Ihor Zlenko

CS4001 Coursework

Student no 22034688

Table of Contents

[Abstract 3](#_Toc455389208)

[[Title here, up to 12 words, on one to two lines] 4](#_Toc455389209)

[[Heading 1] 4](#_Toc455389210)

[[Heading 2] 4](#_Toc455389211)

[[Heading 3] 4](#_Toc455389212)

[References 5](#_Toc455389213)

[Footnotes 6](#_Toc455389214)

[Tables 7](#_Toc455389215)

[Figures 8](#_Toc455389216)

[Tap here to add keywords.]

# Github Link

https://github.com/IHZ0006/GadgetShop.git

# A screenshot of a computer Description automatically generatedUML Class Diagram

# Methods Description

**Gadget.java:**

* getModel – accessor that returns model String type parameter
* getPrice – accessor that returns price double type parameter
* getWeight – accessor that returns weight int type parameter
* getSize – accessor that returns size String type parameter
* display – method, that outputs all for parameter’s values in the console

**Mobile.java:**

* getRemainingMinutes– accessor that returns remainingMinutes int type parameter.
* errorPrompt – takes errorCode int value as an argument(it can be 0 – the user entered 0, 1 – the user doesn’t have enough calling minutes or 2 – the user has entered a value incorrectly). This method is called when there are not enough minutes to make a phone call, so user can add some minutes. We use errorCode, because this method is recursive and calls itself in case of error 0 and 2. This method doesn’t return anything. It is private, because we don’t call it from any other classes.
* makeAPhoneCall – a method that is used to make a call. It takes int phoneNumber and int duration as an arguments, to display them in the message box, when the user is making a call. It checks whether there are enough minutes, if not, it calls errorPrompt method with the argument’s value 1 in the loop, until there will be enough minutes. It doesn’t return anything.
* display – we override Gadget’s method. First, we call display method of the Gadget class and then we print the remaining minutes.

**MP3.java:**

* getMemory – accessor that returns memory double type parameter
* download – puts tracks into the HashMap allTracks. We need to return double value so GadgetShop would understand whether download was successful. It takes String name and double fileSize as arguments and uses them as the key and the value in the HashMap respectively.
* Delete – delete method is used to remove the track from the HashMap. It requires the key value being passed as an argument. Tries to get the memory value by the key from the HashMap allTracks and if successful – ads the value from the HashMap to the availableMemory and deletes the value from there.
* display – we override Gadget’s method. First, we call display method of the Gadget class and then we print the remaining minutes. This method doesn’t return anything.

# Pseudocode

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

*Because I used a different approach from the one described in the assessment’s documentation, this one will be quite simple due to the fact that I check whether the values entered by the user are present and correct right in the actionPerformed block of code instead of writing methods to get the values from each field*

PROGRAM getDisplayNumber:

Get the text from the 'displayNumber' text field;

Convert the text to an integer;

END.

* **Adding a mobile:**

PROGRAM addMobile:

TRY:

Get and convert text inputs for Mobile parameters (model, price, weight, size, credit) and create new Mobile object using the provided attributes;

Add the new Mobile object to the mobiles list;

Display success message;

Use clear method to clear all the fields;

ENDTRY;

CATCH:

Display error message;

ENDCATCH;

END.

* **Adding an MP3:**

PROGRAM addMP3:

TRY:

Get and convert text inputs for MP3 parameters (model, price, weight, size, memory) and create new MP3 object using the provided attributes;

Add the new MP3 object to the MP3s list;

Display success message;

Use clear method to clear all the fields;

ENDTRY;

CATCH (Exception):

Display error message;

ENDCATCH;

END.

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

PROGRAM displayAllGadgets:

Print “Mobiles: ”;

FOREACH Mobile in mobiles list:

Call display method of Mobile;

ENDFOREACH;

Print “MP3s: ”;

FOREACH MP3 in MP3s list:

Call display method of MP3;

ENDFOREACH;

END.

* **Making a call:**

PROGRAM makeACall:

TRY:

Get and convert display number to integer and find Mobile object using display number;

Get and convert inputs for call (phone number, duration);

ENDTRY;

CATCH(NumberFormatException):

Display error message about incorrect user input;

ENDCATCH;

CATCH(IndexOutOfBoundsException):

Display error message that phone is not found;

ENDCATCH;

IF (remaining minutes >= duration)

THEN:

Deduct duration from remaining minutes;

Display call details;

ELSE:

Prompt user to add minutes until sufficient;

Deduct duration from remaining minutes;

Display call details;

ENDIF;

Use clear method to clear all the fields;

END.

