***Module: CC4001 Programming***

***Module Leader: Sandra Fernando***

***Student Name: Ryan Whitelam***

***Student ID: 21007789***

***Github Link: https://github.com/RyanWhitelam/RyanWhitelam.git***

Table of Contents

[Introduction 3](#_Toc165084485)

[Class Diagram 3](#_Toc165084486)

[Method Descriptions 4](#_Toc165084487)

[Pseudocode for Button-Handling Methods 16](#_Toc165084488)

[Button: Add Mobile 16](#_Toc165084489)

[Button: Add MP3 17](#_Toc165084490)

[Button: Clear 17](#_Toc165084491)

[Button: Display All Gadgets 17](#_Toc165084492)

[Button: Make Call 17](#_Toc165084493)

[Button: Download Music 17](#_Toc165084494)

[Textboxes and Input Check 18](#_Toc165084495)

[Buttons and Action Performed Methods 18](#_Toc165084496)

[Testing 18](#_Toc165084497)

[Error Detection and Correction 19](#_Toc165084498)

[Conclusion 19](#_Toc165084499)

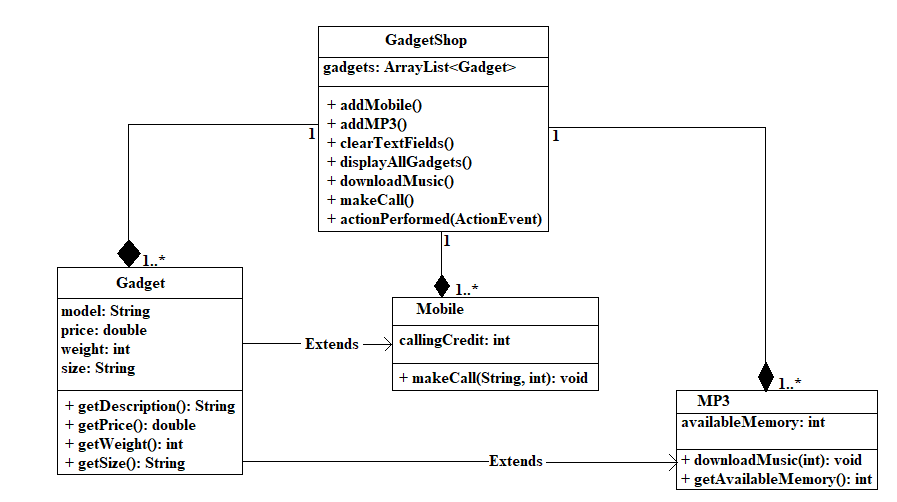
[Appendix 22](#_Toc165084500)

# Introduction

The GadgetShop application offers a solution spanning from managing gadgets and applying a user-friendly graphical user interface. The application assigns priority to the users and allows them to carry out the different. At the centre of GadgetShop, it has a range of smart devices which are designed to streamline your gadget management experience through a set of easy-to-use features. In a nutshell, GadgetShop is an application that is user-oriented, which combines functionality, simplicity and user-friendliness to achieve a smooth. 

Class Diagram

The framework is the GadgetShop class, which runs as the gadget centre for control. This class presents the features which are responsible for the manipulation of the devices like adding new devices to the drawer, showing the details, making calls. Gadget class is a base class from where all gadget’s objects are derived. It carries essential variables common to all gadgets, such as model, price, weight, and size. This class provides superclass for particular gadget types including the likes of Smartphones and MP3 Players. Subclasses Mobile, MP3 and Gadget come from the Gadget class, which provides their attributes and methods and introduces the device-specific functionality for the MP3 and cell phone devices.

 *Class Diagram*

# Method Descriptions

In the Gadget.java class, a set of methods is used to take care of the overall gadget properties. The getDescription( ) method gives the description that acts as the overview of the gadget and the price is retrieved through getPrice( ) which facilitates the evaluation of the price. Apart from that, getWeight( ) function enables a user to obtain weight of the device, making it possible to bring a user to consider the issues related to portability and handling. In the same way, getSize( ) method helps in determining the size of gadget needed, which gives the users an idea of compatibility and the space needed as well. In Mobile.java, inheritance from the Gadget class is used to extend Mobile.java. The methods specially designed for mobile phones are presented. This makeCall (String phoneNumber, int duration) method makes it possible for users to place calls by passing the phone number and the call duration. The introduction of this feature confers great practicality on mobile phones, making them suitable instruments for communication. MP3 is extended by MP3.java class and equipped with MP3 player specific functionalities. With the implementation of the downloadMusic (int downloadSize) method, music is now able to be downloaded into the MP3 player. In addition, getAvailableMemory ( ) method lets users have the memory information of the MP3 player, which is necessary for achieving effective management of the storage space.

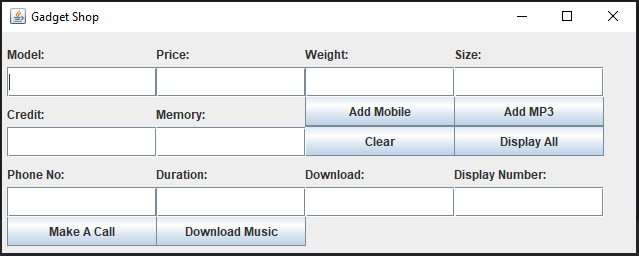


Figure 1: GUI Visualization

Figure 1, shows the initial GUI look after executing the program.

