### lmu_blakCourse Submission Cover Sheet Module: CC4001 Programming Engineering

### Component no: 003

### Weighting: 60% of module mark

### Deadline: 3rd of May 2024

**Module Leader: Sandra Fernando StudentID:23026303**

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:) Bordei Date: 29April 2024

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.

1

Catalog

[GitHub link 3](#_Toc26883)

[A class diagram: 3](#_Toc6203)

[A description of every method 4](#_Toc2925)

[Gadget class: 4](#_Toc24053)

[Mobile class: 4](#_Toc5625)

[MP3 class 5](#_Toc842)

[Pseudocode for some button-handling methods 6](#_Toc20139)

[Textboxes, input check with try/catch 8](#_Toc30776)

[Validation using try/catch 8](#_Toc18293)

[1st example 8](#_Toc26948)

[2nd example 9](#_Toc15281)

[3rd example: 9](#_Toc10632)

[Buttons and Action Performed Methods 10](#_Toc19098)

[Test that the programme can be run in command prompt 10](#_Toc6610)

[GUI: display, add mobile, add MP3, display, make a call, download music 10](#_Toc32342)

[Testing 11](#_Toc24051)

[Test 1: Adding a mobile to the array list 11](#_Toc19602)

[Test 2: Adding an MP3 player to the array list 12](#_Toc31104)

[Test 3: Displaying the details of all of the gadgets in the array list 14](#_Toc25795)

[Test 4: Making a call 14](#_Toc30601)

[Test 6: Test that the program can be compiled and run using the command prompt, including a screenshot similar to Figure 1 in the command prompt learning aid. 16](#_Toc377)

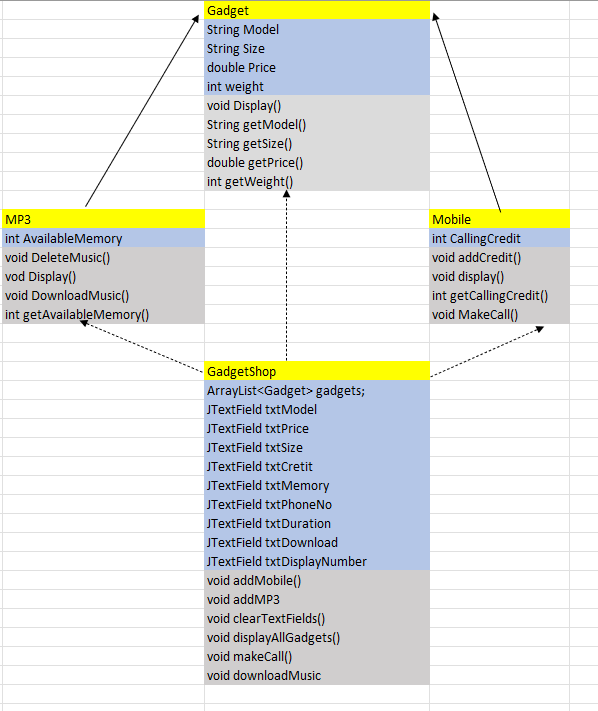
[Test 7: Test that appropriate dialog boxes appear when unsuitable values are entered for the display number. 16](#_Toc7443)

[Error detection 20](#_Toc23929)

[Conclusion 22](#_Toc25688)

# GitHub link

# A class diagram:



# A description of every method

## Gadget class:

The Gadget class serves as a base class for various types of gadgets, encapsulating common properties that all gadgets would have such as model, price, weight, and size. Here’s a detailed description of each method in the Gadget class:

1)Constructor (public Gadget(String model, double price, int weight, String size)):

* Purpose: Constructs a new Gadget object with specified attributes.
* Parameters:
* model: A string representing the model of the gadget.
* price: A double representing the price of the gadget.
* weight: An integer representing the weight of the gadget in grams.
* size: A string representing the physical dimensions or screen size of the gadget.
* Functionality: Initializes the gadget with the given attributes. Each attribute is stored in a private field specific to the gadget.

1. getModel (public String getModel()):

* Purpose: Retrieves the model of the gadget.
* Returns: The model of the gadget as a string.

1. getPrice (public double getPrice()):

* Purpose: Retrieves the price of the gadget.
* Returns: The price of the gadget as a double.

1. getWeight (public int getWeight()):

* Purpose: Retrieves the weight of the gadget.
* Returns: The weight of the gadget in grams as an integer.

1. getSize (public String getSize()):

* Purpose: Retrieves the size of the gadget.
* Returns: The size of the gadget as a string, which could represent either physical dimensions or screen size.

1. display (public void display()):

* Purpose: Prints the details of the gadget to the standard output.
* Functionality: Outputs the model, price formatted to two decimal places, weight in grams, and size of the gadget. This method is intended for debugging or user interface purposes to show all relevant information about the gadget.

This class serves as a foundational template from which specific types of gadgets like Mobile or MP3 players can inherit. These derived classes can extend the basic functionality of Gadget by adding additional properties or behaviors specific to their type while still maintaining the core attributes defined in Gadget.

## Mobile class:

The Mobile class is an extension of the Gadget class, designed to represent a mobile phone with additional properties and behaviors specific to mobile devices, such as managing calling credit. Here’s a brief description of each method in the Mobile class:

1. Constructor (public Mobile(String model, double price, int weight, String size, int callingCredit)):

* Purpose: Initializes a new instance of the Mobile class with specified values for model, price, weight, size, and calling credit.
* Parameters:
* model: A string representing the model of the mobile.
* price: A double representing the price of the mobile.
* weight: An integer representing the weight of the mobile in grams.
* size: A string representing the physical dimensions or screen size of the mobile.
* callingCredit: An integer representing the initial amount of calling credit in minutes.
* Functionality: Sets the attributes of the mobile device and initializes the calling credit.

