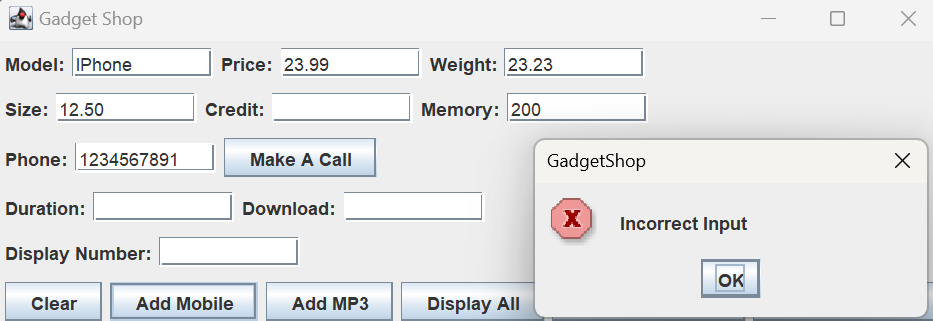
Description automatically generated**

Lastly it clears all the fields.

A screenshot of a computer

Description automatically generated

This is what happens when I have tried to input the wrong data type.



**AddMP3()**

When filling out a wrong field.

**AddMP3()**

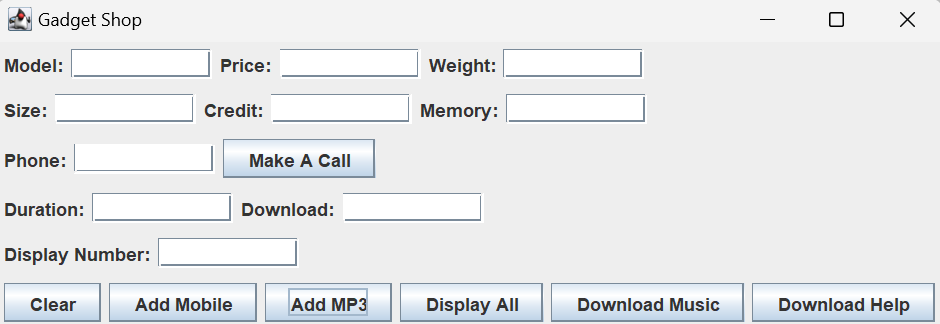
A screenshot of a computer

Description automatically generated

Firstly I input the specifications.

A screenshot of a computer program

Description automatically generated

I then click Add MP3 and receive a pop up message confirming an MP3 has been added to the array list.

Lastly it clears all fields.

