Table of Contents

[UML Class Diagram: 3](#_Toc165665742)

[Project Description: 4](#_Toc165665743)

[Functionality: 5](#_Toc165665744)

[Gadget Class: 5](#_Toc165665745)

[Mobile class inherits Gadget: 7](#_Toc165665746)

[To handle adding mobile: 16](#_Toc165665747)

[Method to handle calling a phone: 19](#_Toc165665748)

[Method to download music: 21](#_Toc165665749)

[GadgetShop.java: 26](#_Toc165665750)

[Graphical User Interface: 27](#_Toc165665751)

[Test Cases: 29](#_Toc165665752)

[1)Adding a phone: 29](#_Toc165665753)

[29](#_Toc165665754)

[2)Adding MP3 Player: 30](#_Toc165665755)

[3)Displaying details of all gadgets: 31](#_Toc165665756)

[31](#_Toc165665757)

[4)Making a call: 32](#_Toc165665758)

[32](#_Toc165665759)

[Test 7: 33](#_Toc165665760)

[Pseudocode: 34](#_Toc165665761)

[Getting Display Number: 34](#_Toc165665762)

[Adding a Mobile: 34](#_Toc165665763)

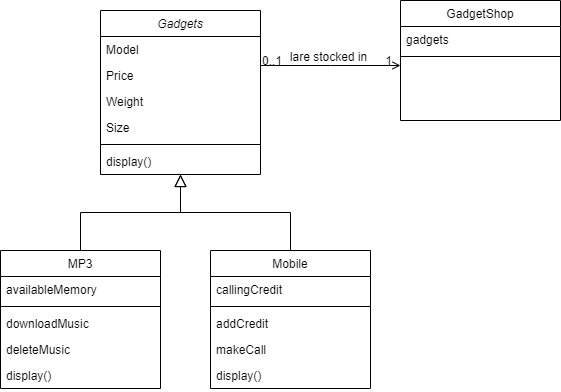
[Adding an MP3: 35](#_Toc165665764)

[Displaying All Gadgets: 36](#_Toc165665765)

[End Making a Call: 36](#_Toc165665766)

[Downloading Music: 37](#_Toc165665767)

## UML Class Diagram:

Project Report: Gadget Shop Application

## Project Description:

1. Introduction:

The Gadget Shop Application is a JavaFX-based software designed to manage gadgets like mobile phones and MP3 players. It allows users to add new gadgets, such as mobile phones or MP3 players, display all gadgets, clear input fields, and perform actions like making calls and downloading music.

2. Project Structure:

- Main.java: Entry point of the application. Loads the FXML file and initializes the primary stage.

-GadgetShop.java: contains a list of gadgets currently stored

- GadgetShop.fxml: FXML file defining the GUI layout using JavaFX elements.

- GadgetShopController.java: Controller class handling user interactions and application logic.

- Mobile.java: Class representing a mobile phone gadget.

- MP3.java: Class representing an MP3 player gadget.

- application.css: CSS file for styling the application (optional).

3. Application Features:

-Add Mobile: Allows users to add a new mobile phone with specified details such as model, price, weight, size, and credit.

- Add MP3: Enables users to add a new MP3 player with details like model, price, weight, size, and memory.

- Display All: Displays all gadgets currently stored in the application.

- Clear: Clears all input fields.

- Make Call: Allows users to make a call using a mobile phone gadget.

- Download Music: Enables users to download music using an MP3 player gadget.

4. Technical Implementation:

- The application is built using JavaFX for the GUI and FXML for defining the layout structure.

- The controller class, `GadgetShopController`, handles user interactions and performs necessary actions based on user inputs.

- Exception handling is implemented to deal with potential errors such as parsing input data.

- The application follows the Model-View-Controller (MVC) architectural pattern to separate concerns and maintain code readability and maintainability.

## Functionality:

### Gadget Class:

package application;

public class Gadget {

protected String model;

protected double price;

protected int weight;

protected String size;

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

this.model = model;

this.price = price;

this.weight = weight;

this.size = size;

}

public String getModel() {

return model;

}

public double getPrice() {

return price;

}

public int getWeight() {

return weight;

}

public String getSize() {

return size;