1. getCallingCredit (public int getCallingCredit()):

* Purpose: Retrieves the current amount of calling credit available on the mobile.
* Returns: The current calling credit in minutes.

1. addCredit (public void addCredit(int additionalCredit)):

* Purpose: Adds more calling credit to the mobile.
* Parameters:
* additionalCredit: An integer representing the amount of additional calling credit to be added.
* Functionality: Checks if the additional credit is a positive number. If positive, it adds this credit to the existing credit and prints a confirmation message. If not, it prints an error message.

1. makeCall (public void makeCall(String phoneNumber, int duration)):

* Purpose: Attempts to make a phone call, deducting the necessary calling credit.
* Parameters:
* phoneNumber: A string representing the phone number to call.
* duration: An integer representing the duration of the call in minutes.
* Functionality: Checks if enough credit is available for the call. If sufficient credit is available, it deducts the duration from the calling credit and prints a message indicating the call is being made. If not enough credit is available, it prints a message indicating insufficient credit.

1. display (public void display()):

* Purpose: Displays the mobile's details along with its calling credit.
* Functionality: Calls the display method from the superclass (Gadget) to show the mobile's basic details and then prints the remaining calling credit.

These methods allow the Mobile class to manage specific attributes and behaviors related to mobile phones, such as handling calling credit and making calls, while still maintaining general gadget properties through its superclass, Gadget.

## MP3 class

The MP3 class extends the Gadget class and specializes it by adding functionality related to a digital music player, specifically managing its memory usage for storing music files. Here's a detailed description of each method in the MP3 class:

1. Constructor (public MP3(String model, double price, int weight, String size, int availableMemory)):

* Purpose: Constructs a new MP3 object with specified attributes including those for a general gadget, plus additional attributes specific to an MP3 player.
* Parameters:
* model: A string representing the model of the MP3 player.
* price: A double representing the price of the MP3 player.
* weight: An integer representing the weight of the MP3 player in grams.
* size: A string representing the physical dimensions or screen size of the MP3 player.
* availableMemory: An integer representing the available memory in megabytes (MB) for storing music.
* Functionality: Initializes the MP3 player with both the general gadget attributes and the specific attribute related to its functionality (memory).

1. getAvailableMemory (public int getAvailableMemory()):

* Purpose: Retrieves the available memory of the MP3 player.
* Returns: The amount of available memory in MB.

1. downloadMusic (public void downloadMusic(int memorySize)):

* Purpose: Attempts to download music by deducting the specified amount of memory from the available memory.
* Parameters:
* memorySize: An integer representing the size of the music file in MB to be downloaded.
* Functionality: Checks if there is enough available memory to store the music file. If sufficient memory exists, it deducts the specified memory from the available memory and prints a success message with the remaining memory. If there isn’t enough memory, it prints an error message.

4) deleteMusic (public void deleteMusic(int memorySize)):

* Purpose: Frees up memory by simulating the deletion of music files.
* Parameters:
* memorySize: An integer representing the size of the music file in MB to be deleted.
* Functionality: Adds the specified amount of memory back to the available memory and prints a message indicating the new available memory.

5)display (public void display()):

* Purpose: Extends the display method of the superclass to include details specific to the MP3 player.
* Functionality: Calls the display method of the superclass to print out general gadget details, then prints additional details specific to the MP3 player, such as the available memory.

This class provides specialized behavior for an MP3 player, focusing on its ability to store and manage digital music files, enhancing the general functionality provided by the Gadget superclass.

## Pseudocode for some button-handling methods

* Getting the display number from the GUI
* Adding a mobile
* Adding an MP3
* Displaying all gadgets in the array list
* Making a call
* Downloading music

FUNCTION getDisplayNumber

Read text from txtDisplayNumber

IF text is numeric THEN

Convert text to integer and RETURN as displayNumber

ELSE

Display error message "Invalid display number"

RETURN -1

END FUNCTION

FUNCTION addMobile

Read and trim text from txtModel, txtPrice, txtWeight, txtSize, txtCredit

TRY

Convert txtPrice, txtWeight, txtCredit to their respective types

IF any field is empty or conversion fails THEN

Display error "Input Error"

ELSE

Create a new Mobile object with read values

Add the new Mobile to gadgets list

Display success message "Mobile added successfully"

Call clearTextFields to reset all text fields

CATCH any conversion errors

Display error message "Please enter valid numbers"

END FUNCTION

FUNCTION addMP3

Read and trim text from txtModel, txtPrice, txtWeight, txtSize, txtMemory

TRY

Convert txtPrice, txtWeight, txtMemory to their respective types

IF any field is empty or conversion fails THEN

Display error "Input Error"

ELSE

Create a new MP3 object with read values

Add the new MP3 to gadgets list

Display success message "MP3 added successfully"

Call clearTextFields to reset all text fields

CATCH any conversion errors

Display error message "Please enter valid numbers"

END FUNCTION

FUNCTION displayAllGadgets

Initialize an empty StringBuilder

FOR each gadget in gadgets list

Call display method on gadget and append output to StringBuilder

END FOR

Display the StringBuilder content in a message dialog

END FUNCTION

FUNCTION makeACall

Retrieve displayNumber using getDisplayNumber

Read phoneNumber and duration from respective text fields

IF displayNumber is valid and within range THEN

Retrieve gadget from gadgets list using displayNumber

IF gadget is an instance of Mobile THEN

Call makeCall method on Mobile instance

ELSE

Display error "Selected gadget is not a mobile"

END IF

ELSE

Display error "Invalid display number"