A screenshot of a computer

Description automatically generated

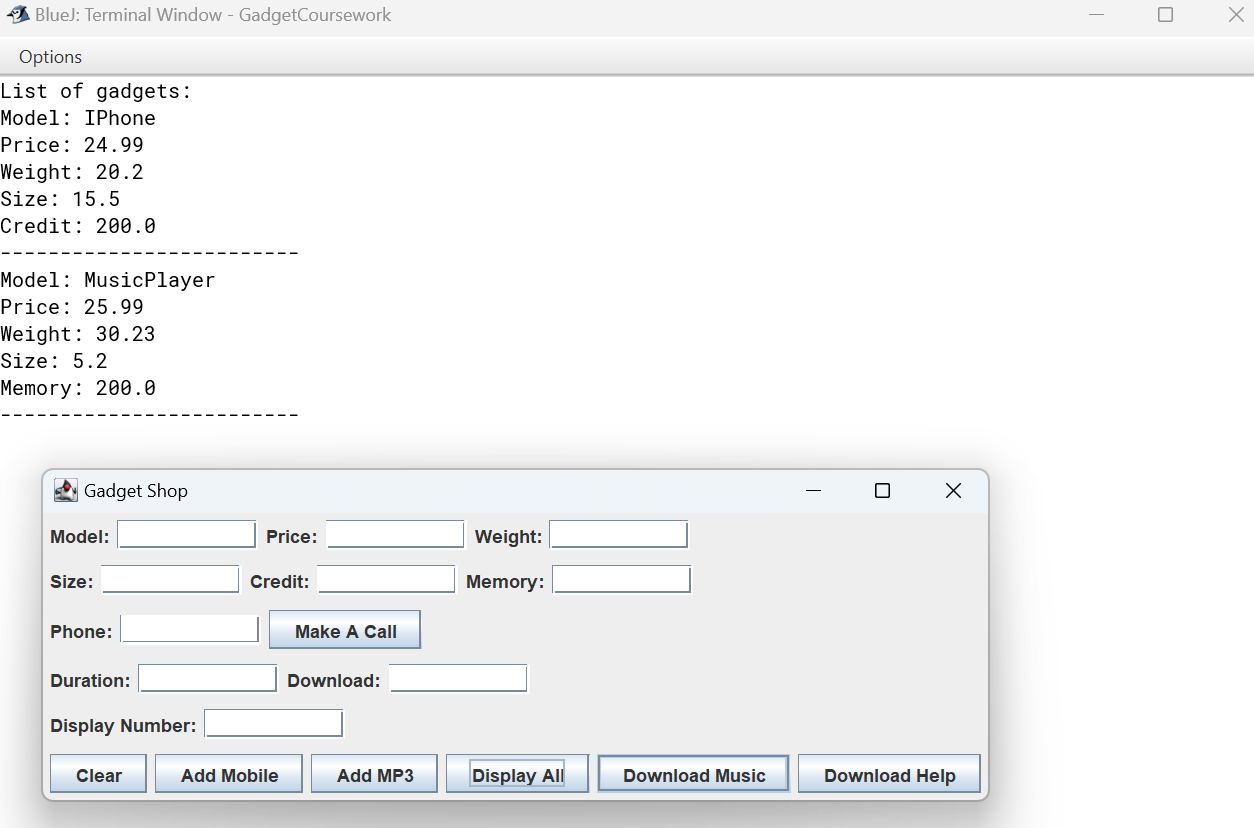
When trying to input the wrong data type.

A screenshot of a computer

Description automatically generated

When filling out a wrong field.

**displayAllGadgets()**

****

I click the Display All button and both the MP3 player and Mobile specifications are displayed.

**makeCall()**

A screenshot of a computer

Description automatically generated

Firstly I input the phone number and duration of the call.

A screenshot of a computer

Description automatically generated

A screenshot of a computer error

Description automatically generatedIt then prints the phone number and the duration of the call and clears the fields.

If the phone number is less than or longer than 10 digits.

A screenshot of a computer

Description automatically generated

If the wrong data type is entered.

**downloadMusic()**

A screenshot of a computer

Description automatically generated

Firstly I input the display number and download size

A screenshot of a computer

Description automatically generated

It then prints that the music is downloading using the given index from the user with the amount of memory, it also displays that it’s found the gadget.

A screenshot of a computer

Description automatically generated

If there isn’t a gadget found with the inputted index.

A screenshot of a computer

Description automatically generated

If the wrong data type is entered.

# **7.Conclusion**

In concluding this project, I am filled with a profound sense of accomplishment and growth. From the initial conception to the final implementation, the journey has been a testament to dedication, perseverance, and the power of continuous learning. Undertaking the project from start to finish provided me with a comprehensive understanding of the software development lifecycle, from requirement analysis to testing and deployment.

Throughout the process, I encountered a myriad of challenges, each offering an opportunity for learning and improvement. Among these challenges, I particularly struggled with using inheritance methods effectively, grappling with the nuances of superclass-subclass relationships and the implementation of polymorphism. Despite the initial setbacks, I approached these challenges with determination, seeking guidance from mentors, consulting resources, and engaging in hands-on experimentation. Through persistence and iterative refinement, I gradually gained proficiency in leveraging inheritance to enhance code reusability and maintainability.

Collaborating with peers and seeking guidance from mentors further enriched the experience, fostering a culture of shared learning and support. Their insights and feedback were invaluable in shaping the direction of the project and refining its outcomes.

Completing this project has not only enhanced my technical skills but also instilled a sense of confidence in my ability to tackle real-world problems. It has taught me the importance of resilience, adaptability, and perseverance in the face of challenges.

As I reflect on the journey, I am grateful for the opportunity to apply my knowledge and skills in a practical setting, and I am excited to carry forward the lessons learned to future endeavours. This project marks not just an endpoint, but a milestone in my journey as a software developer, propelling me forward with newfound confidence and determination.