}

public void display() {

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

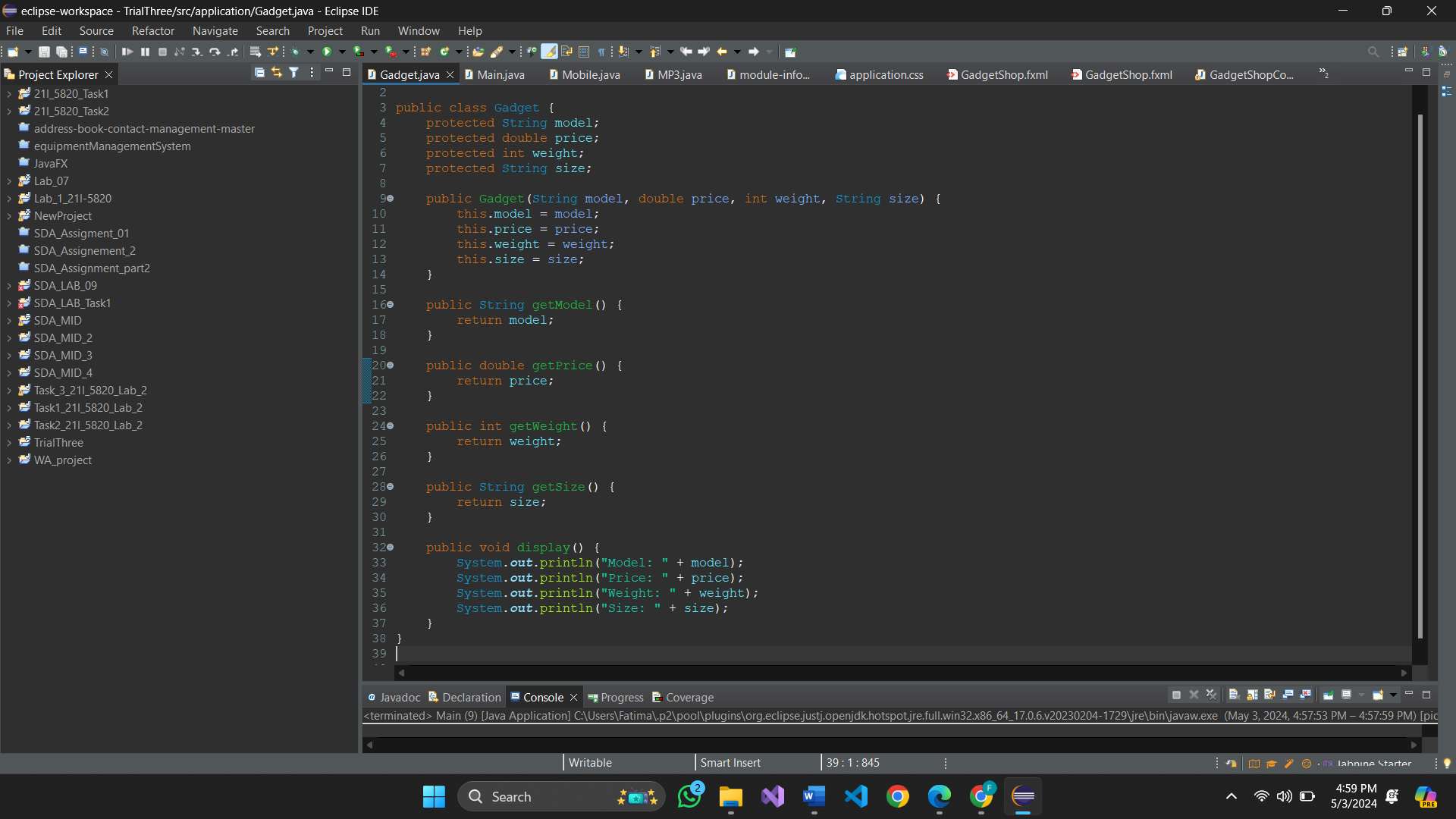
System.out.println("Price: " + price);

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

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

}

}



Mobile class inherits Gadget:  
  
package application;

public class Mobile extends Gadget {

private int callingCredit;

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

super(model, price, weight, size);

this.callingCredit = callingCredit;

}

public int getCallingCredit() {

return callingCredit;

}

public void addCredit(int amount) {

if (amount > 0) {

callingCredit += amount;

} else {

System.out.println("Please enter a positive amount for credit.");

}

}

public void makeCall(String phoneNumber, int duration) {

if (callingCredit >= duration) {

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

callingCredit -= duration;

} else {

System.out.println("Insufficient calling credit to make the call.");

}

}

@Override

public void display() {

super.display();

System.out.println("Calling Credit: " + callingCredit);

}

}

A computer screen with many colorful lines

Description automatically generated

MP3 class also inherits Gadget:  
package application;

public class MP3 extends Gadget {

private int availableMemory;

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

super(model, price, weight, size);

this.availableMemory = availableMemory;

}

public int getAvailableMemory() {

return availableMemory;

}

public void downloadMusic(int size) {

if (size <= availableMemory) {

availableMemory -= size;

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

} else {

System.out.println("Not enough memory to download music.");

}

}

public void deleteMusic(int size) {

availableMemory += size;

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

}

@Override

public void display() {

super.display();

System.out.println("Available Memory: " + availableMemory);

}

}

A screen shot of a computer program

Description automatically generated  
  
  
We used FXML to create GUI:

Our Program consists of only one page which is built on a GridPane. It consists of 10 Labels and 10 TextField along with 6 buttons.  
  