END IF

END FUNCTION

FUNCTION downloadMusic

Retrieve displayNumber using getDisplayNumber

Retrieve downloadSize using getDownloadSize

IF displayNumber is valid and within range THEN

Retrieve gadget from gadgets list using displayNumber

IF gadget is an instance of MP3 THEN

Call downloadMusic method on MP3 instance

ELSE

Display error "Selected gadget is not an MP3 player"

END IF

ELSE

Display error "Invalid display number"

END IF

END FUNCTION

# Textboxes, input check with try/catch

##### Textboxes:

txtModel = new JTextField(10);

txtPrice = new JTextField(5);

txtWeight = new JTextField(5);

txtSize = new JTextField(10);

txtCredit = new JTextField(5);

txtMemory = new JTextField(5);

txtPhoneNo = new JTextField(10);

txtDuration = new JTextField(5);

txtDownload = new JTextField(5);

txtDisplayNumber = new JTextField(5);

## Validation using try/catch

### 1st example

try {

String model = txtModel.getText().trim();

double price = Double.parseDouble(txtPrice.getText().trim());

int weight = Integer.parseInt(txtWeight.getText().trim());

String size = txtSize.getText().trim();

int credit = Integer.parseInt(txtCredit.getText().trim());

if (model.isEmpty() || price <= 0 || weight <= 0 || size.isEmpty() || credit < 0) {

JOptionPane.showMessageDialog(this, "Please fill out all fields correctly", "Input Error", JOptionPane.ERROR\_MESSAGE);

} else {

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

gadgets.add(newMobile);

JOptionPane.showMessageDialog(this, "Mobile added successfully", "Success", JOptionPane.INFORMATION\_MESSAGE);

clearTextFields();

}

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Please enter valid numbers for price, weight, and credit", "Input Error", JOptionPane.ERROR\_MESSAGE);

}

### 2nd example

try {

String model = txtModel.getText().trim();

double price = parseDouble(txtPrice.getText().trim());

int weight = parseInt(txtWeight.getText().trim());

String size = txtSize.getText().trim();

int memory = parseInt(txtMemory.getText().trim());

if (model.isEmpty() || price <= 0 || weight <= 0 || size.isEmpty() || memory <= 0) {

JOptionPane.showMessageDialog(this, "Please fill out all fields correctly", "Input Error", JOptionPane.ERROR\_MESSAGE);

return;

}

MP3 newMP3 = new MP3(model, price, weight, size, memory);

gadgets.add(newMP3);

JOptionPane.showMessageDialog(this, "MP3 added successfully", "Success", JOptionPane.INFORMATION\_MESSAGE);

clearTextFields();

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Please enter valid numbers for price, weight, and memory", "Input Error", JOptionPane.ERROR\_MESSAGE);

}

### 3rd example:

try {

int displayNumber = getDisplayNumber();

int downloadSize = getDownloadSize();

if (displayNumber != -1 && displayNumber < gadgets.size()) {

Gadget gadget = gadgets.get(displayNumber);

if (gadget instanceof MP3) {

((MP3) gadget).downloadMusic(downloadSize);

} else {

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

}

} else {

JOptionPane.showMessageDialog(this, "Invalid display number", "Error", JOptionPane.ERROR\_MESSAGE);

}

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Please enter valid numeric values for display number and download size", "Input Error", JOptionPane.ERROR\_MESSAGE);

}

# Buttons and Action Performed Methods

btnAddMobile.addActionListener(e -> addMobile());

btnAddMP3.addActionListener(e -> addMP3());

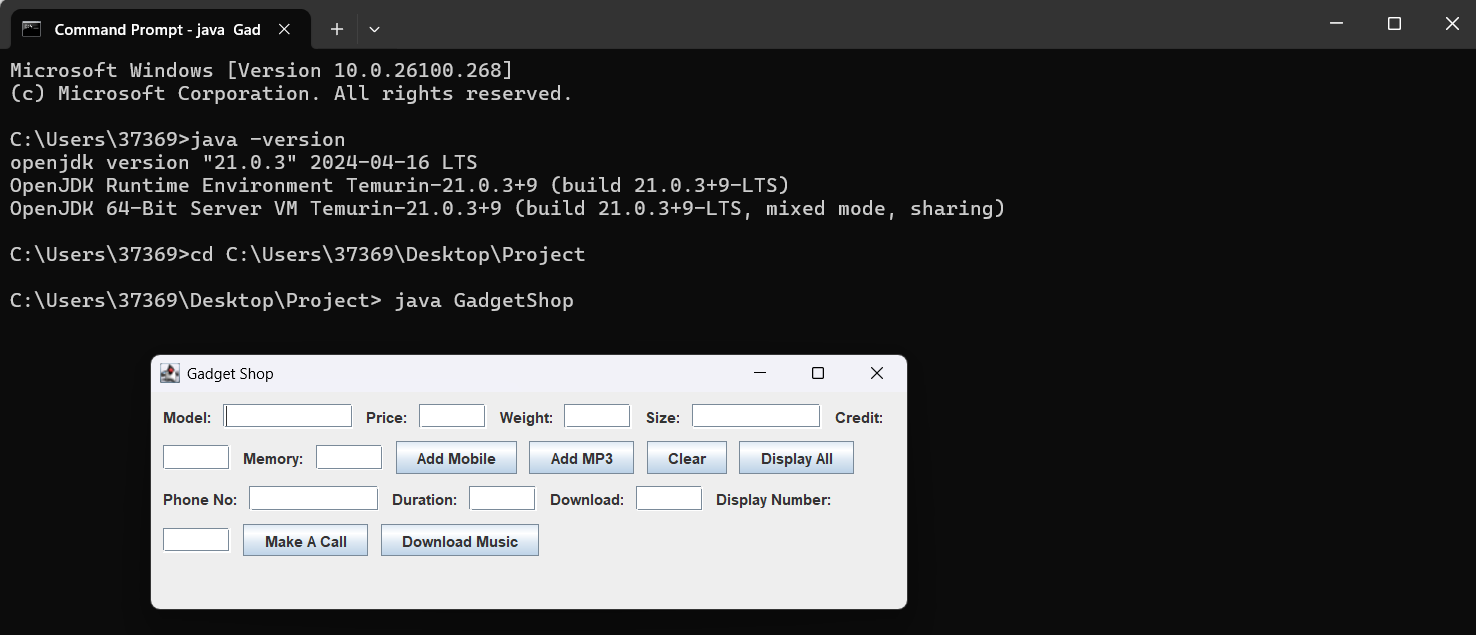
btnClear.addActionListener(e -> clearTextFields());