# **8. Appendix**

**Gadget Class**

public class Gadget

{

private String model;

private double price;

private double weight;

private double size;

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

{

this.model = model;

this.price = price;

this.weight = weight;

this.size = size;

}

public String getModel()

{

return model;

}

public void setModel(String model)

{

this.model = model;

}

public double getPrice()

{

return price;

}

public void setPrice(double price)

{

this.price = price;

}

public double getWeight()

{

return weight;

}

public void setWeight(double weight)

{

this.weight = weight;

}

public double getSize()

{

return size;

}

public void setSize(double size)

{

this.size = size;

}

}

**Mobile Class**

public class Mobile extends Gadget

{

private int creditRemaining;

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

{

super(model, size, weight, price);

this.creditRemaining = credit;

}

public int getCreditRemaining()

{

return creditRemaining;

}

}

**MP3PPlayer class**

public class MP3Player extends Gadget {

private double memoryUsed;

private double memoryAvailable;

private double downloadSize;

public MP3Player(String model, double price, double weight, double size, double memory)

{

super(model, price, weight, size);

this.memoryAvailable = memory;

}

public void downloadMusic(double downloadSize)

{

if (downloadSize <= memoryAvailable)

{

memoryUsed += downloadSize;

memoryAvailable -= downloadSize;

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

} else {

System.out.println("Not enough memory available for download.");

}

}

public double getMemoryAvailable()

{

return memoryAvailable;

}

public void setMemoryAvailable(double memoryAvailable)

{

this.memoryAvailable = memoryAvailable;

}

public double getdownloadSize()

{

return downloadSize;

}

public void setdownloadSize(double downloadSize)

{

this.downloadSize = downloadSize;

}

public double getMemoryUsed()

{

return memoryUsed;

}

public void setMemoryUsed(double memoryUsed)

{

this.memoryUsed = memoryUsed;

}

public void deleteMusic(double memoryToFree)

{

if (memoryToFree > 0 && memoryToFree <= memoryUsed) {

memoryUsed -= memoryToFree;

memoryAvailable += memoryToFree;

System.out.println("Music deleted successfully. Memory freed: " + memoryToFree);

}

}

public void displayMP3()

{

System.out.println("Model: " + getModel());

System.out.println("Size: " + getSize());

System.out.println("Weight: " + getWeight());

System.out.println("Price: " + getPrice());

System.out.println("Memory available: " + memoryAvailable);

}

}

**GadgetShop Class**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.util.ArrayList;

import java.util.HashMap;

public class GadgetShop implements ActionListener

{

private JFrame frame;

private JTextField modelField;

private JTextField priceField;

private JTextField weightField;

private JTextField sizeField;

private JTextField creditField;

private JTextField memoryField;

private JTextField phoneField;

private JTextField durationField;

private JTextField downloadField;

private JTextField disnumberField;

private JButton downloadinfoButton;

private JButton clearButton;

private JButton addmobileButton;

private JButton addmp3Button;

private JButton displayallButton;

private JButton callButton;

private JButton downloadmusicButton;

private ArrayList<Gadget>gadgets;

private HashMap<String, Integer> indexMap;

public class Mobile extends Gadget

{

private double credit;

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

{

super(model, price, weight, size);

this.credit = credit;

}

public double getCredit()

{

return credit;

}

public void setCredit(double credit)

{

this.credit = credit;

}

}

public class MP3Player extends Gadget

{

private double memory;

public MP3Player(String model, double price, double weight, double size, double memory)

{

super(model, price, weight, size);

this.memory = memory;

}

public double getMemory()

{

return memory;

}

public void setMemory(double memory)

{

this.memory = memory;

}

}

public GadgetShop()

{

makeFrame();

gadgets = new ArrayList<>();

indexMap = new HashMap<>();

}

private void incorrectInput()

{

JOptionPane.showMessageDialog(frame,"Incorrect Input", "GadgetShop",JOptionPane.ERROR\_MESSAGE);

}

private void makeFrame()

{

frame = new JFrame("Gadget Shop");