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>

<?import javafx.scene.control.Button?>

<?import javafx.scene.control.Label?>

<?import javafx.scene.control.TextField?>

<?import javafx.scene.layout.ColumnConstraints?>

<?import javafx.scene.layout.GridPane?>

<?import javafx.scene.layout.RowConstraints?>

<GridPane fx:id="gridPane" alignment="center" hgap="10" vgap="10" xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1" fx:controller="application.GadgetShopController">

<columnConstraints>

<ColumnConstraints hgrow="SOMETIMES" maxWidth="200.0" minWidth="10.0" prefWidth="100.0" />

<ColumnConstraints hgrow="SOMETIMES" maxWidth="200.0" minWidth="10.0" prefWidth="100.0" />

</columnConstraints>

<rowConstraints>

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

</rowConstraints>

<children>

<Label text="Model:" GridPane.columnIndex="0" GridPane.rowIndex="0" />

<TextField fx:id="modelTextField" GridPane.columnIndex="1" GridPane.rowIndex="0" />

<Label style="-fx-border-radius: 5;" text="Price (£):" GridPane.columnIndex="0" GridPane.rowIndex="1" />

<TextField fx:id="priceTextField" GridPane.columnIndex="1" GridPane.rowIndex="1" />

<Label style="-fx-border-radius: 5;" text="Weight (g):" GridPane.columnIndex="0" GridPane.rowIndex="2" />

<TextField fx:id="weightTextField" GridPane.columnIndex="1" GridPane.rowIndex="2" />

<Label text="Size (mm):" GridPane.columnIndex="0" GridPane.rowIndex="3" />

<TextField fx:id="sizeTextField" GridPane.columnIndex="1" GridPane.rowIndex="3" />

<Label style="-fx-border-radius: 5;" text="Credit (mins):" GridPane.columnIndex="0" GridPane.rowIndex="4" />

<TextField fx:id="creditTextField" GridPane.columnIndex="1" GridPane.rowIndex="4" />

<Label style="-fx-border-radius: 5;" text="Memory (MB):" GridPane.columnIndex="0" GridPane.rowIndex="5" />

<TextField fx:id="memoryTextField" GridPane.columnIndex="1" GridPane.rowIndex="5" />

<Button fx:id="addMobileButton" onAction="#handleAddMobile" text="Add Mobile" GridPane.rowIndex="10" />

<Button fx:id="addMP3Button" onAction="#handleAddMP3" text="Add MP3" GridPane.columnIndex="1" GridPane.rowIndex="10" />

<Button fx:id="clearButton" onAction="#handleClear" text="Clear" GridPane.halignment="LEFT" GridPane.rowIndex="11" />

<TextField fx:id="displayNumTextField" GridPane.columnIndex="1" GridPane.rowIndex="9" />

<TextField fx:id="phoneTextField" GridPane.columnIndex="1" GridPane.rowIndex="6" />

<TextField fx:id="durationTextField" GridPane.columnIndex="1" GridPane.rowIndex="7" />

<TextField fx:id="downloadTextField" GridPane.columnIndex="1" GridPane.rowIndex="8" />

<Button fx:id="displayAllButton" onAction="#handleDisplayAll" text="Display All" GridPane.columnIndex="1" GridPane.halignment="LEFT" GridPane.rowIndex="11" />

<Label text="PhoneNo:" GridPane.rowIndex="6" />

<Label text="Duration" GridPane.rowIndex="7" />

<Label text="Download:" GridPane.rowIndex="8" />

<Label text="DisplayNumber:" GridPane.rowIndex="9" />

<Button mnemonicParsing="false" text="Call" GridPane.rowIndex="12" />

<Button mnemonicParsing="false" text="Download Music" GridPane.columnIndex="1" GridPane.rowIndex="12" />

</children>

<opaqueInsets>

<Insets />

</opaqueInsets>

<padding>

<Insets bottom="30.0" left="30.0" right="30.0" top="30.0" />

</padding>

</GridPane>  
  
  
  
  
  
  
A computer screen shot of a program

Description automatically generated  
  
A computer screen shot of a program

Description automatically generated  
  
  
GadgetShopController.java is the controller for FXML file:

## To handle adding mobile:

@FXML

public void handleAddMobile() {

String model = modelTextField.getText();

String priceStr = priceTextField.getText();

String weightStr = weightTextField.getText();

String size = sizeTextField.getText();

String creditStr = creditTextField.getText();

if (model.isEmpty() || priceStr.isEmpty() || weightStr.isEmpty() || size.isEmpty() || creditStr.isEmpty()) {

showAlert(Alert.AlertType.ERROR, "Error", "Please fill in all fields.");

return;

}

try {

double price = Double.parseDouble(priceStr);

double weight = Double.parseDouble(weightStr);

int credit = Integer.parseInt(creditStr);