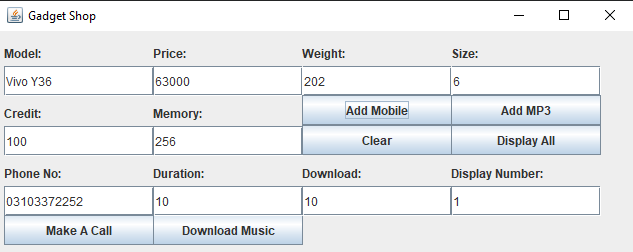


Figure 2: Filling Details

This Figure displays a GUI of “Gadget Shop” with filled values, entered by the user.

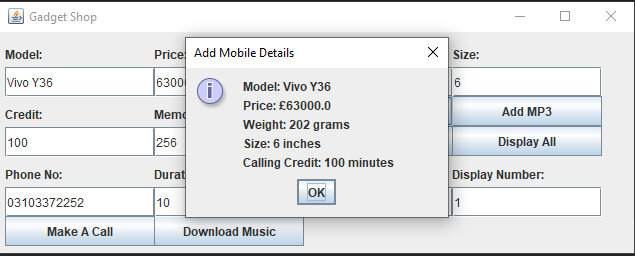


Figure 3:Result of Add Button

This figure demonstrates that by clicking on “Add Button” after entering all values this button adds the details of mobile.

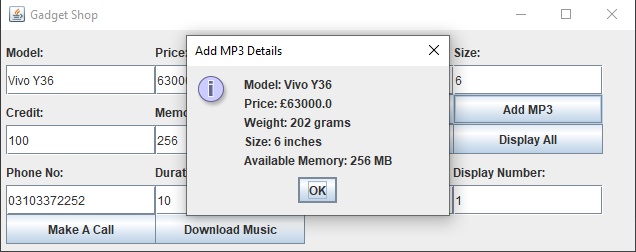


Figure 4: Result of MP3 Button

This figure shows that by clicking on “MP3 Button” after entering all necessary values this button adds the MP3 details.

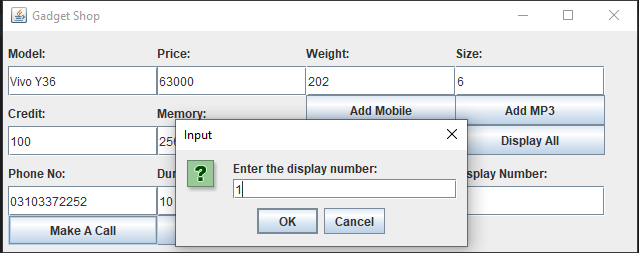


Figure 5: Display Number for Call Button

By clicking on “Make A Call” button it will ask to enter a display number.

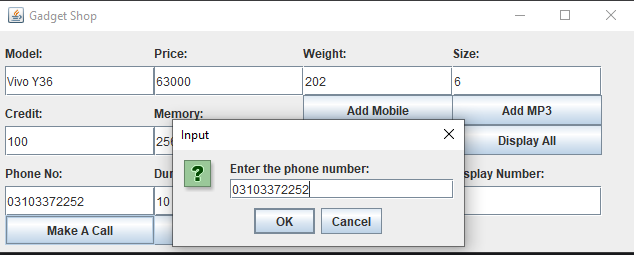


Figure 6: Enter Phone Number

This Figure displays a “Dialogue Box” ask to enter a phone number.

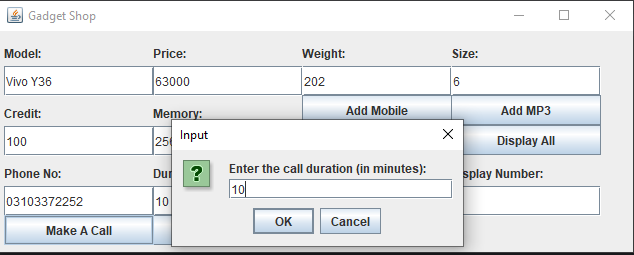
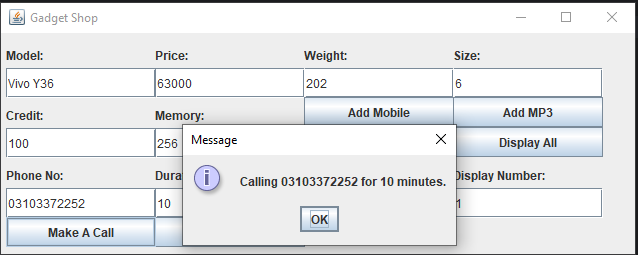


Figure 7: Call Duration

This Figure displays a “Dialogue Box” ask to enter a call duration in minutes.



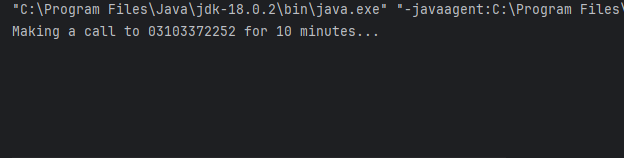


Figure 8: Calling Result

This shows the result of Calling on Command Prompt as well as on Dialogue Box. This will display the calling number with number of minutes for call.