Container contentPane = frame.getContentPane();

contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.Y\_AXIS));

addRow1(contentPane);

addRow2(contentPane);

addRow3(contentPane);

addRow4(contentPane);

addRow5(contentPane);

addButtonPanel(contentPane);

clearButton.addActionListener(this);

addmobileButton.addActionListener(this);

addmp3Button.addActionListener(this);

displayallButton.addActionListener(this);

downloadmusicButton.addActionListener(this);

callButton.addActionListener(this);

downloadinfoButton.addActionListener(this);

frame.pack();

frame.setVisible(true);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

private void addRow1(Container contentPane)

{

JPanel row1 = new JPanel();

row1.setLayout(new FlowLayout(FlowLayout.LEFT));

row1.add(new JLabel("Model:"));

row1.add(modelField = new JTextField(10));

row1.add(new JLabel("Price:"));

row1.add(priceField = new JTextField(10));

row1.add(new JLabel("Weight:"));

row1.add(weightField = new JTextField(10));

contentPane.add(row1);

}

private void addRow2(Container contentPane)

{

JPanel row2 = new JPanel();

row2.setLayout(new FlowLayout(FlowLayout.LEFT));

row2.add(new JLabel("Size:"));

row2.add(sizeField = new JTextField(10));

row2.add(new JLabel("Credit:"));

row2.add(creditField = new JTextField(10));

row2.add(new JLabel("Memory:"));

row2.add(memoryField = new JTextField(10));

contentPane.add(row2);

}

private void addRow3(Container contentPane)

{

JPanel row3 = new JPanel();

row3.setLayout(new FlowLayout(FlowLayout.LEFT));

row3.add(new JLabel("Phone:"));

row3.add(phoneField = new JTextField(10));

row3.add(callButton = new JButton("Make A Call"));

contentPane.add(row3);

}

private void addRow4(Container contentPane)

{

JPanel row4 = new JPanel();

row4.setLayout(new FlowLayout(FlowLayout.LEFT));

row4.add(new JLabel("Duration:"));

row4.add(durationField = new JTextField(10));

row4.add(new JLabel("Download:"));

row4.add(downloadField = new JTextField(10));

contentPane.add(row4);

}

private void addRow5(Container contentPane)

{

JPanel row5 = new JPanel();

row5.setLayout(new FlowLayout(FlowLayout.LEFT));

row5.add(new JLabel("Display Number:"));

row5.add(disnumberField = new JTextField(10));

contentPane.add(row5);

}

private void addButtonPanel(Container contentPane)

{

JPanel buttonPanel = new JPanel();

buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER));

buttonPanel.add(clearButton = new JButton("Clear"));

buttonPanel.add(addmobileButton = new JButton("Add Mobile"));

buttonPanel.add(addmp3Button = new JButton("Add MP3"));

buttonPanel.add(displayallButton = new JButton("Display All"));

buttonPanel.add(downloadmusicButton = new JButton("Download Music"));

buttonPanel.add(downloadinfoButton = new JButton("Download Help"));

contentPane.add(buttonPanel);

}

public static void main(String[] args)

{

SwingUtilities.invokeLater(new Runnable()

{

public void run()

{

new GadgetShop();

}

});

}

public void actionPerformed(ActionEvent e)

{

if (e.getSource() == clearButton)

{

clearFields();

}

else if (e.getSource() == callButton)

{

makeCall();

}

else if (e.getSource() == addmobileButton)

{

addMobile();

}

else if (e.getSource() == addmp3Button)

{

addMP3();

}

else if (e.getSource() == displayallButton)

{

displayAllGadgets();

}

else if (e.getSource() == downloadmusicButton)

{

downloadMusic();

String userInput = disnumberField.getText().trim();

try{

int index = Integer.parseInt(userInput);

getGadgetByIndex(index);

} catch (NumberFormatException ex)

{

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

}

}

else if (e.getSource() == downloadinfoButton)

{

JOptionPane.showMessageDialog(frame, "Enter the index for the required model in the 'Display Number' field ", "Help", JOptionPane.INFORMATION\_MESSAGE);

}

}

private void addMP3()

{

String model = modelField.getText();

double size;

double price;

double memory;

try {

size = Double.parseDouble(sizeField.getText());

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

memory = Double.parseDouble(memoryField.getText());

} catch (NumberFormatException e)

{

incorrectInput();

return;

}

double weight;

try {

weight = Double.parseDouble(weightField.getText());

} catch (NumberFormatException e)

{

incorrectInput();

return;