* **Downloading music:**

PROGRAM makeACall:

TRY:

Get and convert display number to integer and find MP3 object using display number;

Get and convert inputs for download (track name, file size);

ENDTRY;

CATCH(NumberFormatException):

Display error message about incorrect user input;

ENDCATCH;

CATCH(IndexOutOfBoundsException):

Display error message that MP3 is not found;

ENDCATCH;

IF (file size <= available memory)

THEN:

Deduct memory by the file size;

Add the track to the HashMap;

ELSE:

Display error message ;

ENDIF;

Use clear method to clear all the fields;

END.

# GadgetShop Class

*Before we’ll go to this section’s information, I want to mention that provided code parts can seem a bit unstructured due to my approach to the input checks and report structure requirements, but I will make additional comments to make those parts more clear*

**Textboxes, input check with try/catch:**

First, we need to create private variables for text fields as GadgetShop’s parameters, so we can access them in any part of our class later, because I actually create JTextField objects inside the createFrame method and then assign them to those variables.

private JTextField model;

private JTextField price;

private JTextField weight;

private JTextField size;

private JTextField credit;

private JTextField memory;

private JTextField phoneNo;

private JTextField duration;

private JTextField download;

private JTextField trackName;

private JTextField displayNumber;

Here is the block of code where I actually create them and assign to the variables. Also I create labels for each text field:

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

model = new JTextField();

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

price = new JTextField();

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

weight = new JTextField();

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

size = new JTextField();

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

credit = new JTextField();

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

memory = new JTextField();

JLabel phoneNoLabel = new JLabel("Phone No:");

phoneNo = new JTextField();

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

duration = new JTextField();

JLabel downloadLabel = new JLabel("Download:");

download = new JTextField();

JLabel trackNameLabel = new JLabel("Track Name:");

trackName = new JTextField();

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

displayNumber = new JTextField();

To include all those elements in our GUI we need to create a JFrame object, so actually right before the block of code mentioned higher we have three lines to create it, set layout type and get the container of our frame, where all the elements of our GUI will be added:

frame = new JFrame("GadgetShop"); //Creating new JFrame object and assigning it to the frame variable

frame.setLayout(new GridLayout(26,2)); //Setting frame's layout, creating 26 rows and 2 columns

Container content = frame.getContentPane(); //Here we are getting reference to the frame's Container object, that stores all the content of our GUI

Than we add those text fields to the container(*the code will actually show how we add buttons as well, because the order and amount of elements added in this case is important. We need to remember about frame’s layout*), pack that frame object to create window and make this window visible:

//Adding each element to the Container, which we've got by calling getContentPane method

content.add(modelLabel);

content.add(model);

content.add(priceLabel);

content.add(price);

content.add(weightLabel);

content.add(weight);

content.add(sizeLabel);

content.add(size);

content.add(creditLabel);

content.add(credit);

content.add(memoryLabel);

content.add(memory);

content.add(mobileButton);

content.add(mp3Button);

content.add(clearButton);

content.add(displayButton);

content.add(phoneNoLabel);

content.add(phoneNo);

content.add(durationLabel);

content.add(duration);

content.add(downloadLabel);

content.add(download);

content.add(trackNameLabel);

content.add(trackName);

content.add(displayNumberLabel);

content.add(displayNumber);

content.add(callButton);

content.add(downloadButton);

content.add(deleteButton);

frame.pack(); //Here we are creating grafical window with all the elements

frame.setVisible(true); //And making it visible

Because I don’t have separate methods for input checks only, I will provide all the try/catch code blocks. Here is the one for adding a mobile:

try{ //Instead of writing methods to get values from each field, I've decided that it would be better to use methods getting and converting values from the text fields right inside the block of code, where we pass parameters of an object. If values in text fields will be incorrect, then java will throw an error, that's why we use try/catch statement

allMobiles.add(new Mobile(Integer.parseInt(credit.getText()), model.getText(), Double.parseDouble(price.getText()), Integer.parseInt(weight.getText()), size.getText())); //Heare we are trying to add a new Mobile object to the ArrayList