btnDisplayAll.addActionListener(e -> displayAllGadgets());

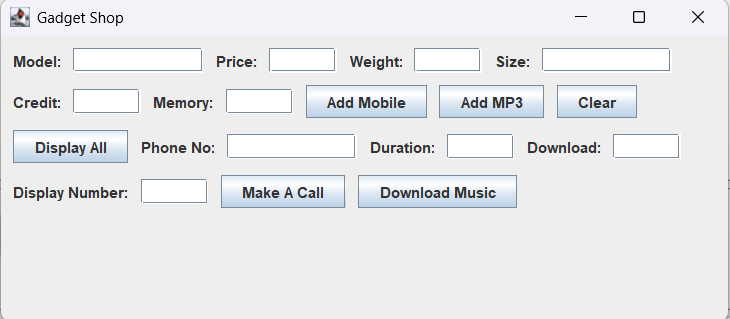
btnMakeCall.addActionListener(e -> makeACall());

btnDownloadMusic.addActionListener(e -> downloadMusic());

# Test that the programme can be run in command prompt

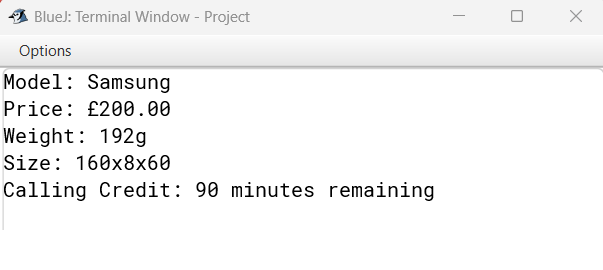
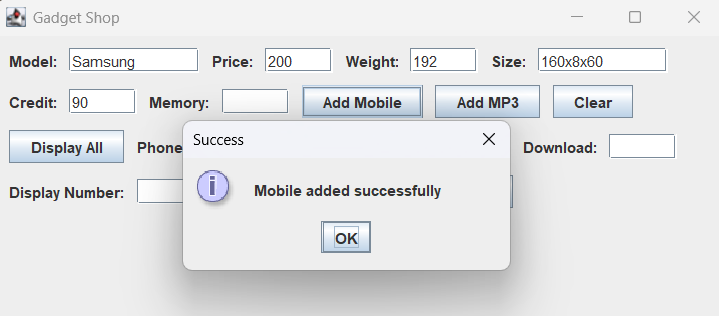
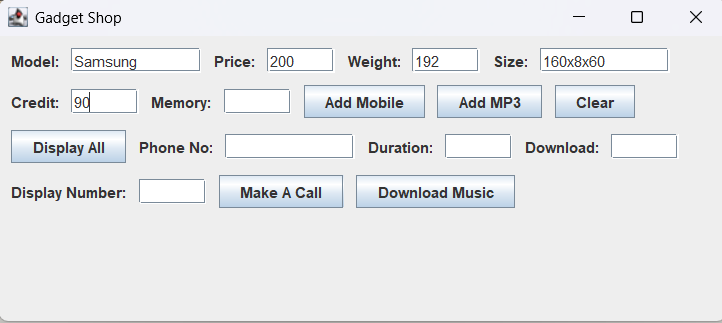


# GUI: display, add mobile, add MP3, display, make a call, download music

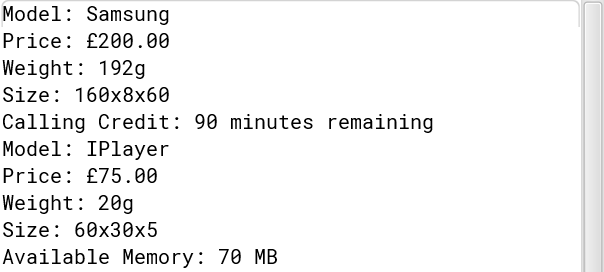
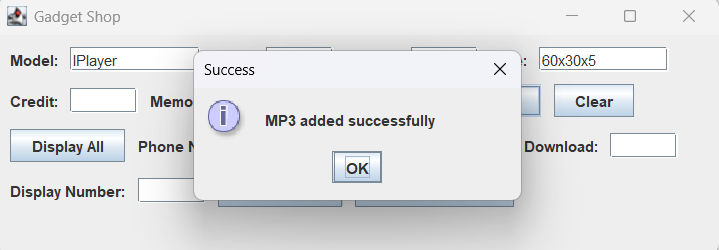
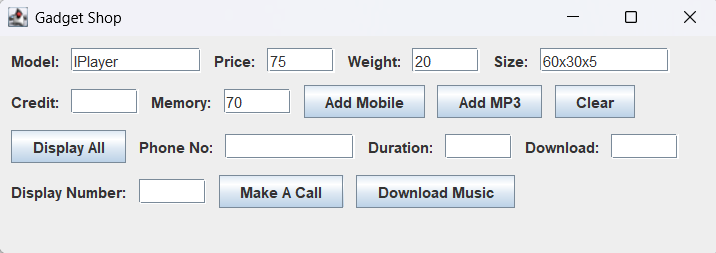