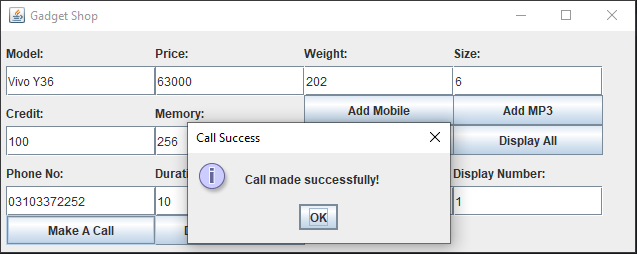


Figure 9: Message of Call Success

Once made a call after it will display a message of successful call made.

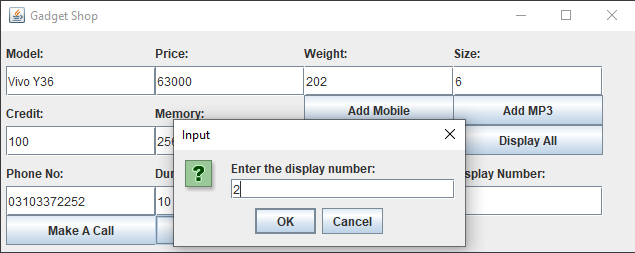


Figure 10: Enter Display Number for Download Music

By clicking on “Download Music” button it will ask to enter a display number.

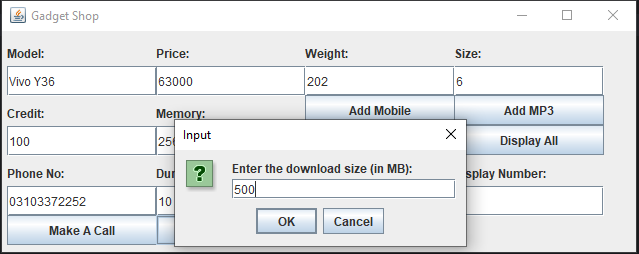
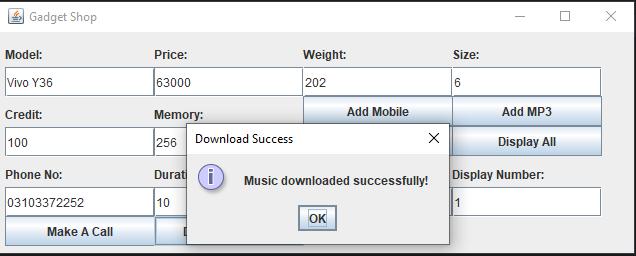


Figure 11: Enter Download Size

This will ask to enter a download size of music.



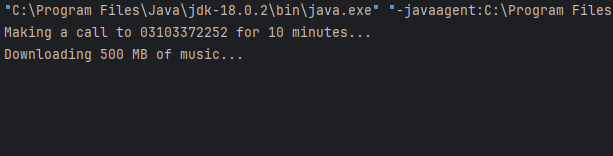


Figure 12: Message for Successful Download Music

This displays a message once the music downloads successfully. This message will be displayed on “Dialog Box” as well as on “Command Prompt”.

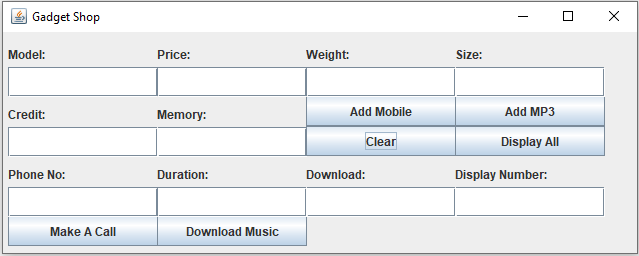


Figure 13: Clear Button

By clicking on “Clear Button” will clear all the Text Field input data.

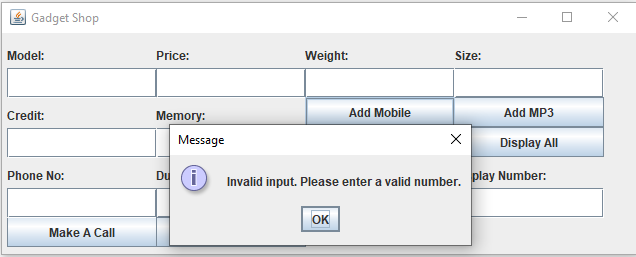


Figure 14: Testing Add Mobile Button by Keeping Field Empty

Again, executes a program to examine the further operations, if Text fields kept empty and “Add Mobile” buttons get clicked it will display a following message.

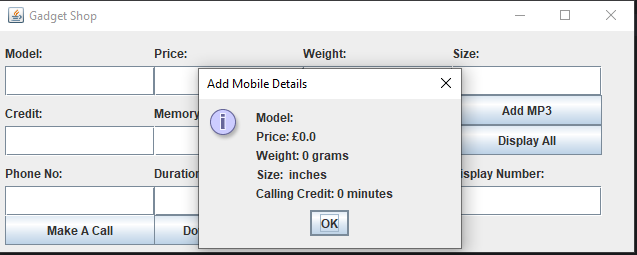


Figure 15: Zero Input Values

Due to entering empty text fields, it will display an attribute with zero value.

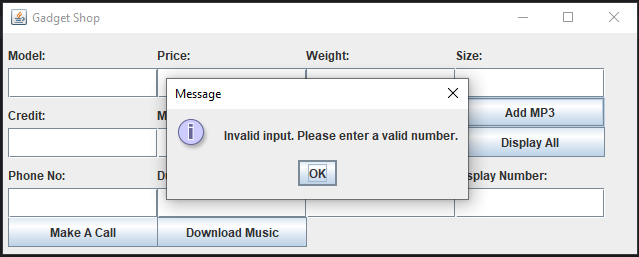


Figure 16: Testing Add MP3 Button by Keeping Field Empty

This shows if Text fields kept empty and “Add MP3” buttons get clicked it will display a following message.