GadgetShop.Gadgets.add(new Mobile(model, price, (int) weight, size, credit));

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

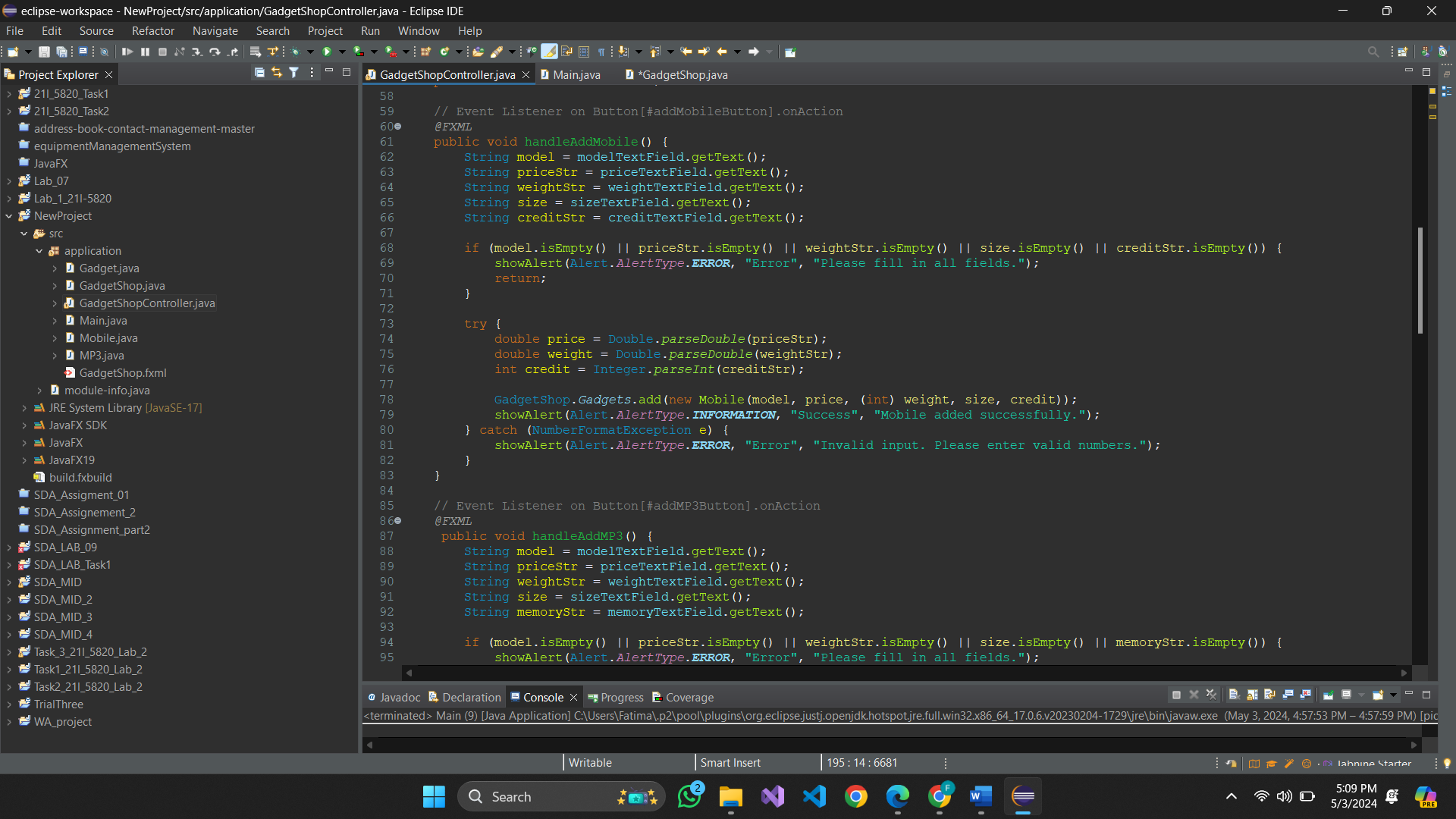
} catch (NumberFormatException e) {

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

}

}

**Notice that we have added error handling methods to ensure we don’t get empty or unexpected values.**

Method to handle adding MP3:  
public void handleAddMP3() {

String model = modelTextField.getText();

String priceStr = priceTextField.getText();

String weightStr = weightTextField.getText();

String size = sizeTextField.getText();

String memoryStr = memoryTextField.getText();

if (model.isEmpty() || priceStr.isEmpty() || weightStr.isEmpty() || size.isEmpty() || memoryStr.isEmpty()) {

showAlert(Alert.AlertType.ERROR, "Error", "Please fill in all fields.");

return;

}

try {

double price = Double.parseDouble(priceStr);

double weight = Double.parseDouble(weightStr);

int memory = Integer.parseInt(memoryStr);

GadgetShop.Gadgets.add(new MP3(model, price, (int) weight, size, memory));