# Testing

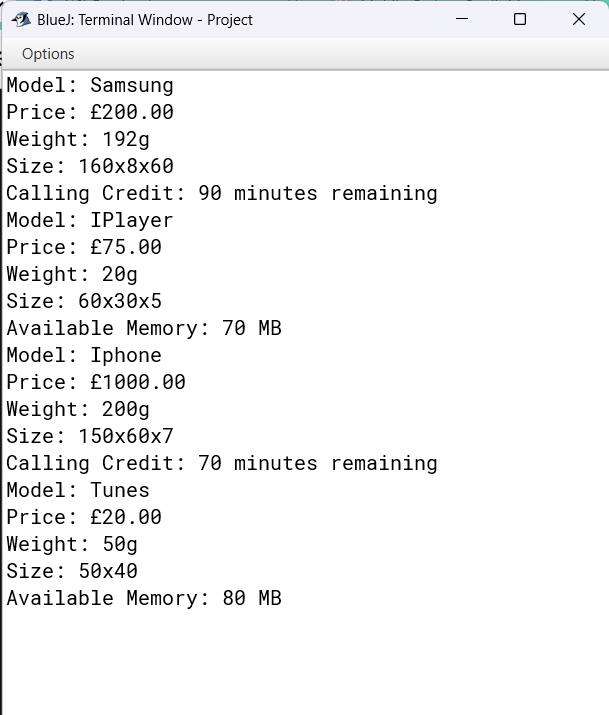
## Test 1: Adding a mobile to the array list



## Test 2: Adding an MP3 player to the array list

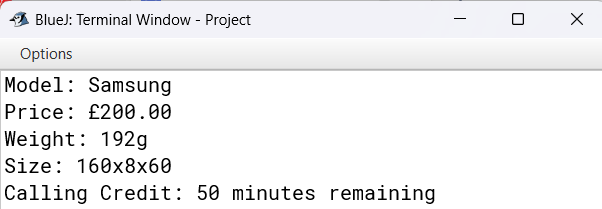


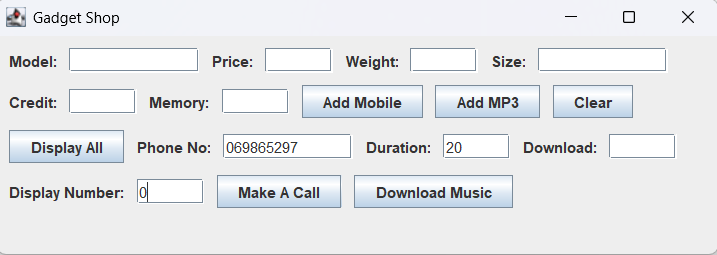
## Test 3: Displaying the details of all of the gadgets in the array list



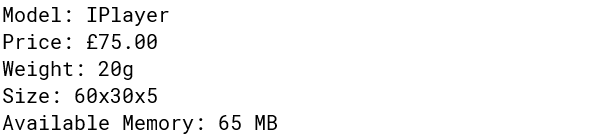
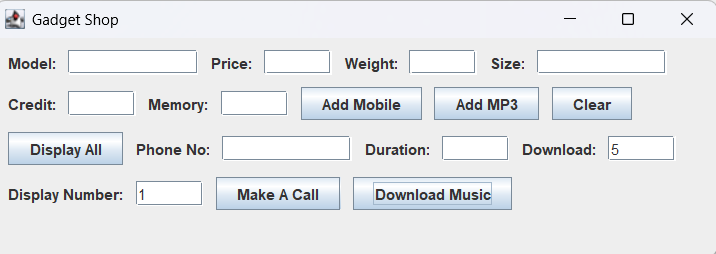
Test 4: Making a call

You can notice that I made the call twice and it lowered the amount of credit by 40 ,there are 50 credits remaining from 90 that were initially.