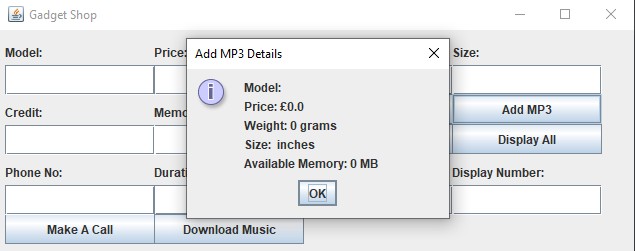
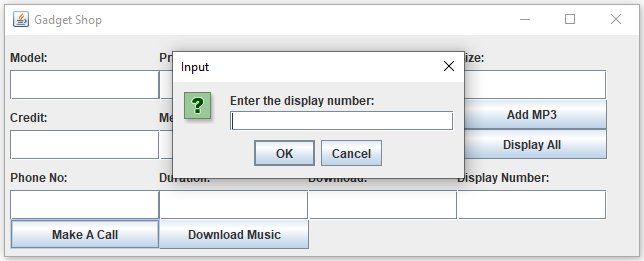


Figure 17: Zero Input Values

By entering empty text fields, it will display an attribute with zero value.



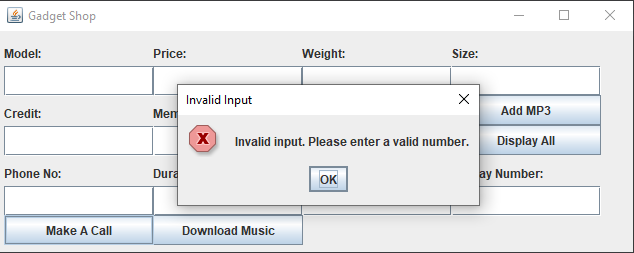
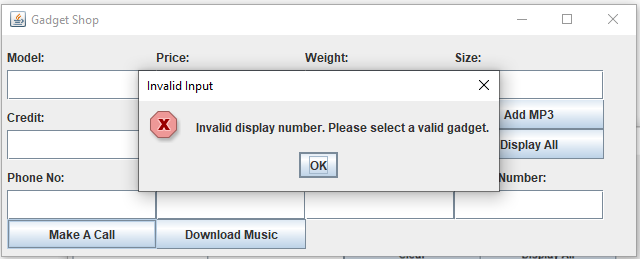
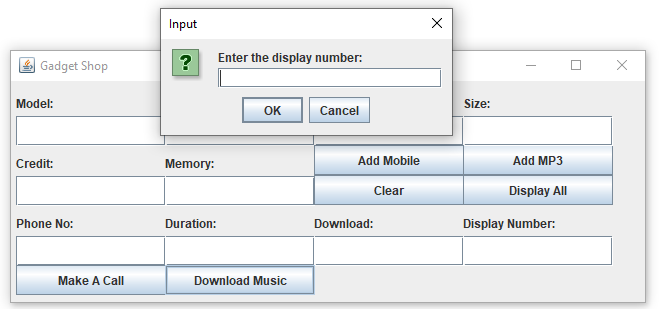


Figure 18: Empty Display Number

By clicking on “Make A Call” button will open a dialog box that will ask to enter a display number, by clicking on “Ok” button without entering a display number text field will give the following error message.



By entering invalid input in display number of “Make A Call” button will display a following message.



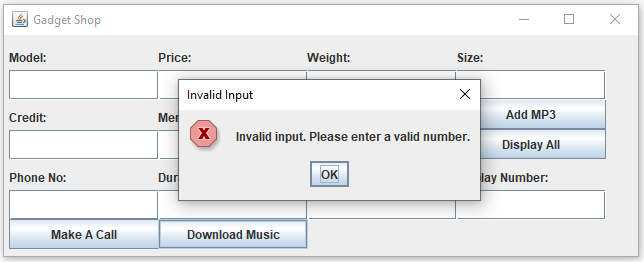


Figure 19: Empty Display Number

By clicking on “Download Music” button will open a dialog box that will ask to enter a display number, by clicking on “Ok” button without entering a display number text field will give the following error message.

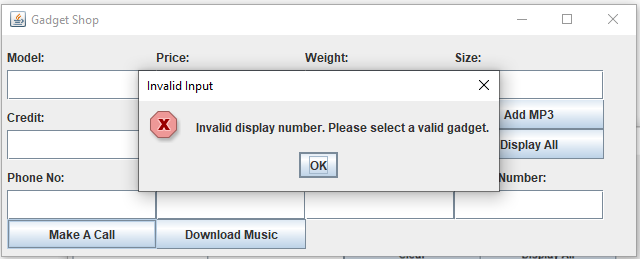


Figure 20: Invalid Input

By entering invalid input in display number of “Download Music” button will display a following message.

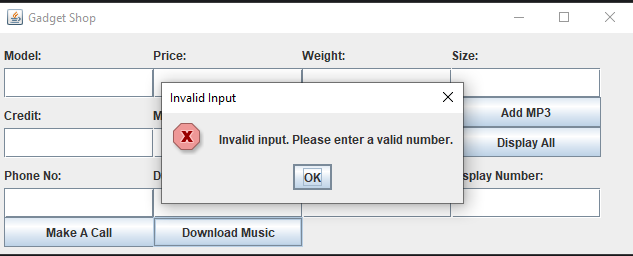


Figure 21 Testing Add MP3 Button by Keeping Field Empty

This shows if Text fields kept empty and “Display All” buttons get clicked it will display a following message.