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

} catch (NumberFormatException e) {

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

}

}  
  
  
Method to handle clear button:  
public void handleClear() {

modelTextField.clear();

priceTextField.clear();

weightTextField.clear();

sizeTextField.clear();

creditTextField.clear();

memoryTextField.clear();

displayNumTextField.clear();

phoneTextField.clear();

durationTextField.clear();

downloadTextField.clear();

}

A computer screen shot of a program

Description automatically generated

Method to handle calling a phone:  
public void MakeCall() {

// Check if display number and duration fields are empty

String displayNum = displayNumTextField.getText().trim();

String duration = durationTextField.getText().trim();

if (displayNum.isEmpty() || duration.isEmpty()) {

// Show an alert if any of the fields are empty

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Error");

alert.setHeaderText(null);

alert.setContentText("Please enter both display number and duration.");

alert.showAndWait();

return;

}

int displayNum2 = Integer.parseInt(displayNum);

int duration2 = Integer.parseInt(duration);

Mobile mob =(Mobile)GadgetShop.Gadgets.get(displayNum2);

// Check call credit left

double callCreditLeft = mob.getCallingCredit(); // You need to implement this method

// Parse duration to double

if (callCreditLeft >= duration2) {

// If call credit is greater than or equal to duration, show success message

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Success");

alert.setHeaderText(null);

alert.setContentText("Call successfully initiated!");

alert.showAndWait();

} else {

// Otherwise, show an alert indicating insufficient call credit

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Error");

alert.setHeaderText(null);

alert.setContentText("Insufficient call credit!");

alert.showAndWait();

}

}  
A computer screen shot of a program

Description automatically generated

Method to download music:  
public void DownloadMusic() {

if(downloadTextField.getText().isEmpty()||displayNumTextField.getText().isEmpty())

{

showAlert(Alert.AlertType.ERROR, "Error", "Enter values in download size and display field");

}

else

{

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

int display = Integer.parseInt(displayNumTextField.getText());

MP3 MP3player = (MP3)GadgetShop.Gadgets.get(display);

if(MP3player.getAvailableMemory()>=downloadSize)

{

showAlert(Alert.AlertType.INFORMATION,"Success", "Song Downloaded!");

}

else

{

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

}

}

}

Helper function to show alerts where needed:

private void showAlert(Alert.AlertType type, String title, String message) {

Alert alert = new Alert(type);

alert.setTitle(title);

alert.setContentText(message);

alert.showAndWait();

}

A computer screen shot of a program

Description automatically generated  
  
Main.java:

Simply loads the FXML file and shows the scene.

package application;

import javafx.application.Application;

import javafx.stage.Stage;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Alert.AlertType;

import javafx.scene.layout.GridPane;

import javafx.fxml.FXMLLoader;

public class Main extends Application {

@Override

public void start(Stage primaryStage) {

try {

// Load the FXML file

GridPane root = (GridPane)FXMLLoader.load(getClass().getResource("GadgetShop.fxml"));

// Create a new scene with the loaded root and set the size

Scene scene = new Scene(root, 400, 400);

// Add external CSS file to the scene

//scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

// Set the scene to the primary stage and show it

primaryStage.setScene(scene);

primaryStage.show();

} catch(Exception e) {

// Handle the exception gracefully

System.err.println("Error loading FXML file: " + e.getMessage());

// Display an alert dialog to inform the user about the error

Alert alert = new Alert(AlertType.ERROR);

alert.setTitle("Error");

alert.setHeaderText("Unable to load application");

alert.setContentText("An error occurred while loading the application. Please try again later.");

alert.showAndWait();

// Optionally, you can close the application or perform other actions based on the error

e.printStackTrace();

// Exit the application

System.exit(1);

}

}

public static void main(String[] args) {

launch(args);

}

}

A computer screen shot of a program

Description automatically generated

GadgetShop.java:  
package application;

import java.util.ArrayList;

public class GadgetShop {

public static ArrayList<Gadget> Gadgets;

GadgetShop(){

Gadgets = new ArrayList<Gadget>();

}

}  
  
Simply contains a list of Gadgets currently stocked.

# Graphical User Interface:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated  
  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated

# Test Cases:

### 1)Adding a phone:

### 

### 2)Adding MP3 Player:

A screenshot of a computer

Description automatically generated

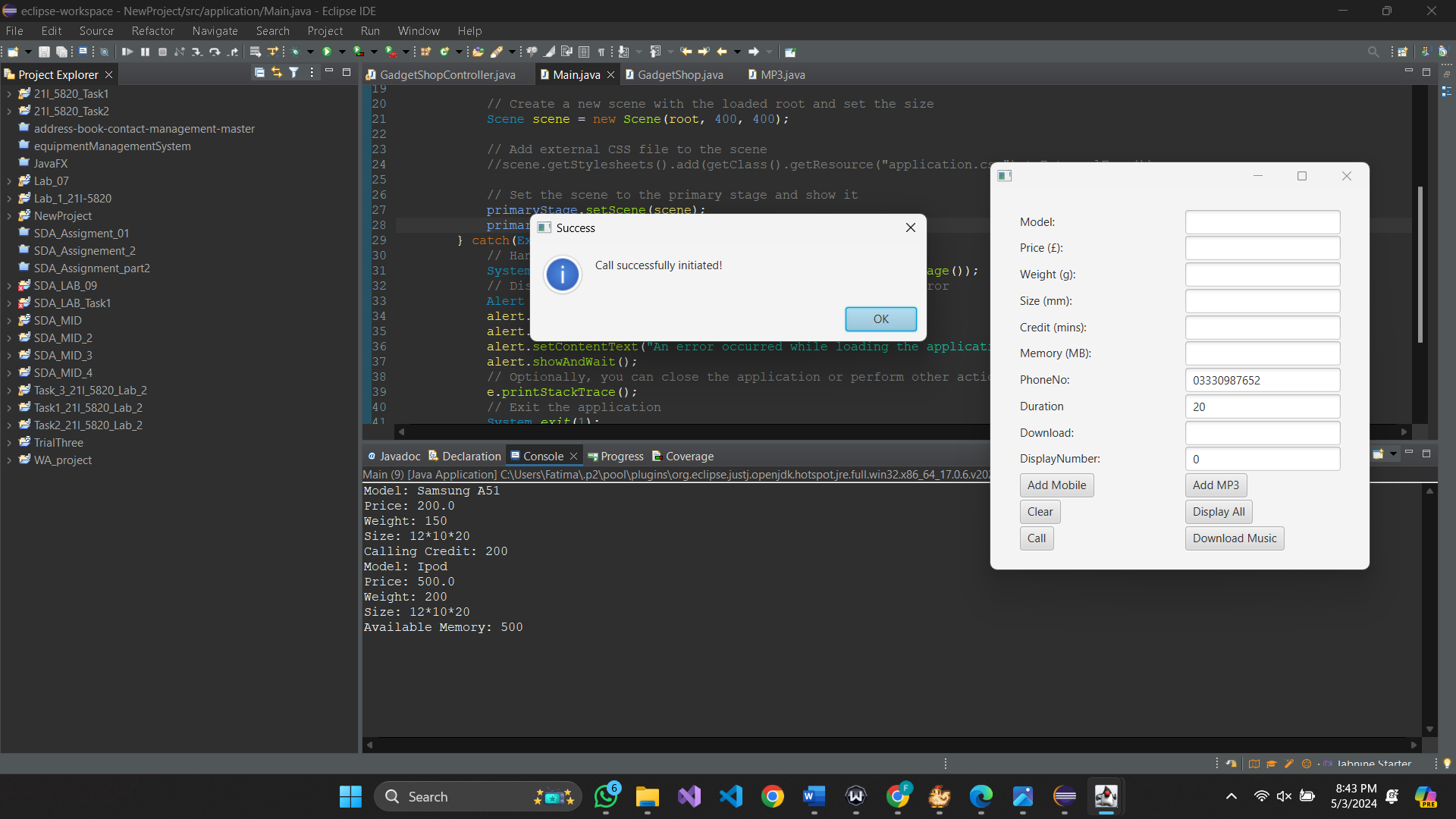
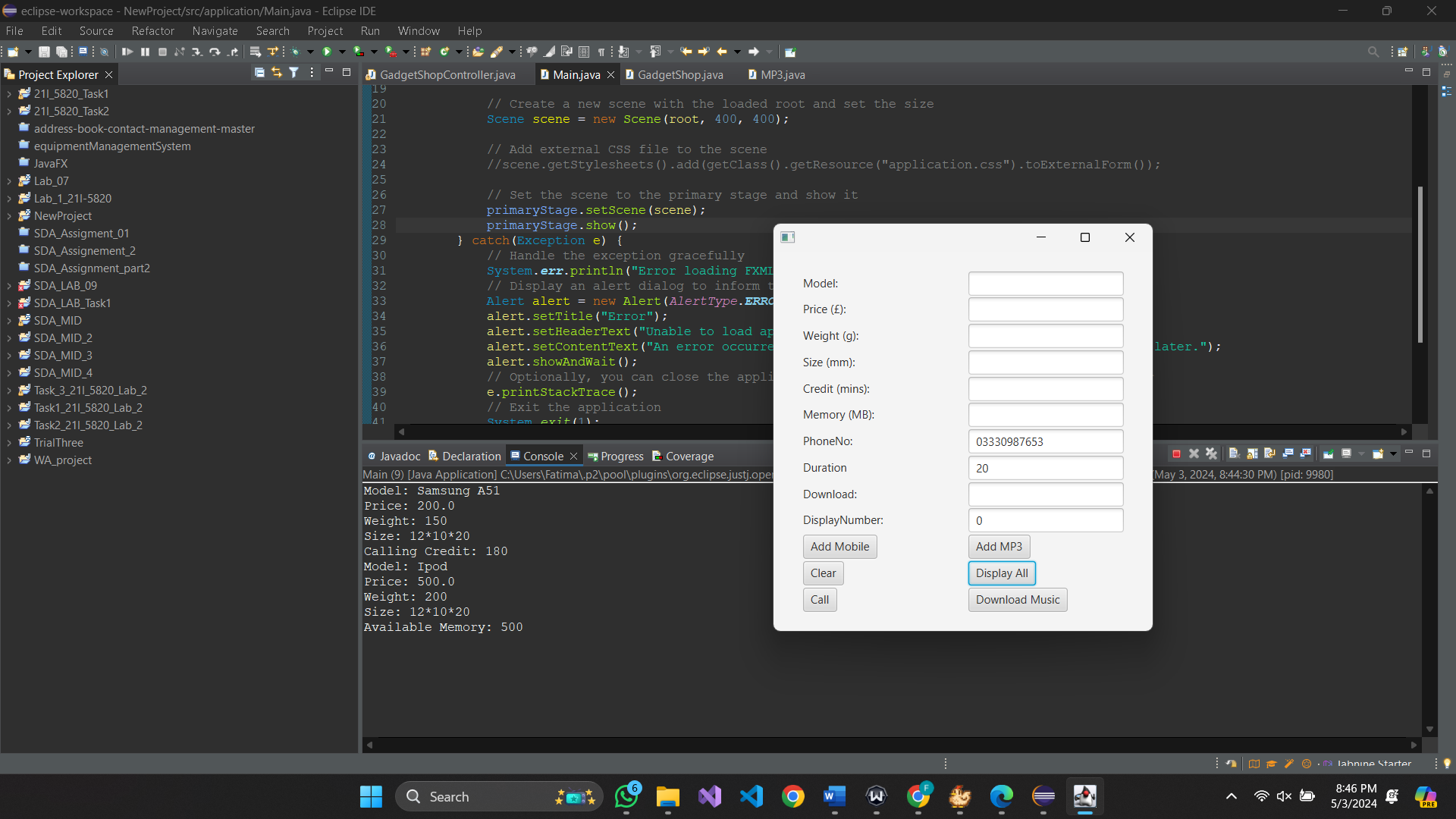
### 3)Displaying details of all gadgets:

### A computer screen with a white box Description automatically generated

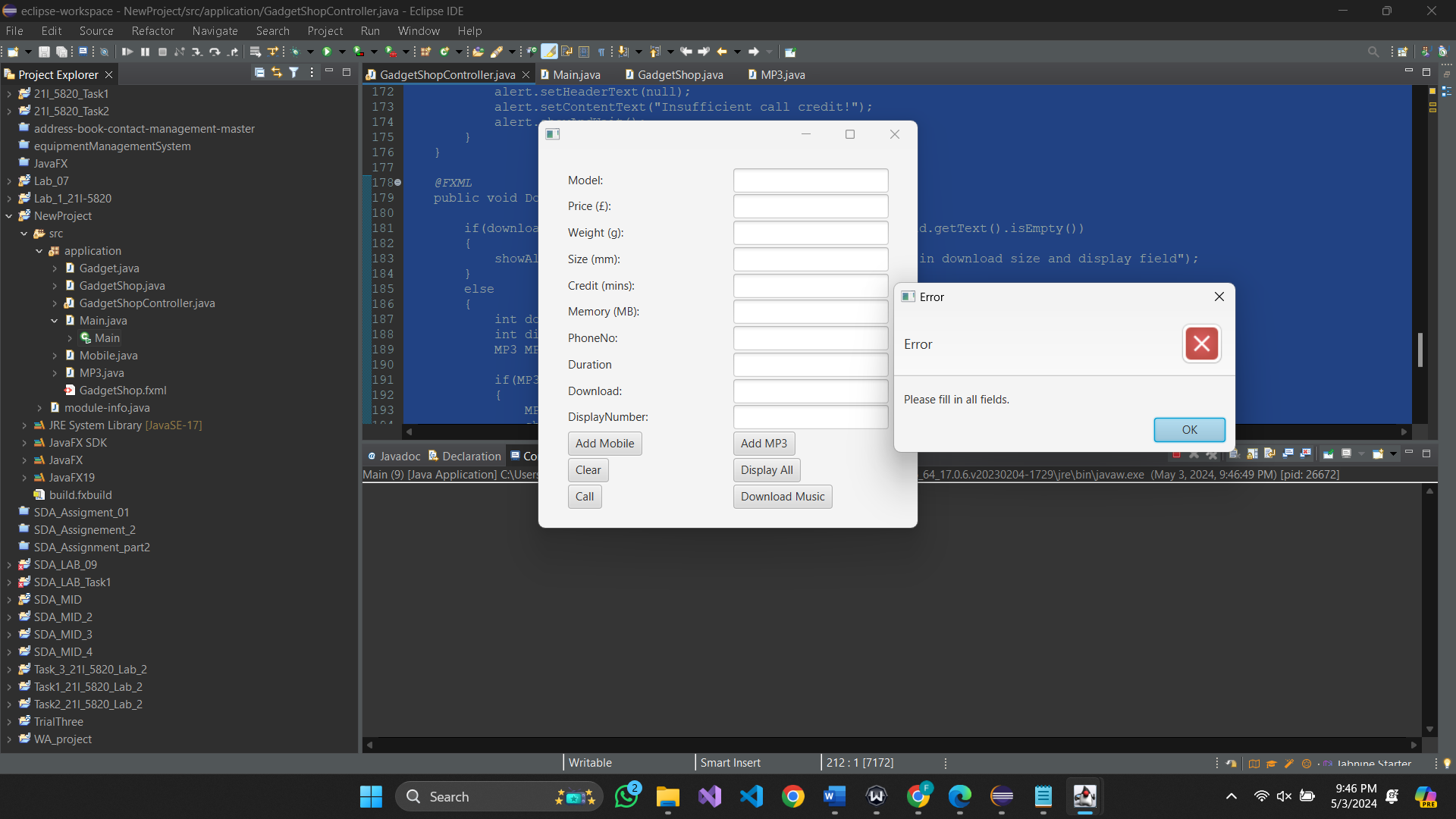
### 4)Making a call:

Details before making a call:

### 

call initiated:  
  
  
details after call:  


### Test 7:



# Pseudocode:

## Getting Display Number:

Method: GetDisplayNumber()

Input: None

Output: DisplayNumber

Begin

// Get the display number from the displayNumTextField

DisplayNumber = ParseInteger(displayNumTextField.getText())

Return DisplayNumber

End

## Adding a Mobile:

Method: AddMobile()

Input: None

Output: None

Begin

// Get values from the text fields

Model = modelTextField.getText()

Price = ParseDouble(priceTextField.getText())

Weight = ParseDouble(weightTextField.getText())

Size = sizeTextField.getText()

Credit = ParseInteger(creditTextField.getText())

// Create a new Mobile object with the obtained values and add it to the GadgetShop's gadgets list

GadgetShop.Gadgets.add(new Mobile(Model, Price, Weight, Size, Credit))

ShowAlert("Success", "Mobile added successfully.")

End

## Adding an MP3:

Method: AddMP3()

Input: None

Output: None

Begin

// Get values from the text fields

Model = modelTextField.getText()

Price = ParseDouble(priceTextField.getText())

Weight = ParseDouble(weightTextField.getText())

Size = sizeTextField.getText()

Memory = ParseInteger(memoryTextField.getText())

// Create a new MP3 object with the obtained values and add it to the GadgetShop's gadgets list

GadgetShop.Gadgets.add(new MP3(Model, Price, Weight, Size, Memory))

ShowAlert("Success", "MP3 player added successfully.")

End

## Displaying All Gadgets:

Method: DisplayAllGadgets()

Input: None

Output: None

Begin

// Iterate through each gadget in the GadgetShop's gadgets list

For each gadget in GadgetShop.Gadgets

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

## End Making a Call:

Method: MakeCall()

Input: None

Output: None

Begin

// Get values from the text fields

DisplayNumber = ParseInteger(displayNumTextField.getText().trim())

Duration = ParseInteger(durationTextField.getText().trim())

// Retrieve the Mobile object corresponding to the given display number from the gadgets list

Mobile = GadgetShop.Gadgets[DisplayNumber]

// Check if there is sufficient call credit for the given duration

If Mobile.getCallingCredit() >= Duration

// Deduct the call duration from the call credit

Mobile.setCallingCredit(Mobile.getCallingCredit() - Duration)

ShowAlert("Success", "Call successfully initiated!")

Else

ShowAlert("Error", "Insufficient call credit!")

End

## Downloading Music:

Method: DownloadMusic()

Input: None

Output: None

Begin

// Get values from the text fields

DownloadSize = ParseInteger(downloadTextField.getText().trim())

DisplayNumber = ParseInteger(displayNumTextField.getText().trim())

// Retrieve the MP3 object corresponding to the given display number from the gadgets list

MP3Player = GadgetShop.Gadgets[DisplayNumber]

// Check if there is sufficient memory for downloading music

If MP3Player.getAvailableMemory() >= DownloadSize

// Deduct the download size from the available memory

MP3Player.setMemory(MP3Player.getAvailableMemory() - DownloadSize)

ShowAlert("Success", "Song Downloaded!")

Else

ShowAlert("Error", "Insufficient memory!")

End