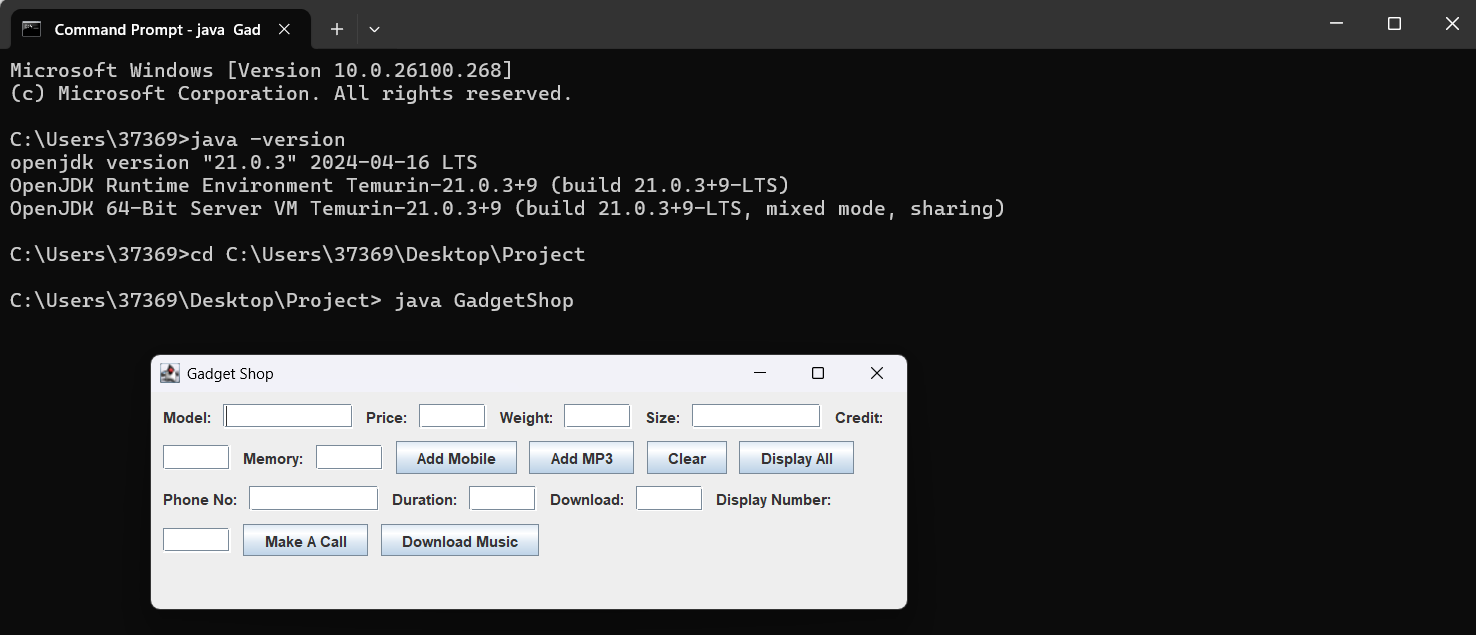


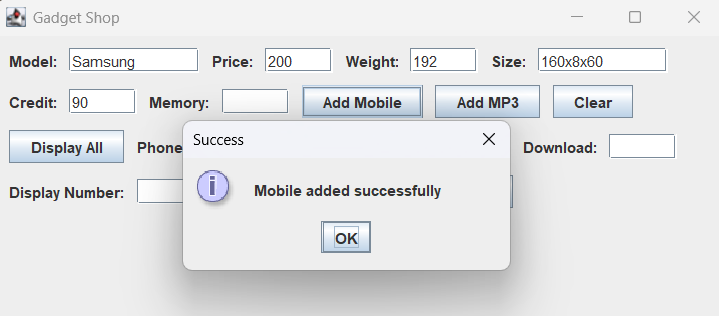
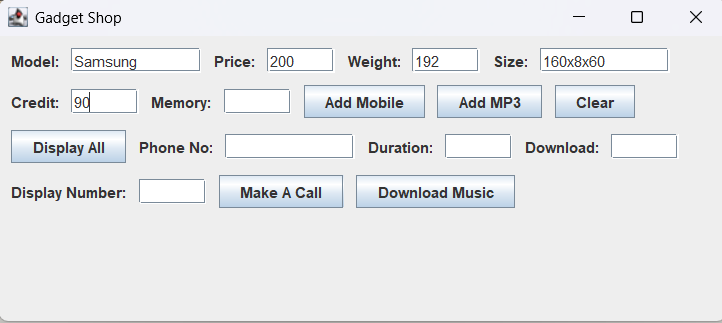
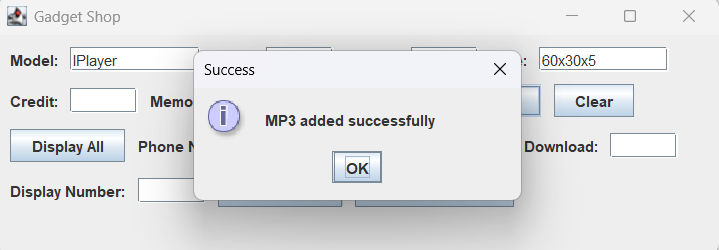
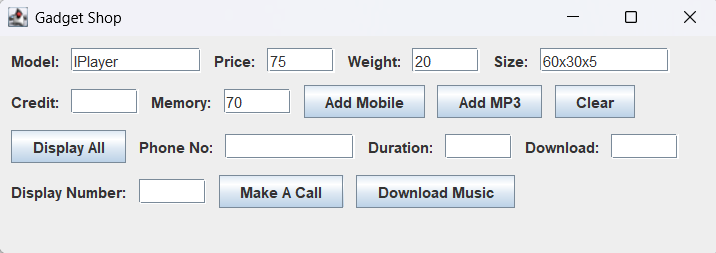


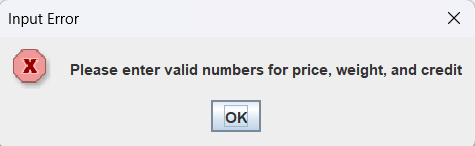
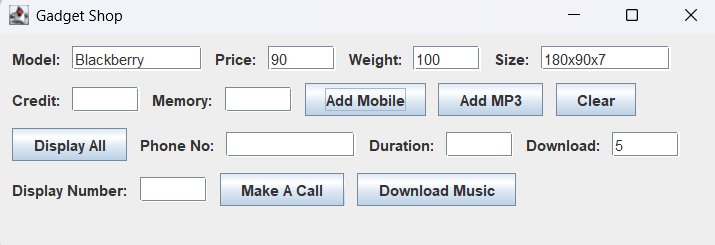
Test 5: Downloading music

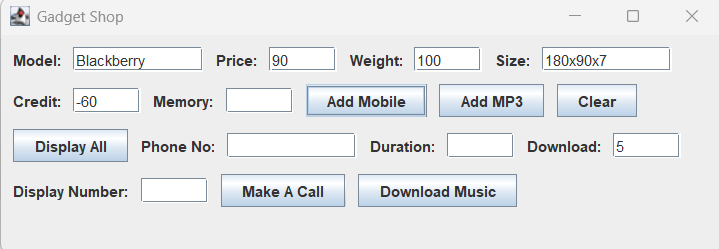


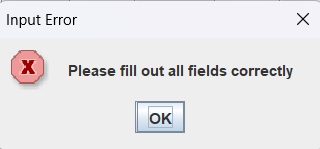
## Test 6: Test that the program can be compiled and run using the command prompt, including a screenshot similar to Figure 1 in the command prompt learning aid.