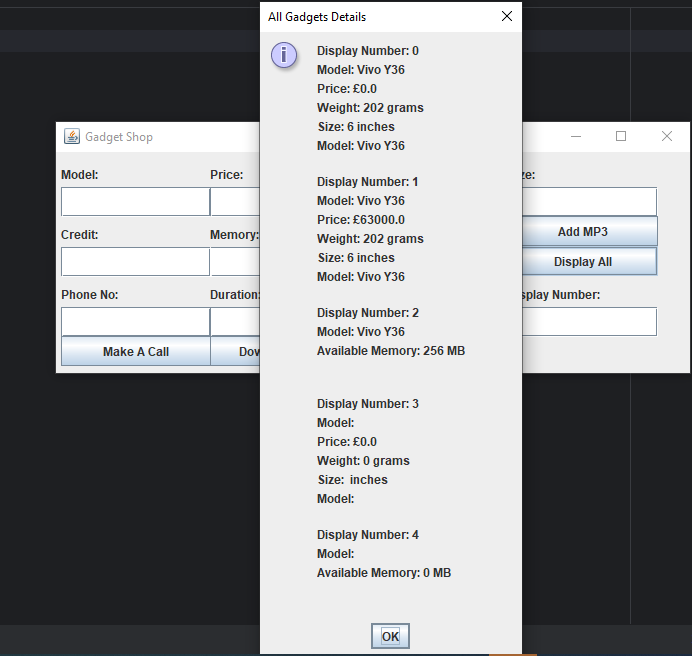


Figure 22: Current details with Zero input values

By entering empty text fields and then clicking on “Display All” button will display all previous attributes and their values details including current details with zero value.

# Pseudocode for Button-Handling Methods

## Button: Add Mobile

Read mobile details from text fields

Create new Mobile object with details

Add Mobile object to GadgetShop list

Display success message

## Button: Add MP3

Read MP3 details from text fields

Create new MP3 object with details

Add MP3 object to GadgetShop list

Display success message

## Button: Clear

Clear all text fields in the GUI

## Button: Display All Gadgets

Iterate over GadgetShop list

Display details of each gadget in a dialog box

## Button: Make Call

Read display number from text field

Retrieve corresponding gadget from GadgetShop list

If gadget is a Mobile, prompt for phone number and duration

Initiate call using Mobile method

Display success/error message

## Button: Download Music

Read display number from text field

Retrieve corresponding gadget from GadgetShop list

If gadget is an MP3 player, prompt for download size

Download music using MP3 method

Display success/error message

# Textboxes and Input Check

The GadgetShop application’s GUI (graphical user interface) features the fields of text as the means of input where users can provide different details for managing gadgets. These text fields allow users to type information like gadget details (model, price, weight and sizes), phone numbers, call duration, download size and displays. The application for a device would be checked not only for these attributes such as price, weight, and calling credit but also whether they are numeric value or not. Likewise, the application such as a phone call or downloading music will validate the input of the phone number, duration, and download size whether they are valid and within the reasonable limits.

# Buttons and Action Performed Methods

Users are guided through the GUI of the GadgetShop app by using the buttons that relate to different actions and functions of the application. These buttons are configured to perform particular functions like gadget addition, calling, music download, and gadget details display.

All GUI buttons are mapped to ActionPerformed methods, which in turn contain the implementation to execute the related action once clicked on the button. As for an example, clicking the "Add Mobile" button initiates the ActionPerformed method that implements the adding mobile gadget to the GadgetShop. Also, clicking the "Make Call" button generates the ActionPerformed method for processing a call. To provide users with a convenient and hands-on GUI, the GadgetShop application integrates buttons and associates those with ActionPerformed methods. It helps the customers to perform various tasks smoothly and provides a friendly interface throughout the entire application.  
  
Testing

Testing is a crucial phase in software development in which the aim is to realize if the application functions correctly. The GadgetShop application undergoes testing in order to certify that it features all the range of functions such as adding gadgets, displaying details, making calls, and downloading music. The snapshots are taken to record that the gadgets have been successfully added as well as to be sure that the details are correctly presented and shown in the application widget. The displaying of gadget info is tested by confirming that markup of all the added gadgets in the GadgetShop shows up correctly and their attributes show up correctly as well. Snapshots are taken and saved to display the capability of the machine such model, price, weight, size and the feature specific to mobile phones or MP3 Players. The testing of the functions calls on mobile phones that have been added to the GadgetShop should be carried out to confirm that the call is made successfully using that phone number and duration. The screenshots are taken as verification that the call is successfully placed and any comments or dialogs that are presented to the user are within the scope of the application's functionality. Likewise, music downloading function will be tested installing music to MP3 players on the GadgetShop and collecting bugs in downloading process. Snapshots are taken in order to record completion of music files downloading and status of progress indicators or any dialog confirmation box.