}

MP3Player mp3Player = new MP3Player(model, price, weight, size, memory);

gadgets.add(mp3Player);

JOptionPane.showMessageDialog(frame, "MP3 succesfully added","MP3", JOptionPane.INFORMATION\_MESSAGE);

indexMap.put(model, gadgets.size() - 1);

clearFields();

}

private void addMobile()

{

String model = modelField.getText();

double price;

double size;

double credit;

try {

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

size = Double.parseDouble(sizeField.getText());

credit = Double.parseDouble(creditField.getText());

} catch (NumberFormatException e)

{

incorrectInput();

return;

}

double weight;

try {

weight = Double.parseDouble(weightField.getText());

} catch (NumberFormatException e)

{

incorrectInput();

return;

}

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

gadgets.add(mobile);

JOptionPane.showMessageDialog(frame, "Mobile succesfully added","Mobile", JOptionPane.INFORMATION\_MESSAGE);

indexMap.put(model, gadgets.size() - 1);

clearFields();

}

private void makeCall()

{

String phoneNumber = phoneField.getText();

if (phoneNumber.isEmpty() || phoneNumber.length() != 10)

{

JOptionPane.showMessageDialog(frame, "Phone number must be exactly 10 digits", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

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

System.out.println("Calling: " + phoneNumber);

System.out.println("Duration is " + callDuration + " minutes");

clearFields();

}

private void displayAllGadgets()

{

if (gadgets.isEmpty())

{

System.out.println("No gadgets added");

} else

{

System.out.println("List of gadgets:");

for (Gadget gadget : gadgets)

{

System.out.println("Model: " + gadget.getModel());

System.out.println("Price: " + gadget.getPrice());

System.out.println("Weight: " + gadget.getWeight());

System.out.println("Size: " + gadget.getSize());

if (gadget instanceof Mobile)

{

Mobile mobile = (Mobile) gadget;

System.out.println("Credit: " + mobile.getCredit());

} else if (gadget instanceof MP3Player)

{

MP3Player mp3Player = (MP3Player) gadget;

System.out.println("Memory: " + mp3Player.getMemory());

}

System.out.println("-------------------------");

}

}

}

private void downloadMusic()

{

String userInput = disnumberField.getText().trim();

try {

int index = Integer.parseInt(userInput);

Gadget gadget = findGadgetByIndex(index);

if (gadget != null)

{

if (gadget instanceof MP3Player)

{

MP3Player mp3Player = (MP3Player) gadget;

String downloadInput = downloadField.getText().trim();

if (!downloadInput.isEmpty())

{

try {

double downloadSize = Double.parseDouble(downloadInput);

System.out.println("Downloading music using " + mp3Player.getModel() + " with " + downloadSize + " memory usage.");

} catch (NumberFormatException e)

{

JOptionPane.showMessageDialog(frame, "Invalid input for memory usage. Please enter a valid number.", "Error", JOptionPane.ERROR\_MESSAGE);

}

} else

{

JOptionPane.showMessageDialog(frame, "Please enter the amount of memory to use for downloading music.", "Error", JOptionPane.ERROR\_MESSAGE);

}

} else

{

JOptionPane.showMessageDialog(frame, "Selected gadget is not an MP3 player.", "Error", JOptionPane.ERROR\_MESSAGE);

}

} else

{

JOptionPane.showMessageDialog(frame, "Invalid index or gadget not found.", "Error", JOptionPane.ERROR\_MESSAGE);

}

} catch (NumberFormatException ex)

{

JOptionPane.showMessageDialog(frame, "Please enter a valid integer for the gadget index.", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

private Gadget findGadgetByIndex(int index)

{

if (index >=0 && index < gadgets.size())

{

return gadgets.get(index);

} else

{

return null;

}

}

private void getGadgetByIndex(int index)

{

Gadget gadget = findGadgetByIndex(index);

if(gadget != null)

{

System.out.println("Gadget found: " + gadget.getModel());

} else

{

JOptionPane.showMessageDialog(frame, "Invalid index or gadget not found.", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

private void clearFields()

{

modelField.setText("");

priceField.setText("");

weightField.setText("");

sizeField.setText("");

creditField.setText("");

memoryField.setText("");

phoneField.setText("");

durationField.setText("");

downloadField.setText("");

disnumberField.setText("");

}

}