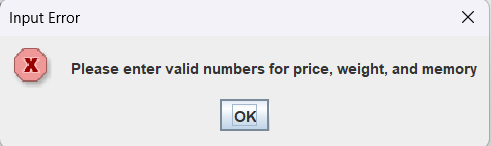
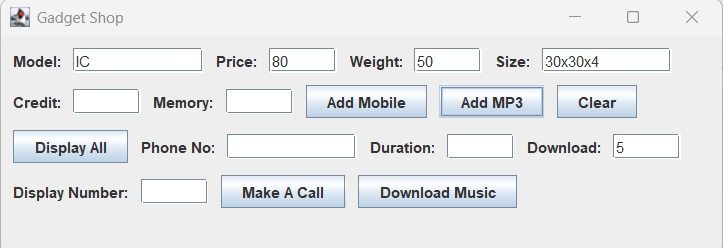


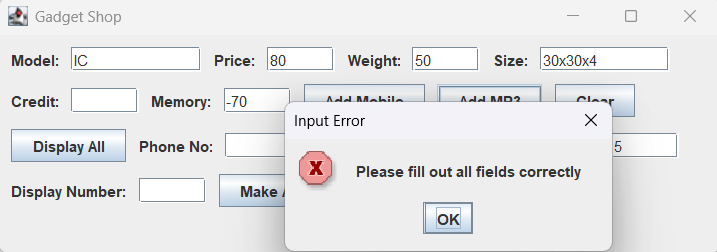
Test 7: Test that appropriate dialog boxes appear when unsuitable values are entered for the display number. 

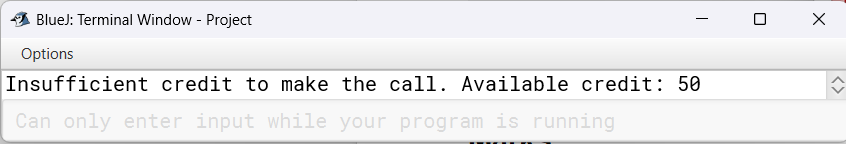
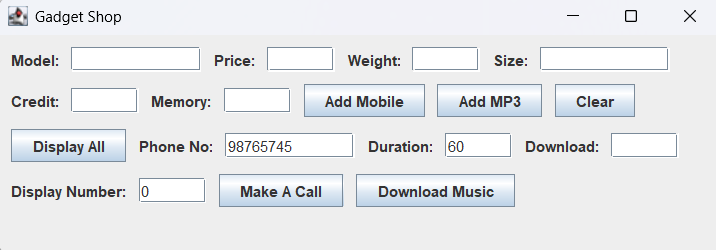


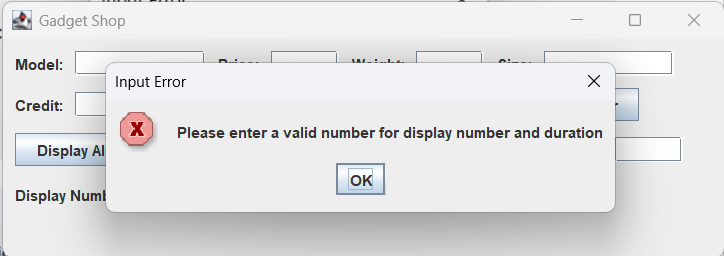
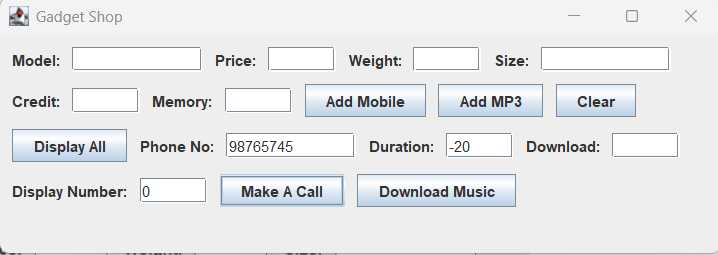


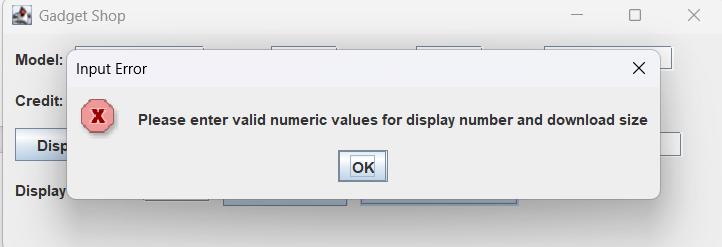
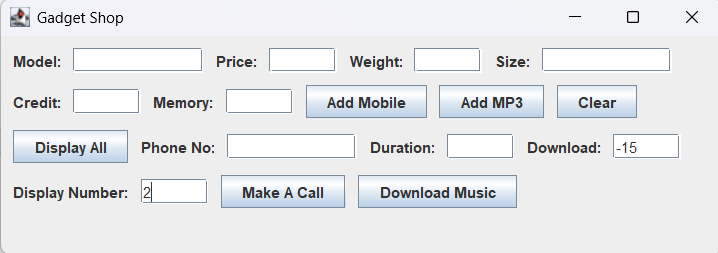