# Error Detection and Correction

The actual process of implementing the GadgetShop application has, among others, revealed some syntax errors and runtime issues. One had to check for missing semicolons at the end of lines of the code, so that any compilation errors could be detected. The missing semicolons were easily identified through compiler errors and was to insert them at the right locations. In addition, runtime errors sprang from user inputs testing. Error mitigation was done by implementing index bounds checking to restrict the usage to only valid indices. Through the process of recording these errors along with the taken actions for their correction, the team of developers was able to stay in a systematic way of discovering and fixing such errors, eventually contributing to the application of GadgetShop attaining enhanced reliability.

# Conclusion

The GadgetShop Application has made progress in gadget management. It offers the user with a user-friendly interface and a range of functionalities. From the beginning of the development process, I have been constantly reinforced by various types of information, which helped to fill in the gaps and increase my knowledge. Firstly, the use of Graphical User Interface (GUI) created the opportunity to understand theoretical concepts of GUI programming. In addition, the application strictly adhered to object-oriented design principles, which enabled it to be modular and reusable, and hence maintainable. The program is tested until it does not give any semantic nor syntax errors, and runtime as well. They will provide us with the tools and techniques which will use to achieve the goal of creating user-friendly applications.

# Appendix

// Class Gadget  
class Gadget {  
  
 //Private attributes  
 private String model;  
 private double price;  
 private int weight;  
 private String size;  
  
 // Constructor to initialize the gadget  
 public Gadget(String model, double price, int weight, String size) {  
 this.model = model;  
 this.price = price;  
 this.weight = weight;  
 this.size = size;  
 }  
  
 // Getter methods to retrieve the model of the gadget  
 public String getModel() {  
  
 return model;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
  
 public int getWeight() {  
 return weight;  
 }  
  
 public String getSize() {  
 return size;  
 }  
 // Override toString method for string representation of the gadget  
 @Override  
 public String toString() {  
 return "Model: " + model + "\n" +  
 "Price: £" + price + "\n" +  
 "Weight: " + weight + " grams\n" +  
 "Size: " + size + " inches\n";  
 }  
}

import javax.swing.\*;  
  
// Mobile class extending Gadget attributes  
class Mobile extends Gadget {  
 private String model;  
 private int callingCredit;  
  
 // Constructor  
 public Mobile(String model, double price, int weight, String size, int callingCredit) {  
 super(model, price, weight, size);  
 this.model = model;  
 this.callingCredit = callingCredit;  
 }  
  
 // Method to make a call  
 public void makeCall(String phoneNumber, int duration) {  
 System.*out*.println("Making a call to " + phoneNumber + " for " + duration + " minutes...")  
  
 if (callingCredit >= duration) {  
 JOptionPane.*showMessageDialog*(null, "Calling " + phoneNumber + " for " + duration + " minutes.");  
 callingCredit -= duration;  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Insufficient credit to make the call.");  
 }  
 }  
 // Override toString method   
 @Override  
 public String toString() {  
 return super.toString() +  
 "Model: " + model + "\n";  
 }  
}

import java.io.Serializable;  
  
// Define the MP3 class which extends Gadget and implements Serializable interface  
class MP3 extends Gadget implements Serializable {  
  
 // Private attributes  
 private String model;  
 private int availableMemory;  
  
 // Constructor  
 public MP3(String model, double price, int weight, String size, int availableMemory) {  
 super(model, price, weight, size);  
 this.model = model;  
 this.availableMemory = availableMemory;  
 }  
 // Getter method to retrieve the available memory of the MP3 player  
 public int getAvailableMemory() {  
 return availableMemory;  
 }  
  
 // download music method of MP3 class  
 public void downloadMusic(int downloadSize) {  
 System.*out*.println("Downloading " + downloadSize + " MB of music...");  
 // Code to download music goes here  
 }  
  
 // Override toString method   
 @Override  
 public String toString() {  
 return "Model: " + model + "\n" +  
 "Available Memory: " + availableMemory + " MB\n\n";  
 }  
}

import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.util.ArrayList;  
  
// Define the GadgetShop class which extends JFrame and implements ActionListener  
public class GadgetShop extends JFrame implements ActionListener {  
  
 // ArrayList to store gadgets  
 public ArrayList<Gadget> gadgets = new ArrayList<>();  
  
 // Text fields for input  
 private JTextField textModel, textPrice, textWeight, textSize, textCredit, textMemory,  
 textPhoneNo, textDuration, textDownload, textDisplayNumber;  
  
 // Labels for text fields  
 private JLabel labelModel, labelPrice, labelWeight, labelSize, labelCredit, labelMemory, labelPhoneNo,  
 labelDuration, labelDownload, labelDisplayNumber;  
  
 // Buttons for actions  
 private JButton btnAddMobile;  
 private JButton btnAddMP3;  
 private JButton btnClear;  
 private JButton btnDisplayAll;  
 private JButton btnMakeCall;  
 private JButton btnDownloadMusic;  
  
 // Constructor  
 public GadgetShop() {  
  
 super("Gadget Shop");  
 // defining a frame  
 JFrame frame = new JFrame();  
  
 // Initialization of GUI components  
 textModel = new JTextField();  
 textPrice = new JTextField();  
 textWeight = new JTextField();  
 textSize = new JTextField();  
 textCredit = new JTextField();  
 textMemory = new JTextField();  
 textPhoneNo = new JTextField();  
 textDuration = new JTextField();  
 textDownload = new JTextField();  
 textDisplayNumber = new JTextField();  
  
  
 //Labels for text field  
 labelModel = new JLabel("Model:");  
 labelPrice = new JLabel("Price:");  
 labelWeight = new JLabel("Weight:");  
 labelSize = new JLabel("Size:");  
 labelCredit = new JLabel("Credit:");  
 labelMemory = new JLabel("Memory:");  
 labelPhoneNo = new JLabel("Phone No:");  
 labelDuration = new JLabel("Duration:");  
 labelDownload = new JLabel("Download:");  
 labelDisplayNumber = new JLabel("Display Number:");  
  
 // action listeners for buttons  
 btnAddMobile.addActionListener(this);  
 btnAddMP3.addActionListener(this);  
 btnClear.addActionListener(this);  
 btnDisplayAll.addActionListener(this);  
 btnMakeCall.addActionListener(this);  
 btnDownloadMusic.addActionListener(this);  
  
 //Panel to add text field  
 JPanel panel = new JPanel();  
 panel.add(textModel);  
 panel.add(textPrice);  
 panel.add(textWeight);  
 panel.add(textSize);  
 panel.add(textCredit);  
 panel.add(textMemory);  
 panel.add(textPhoneNo);  
 panel.add(textDuration);  
 panel.add(textDownload);  
 panel.add(textDisplayNumber);  
 panel.add(labelModel);  
 panel.add(labelPrice);  
 panel.add(labelWeight);  
 panel.add(labelSize);  
 panel.add(labelCredit);  
 panel.add(labelMemory);  
 panel.add(labelPhoneNo);  
 panel.add(labelDuration);  
 panel.add(labelDownload);  
 panel.add(labelDisplayNumber);  
 frame.add(panel);  
  
  
 // Add buttons to panel  
 panel.add(btnAddMobile);  
 panel.add(btnAddMP3);  
 panel.add(btnClear);  
 panel.add(btnDisplayAll);  
 panel.add(btnMakeCall);  
 panel.add(btnDownloadMusic);  
  
  
 labelModel.setBounds(5, 10, 150, 25);  
 textModel.setBounds(5, 35, 150, 30);  
  
 labelPrice.setBounds(154, 10, 150, 25);  
 textPrice.setBounds(154, 35, 150, 30);  
  
 labelWeight.setBounds(303, 10, 150, 25);  
 textWeight.setBounds(303, 35, 150, 30);  
  
 labelSize.setBounds(452, 10, 150, 25);  
 textSize.setBounds(452, 35, 150, 30);  
  
 labelCredit.setBounds(5, 70, 150, 25);  
 textCredit.setBounds(5, 95, 150, 30);  
  
 labelMemory.setBounds(154, 70, 150, 25);  
 textMemory.setBounds(154, 95, 150, 30);  
  
 labelPhoneNo.setBounds(5, 130, 150, 25);  
 textPhoneNo.setBounds(5, 155, 150, 30);  
  
 labelDuration.setBounds(154, 130, 150, 25);  
 textDuration.setBounds(154, 155, 150, 30);  
  
 labelDownload.setBounds(303, 130, 150, 25);  
 textDownload.setBounds(303, 155, 150, 30);  
  
 labelDisplayNumber.setBounds(452, 130, 150, 25);  
 textDisplayNumber.setBounds(452, 155, 150, 30);  
  
 btnAddMobile.setBounds(303, 64, 150, 30);  
 btnAddMP3.setBounds(452, 64, 150, 30);  
 btnClear.setBounds(303, 94, 150, 30);  
 btnDisplayAll.setBounds(452, 94, 150, 30);  
 btnMakeCall.setBounds(5, 184, 150, 30);  
 btnDownloadMusic.setBounds(154, 184, 150, 30);  
  
 // Set the panel as content pane  
 setContentPane(panel);  
  
 // properties to set size, layout etc  
 setSize(650, 260);  
 setLayout(null);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLocationRelativeTo(null);  
 setVisible(true);  
 }  
 // Action Listener for buttons actions  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if (e.getSource() == btnAddMobile) {  
 addMobile();  
 } else if (e.getSource() == btnAddMP3) {  
 addMP3();  
 } else if (e.getSource() == btnClear) {  
 clearTextFields();  
 } else if (e.getSource() == btnDisplayAll) {  
 displayAllGadgets();  
 } else if (e.getSource() == btnMakeCall) {  
 makeCall();  
 } else if (e.getSource() == btnDownloadMusic) {  
 downloadMusic();  
 }  
 }  
 // Method to add a mobile gadget  
 private void addMobile() {  
 try {  
 String model = textModel.getText();  
 double price = getDoubleValue(textPrice);  
 int weight = getIntegerValue(textWeight);  
 String size = textSize.getText();  
 int credit = getIntegerValue(textCredit);  
 Mobile mobile = new Mobile(model, price, weight, size, credit);  
 gadgets.add(mobile);  
 String message = "Model: " + model + "\n" +  
 "Price: £" + price + "\n" +  
 "Weight: " + weight + " grams\n" +  
 "Size: " + size + " inches\n" + "Calling Credit: " + credit + " minutes";  
 JOptionPane.*showMessageDialog*(null, message, "Add Mobile Details", JOptionPane.*INFORMATION\_MESSAGE*);  
 } catch (NumberFormatException e) {  
 // Display error message for invalid input  
 JOptionPane.*showMessageDialog*(null, "Invalid input. Please enter a valid integer.");  
 }  
 }  
 // Method to add an MP3 gadget  
 private void addMP3() {  
 try {  
 String model = textModel.getText();  
 double price = getDoubleValue(textPrice);  
 int weight = getIntegerValue(textWeight);  
 String size = textSize.getText();  
 int memory = getIntegerValue(textMemory);  
 MP3 mp3 = new MP3(model, price, weight, size, memory);  
 gadgets.add(mp3);  
 String message = "Model: " + model + "\n" +  
 "Price: £" + price + "\n" +  
 "Weight: " + weight + " grams\n" +  
 "Size: " + size + " inches\n" +  
 "Available Memory: " + memory + " MB";  
 JOptionPane.*showMessageDialog*(null, message, "Add MP3 Details", JOptionPane.*INFORMATION\_MESSAGE*);  
 } catch (NumberFormatException e) {  
 // Display error message for invalid input  
 JOptionPane.*showMessageDialog*(null, "Invalid input. Please enter a valid integer.");  
 }  
 }  
  