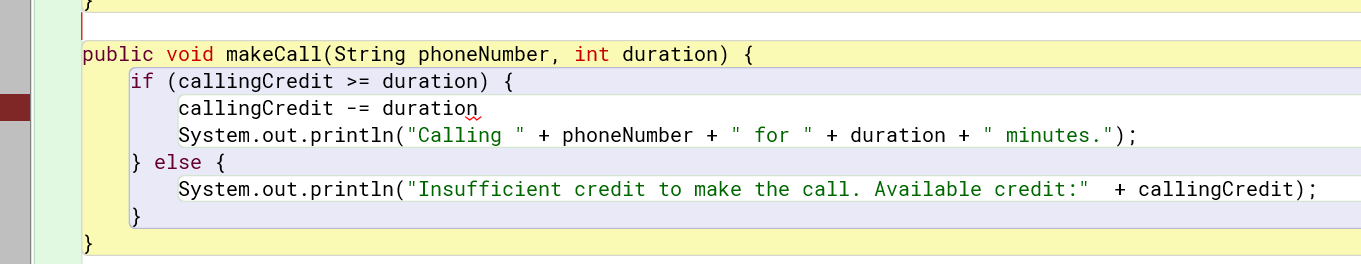


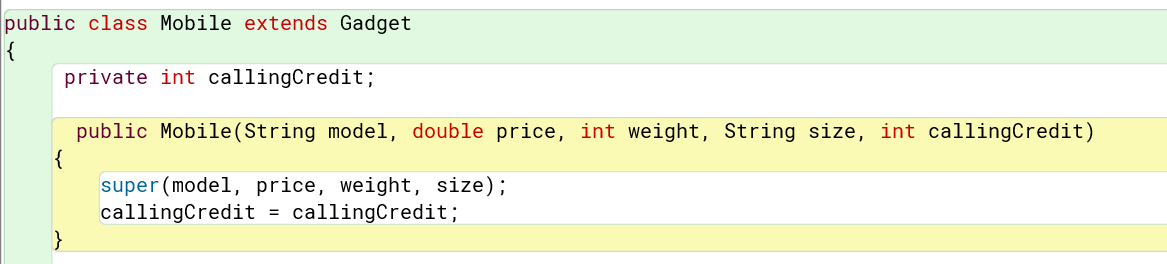




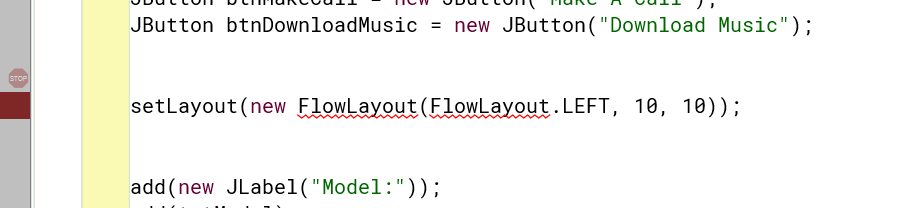
# Error detection

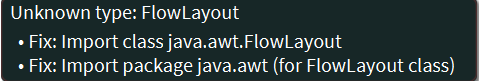






In this case I forgot to add the pointer “this” because I named the variabe and the parameter the same ,witch is a problem when runnig the code.







I did not include the proper package for the syntax I was using.

# Conclusion

When working on the Gadget Shop project, I enhanced my ability to articulate and organise information, develop strategies, and communicate with people. Overall this capstone project was a great experience because I vastly improved my app development proficiency, particularly in programming in Java and Swing for GUIs (graphical user interfaces). Furthermore, I became familiar with user-centric design principles that will be a valuable asset for creating well-designed server applications.

Our senior capstone project allowed me to improve my communication skills – with people and with the computer; it also allowed me to practice providing feedback to other members of the team. I gained a better understanding of thinking through a problem, developing a strategy, and implementing it. Overall this was a rewarding experience because I dramatically improved my ability to develop apps. I learned many technical details of programming in Java and Swing for GUIs (graphical user interfaces). In addition, I learned a lot about user-centred design techniques that I will be able to apply to build server applications that are user-friendly.

One of my biggest goals with this project was shifting to building a GUI, and I believe that accomplishing this was one of the highlights. The biggest first step on this journey was learning how to limit the set of Swing library presets to make it easy to write and visually appealing to interact with. I was able to take advantage of my newfound Swing fluency when building and laying out the GUI. I would often memorise JFrame and JPanel and JTextField and JButton and many more, and a major difficulty I faced was putting them together in an optimal layout. I probably didn’t put in as much time and effort as I should have, but I am satisfied to say that my layout is now functional and aesthetically pleasing. I also had to learn event handling in Swing. Would you believe I cried while writing action listeners for buttons? I did, and this was the highlight of my learning experience with GUIs. Event-driven programming, which is the programming paradigm I used here, is fundamentally different than any procedural programming paradigm I was familiar with before. This part of the project showed me the importance of writing responsive programs — one that responds in real-time in reaction to user inputs. Also, connecting the GUI to the backend logic makes the application interactive. For example, adding a gadget to the inventory, viewing all gadgets, or deleting a certain sale of a gadget would all involve writing methods that not only responded to the user’s GUI inputs, but also affected the data structures that stored the information about gadgets.

The most obvious example is error handling. The application does the properly for input errors from the very beginning, but the handling of those error situations or of other errors that might turn up during the session or even to malicious input from the user, are not really dealt with from the beginning. So a major reframing that happened later on was that I would embed more and more of the application in proper try-catch blocks, and properly vet all input from the user.