 // Method to clear text fields  
 private void clearTextFields() {  
 textModel.setText("");  
 textPrice.setText("");  
 textWeight.setText("");  
 textSize.setText("");  
 textCredit.setText("");  
 textMemory.setText("");  
 textPhoneNo.setText("");  
 textDuration.setText("");  
 textDownload.setText("");  
 textDisplayNumber.setText("");  
 }  
 // Method to display all gadgets  
 public void displayAllGadgets() {  
 // Method to display all gadgets  
 StringBuilder allGadgetsDetails = new StringBuilder();  
 for (int i = 0; i < gadgets.size(); i++) {  
 allGadgetsDetails.append("Display Number: ").append(i).append("\n");  
 allGadgetsDetails.append(gadgets.get(i).toString()).append("\n");  
 }  
 JOptionPane.*showMessageDialog*(null, allGadgetsDetails.toString(), "All Gadgets Details", JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
  
 // Method to get the download size via GUI input  
 private int getDownloadSize() {  
 return Integer.*parseInt*(JOptionPane.*showInputDialog*(null, "Enter the download size (in MB):"));  
 }  
  
 // Method to handle downloading music  
 private void downloadMusic() {  
 try {  
 int displayNumber = getDisplayNumber();  
 if (displayNumber >= 0 && displayNumber < gadgets.size()) {  
 Gadget selectedGadget = gadgets.get(displayNumber);  
 if (selectedGadget instanceof MP3) {  
 int downloadSize = getDownloadSize();  
 ((MP3) selectedGadget).downloadMusic(downloadSize);  
 JOptionPane.*showMessageDialog*(null, "Music downloaded successfully!", "Download Success", JOptionPane.*INFORMATION\_MESSAGE*);  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Selected gadget is not an MP3 player.", "Invalid Gadget", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Invalid display number. Please select a valid gadget.", "Invalid Input", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 } catch (NumberFormatException e) {  
 JOptionPane.*showMessageDialog*(null, "Invalid input. Please enter a valid number.", "Invalid Input", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
  
  
 private double getDoubleValue(JTextField textField) {  
 // Get double value from text field  
 try {  
 return Double.*parseDouble*(textField.getText().trim());  
 } catch (NumberFormatException e) {  
 // Display error message for invalid input  
 JOptionPane.*showMessageDialog*(null, "Invalid input. Please enter a valid number.");  
 return 0;  
 }  
 }  
  
 private int getIntegerValue(JTextField textField) {  
 // Get integer value from text field  
 try {  
 return Integer.*parseInt*(textField.getText().trim());  
 } catch (NumberFormatException e) {  
 // Display error message for invalid input  
 JOptionPane.*showMessageDialog*(null, "Invalid input. Please enter a valid integer.");  
 return 0;  
 }  
 }  
  
 // Method to get the display number via GUI input  
 private int getDisplayNumber() {  
 return Integer.*parseInt*(JOptionPane.*showInputDialog*(null, "Enter the display number:"));  
 }  
  
 // Method to get the phone number via GUI input  
 private String getPhoneNumber() {  
 return JOptionPane.*showInputDialog*(null, "Enter the phone number:");  
 }  
  
 // Method to get the call duration via GUI input  
 private int getDuration() {  
 return Integer.*parseInt*(JOptionPane.*showInputDialog*(null, "Enter the call duration (in minutes):"));  
 }  
  
 // Method to handle making a call  
 public void makeCall() {  
 try {  
 int displayNumber = getDisplayNumber();  
 if (displayNumber >= 0 && displayNumber < gadgets.size()) {  
 Gadget selectedGadget = gadgets.get(displayNumber);  
 if (selectedGadget instanceof Mobile) {  
 String phoneNumber = getPhoneNumber();  
 int duration = getDuration();  
 ((Mobile) selectedGadget).makeCall(phoneNumber, duration);  
 JOptionPane.*showMessageDialog*(null, "Call made successfully!", "Call Success", JOptionPane.*INFORMATION\_MESSAGE*);  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Selected gadget is not a mobile phone.", "Invalid Gadget", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Invalid display number. Please select a valid gadget.", "Invalid Input", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 } catch (NumberFormatException e) {  
 JOptionPane.*showMessageDialog*(null, "Invalid input. Please enter a valid number.", "Invalid Input", JOptionPane.*ERROR\_MESSAGE*);  
 } catch (IndexOutOfBoundsException e) {  
 JOptionPane.*showMessageDialog*(null, "Invalid display number. Please select a valid gadget.", "Invalid Input", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
}

// Main Class  
public class Main {  
 public static void main(String[] args) { // Main Method  
 GadgetShop gadgetShop = new GadgetShop();  
 }  
}