JOptionPane.showMessageDialog(frame, "You've successfuly added a mobile phone"); //If previous line won't throw an error, than this line will work. Otherwise java will just quit the try block and will go to the catch block before reaching this line

clear(); //Using the clear method to empty all the text fields

}

catch(Exception e) //In case of java throwing an error, we'll understand that one or more values weren't entered or were entered incorrectly and we'll tell the user about that

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

Here is the one for adding an MP3:

//Same as case "Add Mobile", but with MP3 objects and allMP3 ArrayList

try{

allMP3.add(new MP3(Integer.parseInt(memory.getText()), model.getText(), Double.parseDouble(price.getText()), Integer.parseInt(weight.getText()), size.getText()));

JOptionPane.showMessageDialog(frame, "You've successfuly added an MP3 player");

clear();

}

catch(Exception e)

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

Here is the one for making a call:

try{ //If java will throw a NumberFormatException error after trying to call get or makeAPhoneCall methods, than the user typed something in the incorect format or/and left the field(s) empty. If it will throw the IndexOutOfBoundsException after trying to call the get method, then there is no such index in the ArrayList

Mobile current = allMobiles.get(Integer.parseInt(displayNumber.getText())); //Here we are getting Mobile object form the ArrayList allMobiles by the display number from the text field

current.makeAPhoneCall(Integer.parseInt(phoneNo.getText()), Integer.parseInt(duration.getText())); //Calling the makeAPhoneCall Method of the Mobile class

JOptionPane.showMessageDialog(frame, "You've successfuly made a call. Remaining calling minutes: " + current.getRemainingMinutes());

clear();

}

catch(NumberFormatException e)

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

catch(IndexOutOfBoundsException e)

{

JOptionPane.showMessageDialog(frame, "Mobile phone not found");

}

Because we’ve used the makeAPhoneCall method, I want to show try/catch block inside errorPrompt method, which is called by makeAPhoneCall if therr are not enough calling minutes and use it to top user’s balance up:

//errorPrompt method is used for the situation, when there is not enough miutes to make a call

private void errorPrompt(int errorCode)

{

int remainingMinutesPrompt = 0; //This variable will be used for the added minutes. I've used a separate variable instead of adding a value straight to the remainingMinutes to check whether the user entered 0

try //For the case when user enters not an integer value, we have a try-catch block

{

switch(errorCode) //We use switch statement on the errorCode value to compare it with values that go after case operator. There are 3 possible error codes: 0, 1 and 2. 0 is used when the user entered a 0, 1 is used when there is insufficient credit for a call, 2 is used when user entered the value which is not an integer

{

case 0: remainingMinutesPrompt = Integer.parseInt(JOptionPane.showInputDialog(GadgetShop.frame, "Enter the value greater than 0")); break;

case 1: remainingMinutesPrompt = Integer.parseInt(JOptionPane.showInputDialog(GadgetShop.frame, "Insufficient credit. Please, enter the desired number of minutes to top up")); break;

case 2: remainingMinutesPrompt = Integer.parseInt(JOptionPane.showInputDialog(GadgetShop.frame, "Please, enter the correct value")); break;

}

if(remainingMinutesPrompt > 0) //If user topped up the credit, we add those remaining minutes to the main remainingMinutes variable

{

remainingMinutes += remainingMinutesPrompt;

}

else //If not, we have a recursive call of this function with an errorCode value equal to 0

{

errorPrompt(0);

}

}

catch(Exception e)

{

errorPrompt(2); //Here we have a recursive call of this function with an errorCode value equal to 2

}

}

Here is the block for downloading music:

try{ //try/catch statements work the same as with "Make A Call" button block

MP3 current = allMP3.get(Integer.parseInt(displayNumber.getText()));

boolean checkDownload = current.download(trackName.getText(), Double.parseDouble(download.getText())); //The download method returns boolean value. If our download was successful, than it returns true, otherwise - false

if(checkDownload == true) //We check whether the download was successful, if it was, we notify the user and show him the amount of space left

{

JOptionPane.showMessageDialog(frame, "You've successfuly downloaded music. Remaining memory: " + current.getMemory());

}

clear();

}

catch(NumberFormatException e)

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

catch(IndexOutOfBoundsException e)

{

JOptionPane.showMessageDialog(frame, "MP3 player not found");

}

Also I’ve created the method to delete music and here are try/catch statements to call it:

//This one works the same as "Download Music", but calls the delete method and instead of checking remaining memory, we actually check if the track's name exist in the HashMap of the MP3 class. There is a try/catch block for that inside the delete method

try{

MP3 current = allMP3.get(Integer.parseInt(displayNumber.getText()));

boolean checkTrackAvailability = current.delete(trackName.getText());

if(checkTrackAvailability == true)

{

JOptionPane.showMessageDialog(frame, "You've successfuly deleted music. Remaining memory: " + current.getMemory());

}

clear();

}

catch(NumberFormatException e)

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

catch(IndexOutOfBoundsException e)

{

JOptionPane.showMessageDialog(frame, "MP3 player not found");

}

And the try/catch block inside the delete method itself:

//delete method is used to remove the track from the HashMap. It requires the key value being passed as an argument

public boolean delete(String name)

{

try{ //If we'll use the name value as a get method's argument which is not a key in the allTracks HashMap, java will throw an error

double fileSize = allTracks.get(name); //Trying to get the file size from the HashMap by the name key

availableMemory += fileSize; //Adding the amount of available memory that is equal to the file size

allTracks.remove(name); //Removing the value form the HashMap

return true; //Because we successfully deleted the track, we return true

}

catch(Exception e) //If the error is thrown, we need to tell the user that the track is not found

{

JOptionPane.showMessageDialog(GadgetShop.frame, name + " not found");

return false; //Because operation wasn't successful, we return false

}

}

**Buttons and Action Performed Methods:**

Same as we’ve created private variables for placing JTextField objects, we need to do this for JButton objects:

private JButton mobileButton;

private JButton mp3Button;

private JButton clearButton;

private JButton displayButton;

private JButton callButton;

private JButton downloadButton;

private JButton deleteButton;

Block of code where we create and assign buttons to the variables. Using addActionListener method on them, passing themselves as a parameter, we would know when the user clicked on them:

mobileButton = new JButton("Add Mobile");

mobileButton.addActionListener(this);

mp3Button = new JButton("Add MP3");

mp3Button.addActionListener(this);

clearButton = new JButton("Clear");

clearButton.addActionListener(this);

displayButton = new JButton("Display All");

displayButton.addActionListener(this);

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

callButton.addActionListener(this);

downloadButton = new JButton("Download Music");

downloadButton.addActionListener(this);

deleteButton = new JButton("Delete Music");

deleteButton.addActionListener(this);

*The block of code where we add those buttons to the frame is on the pages 11 and 12.*

And the whole actionPerformed method:

public void actionPerformed(ActionEvent event)

{

String buttonType = event.getActionCommand(); //getActionCommand method basically gives us the string which we've passed as a parameter of a button that the user clicked in this case

switch (buttonType) //We use switch statement on the buttonType value to compare it with values that go after the case operator. Each time after the user clicked on any of the buttons, we check which button he clicked on exactly.

{

case "Add Mobile": //Block of code to add a Mobile type object to the allMobiles ArrayList

try{ //Instead of writing methods to get values from each field, I've decided that it would be better to use methods getting and converting values from the text fields right inside the block of code, where we pass parameters of an object. If values in text fields will be incorrect, then java will throw an error, that's why we use try/catch statement

allMobiles.add(new Mobile(Integer.parseInt(credit.getText()), model.getText(), Double.parseDouble(price.getText()), Integer.parseInt(weight.getText()), size.getText())); //Here we are trying to add a new Mobile object to the ArrayList

JOptionPane.showMessageDialog(frame, "You've successfuly added a mobile phone"); //If previous line won't throw an error, than this line will work. Otherwise java will just quit the try block and will go to the catch block before reaching this line

clear(); //Using the clear method to empty all the text fields

}

catch(Exception e) //In case of java throwing an error, we'll understand that one or more values weren't entered or were entered incorrectly and we'll tell the user about that

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

break;

case "Add MP3": //Same as case "Add Mobile", but with MP3 objects and allMP3 ArrayList

try{

allMP3.add(new MP3(Integer.parseInt(memory.getText()), model.getText(), Double.parseDouble(price.getText()), Integer.parseInt(weight.getText()), size.getText()));

JOptionPane.showMessageDialog(frame, "You've successfuly added an MP3 player");

clear();

}

catch(Exception e)

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

break;

case "Clear": clear(); break; //Calling the clear method if the "Clear" button is clicked

case "Display All": displayAll(); break; //Calling the displayAll method if the "Display All" button is clicked

case "Make A Call": //If user clicks the button "Make A Call", we are trying to get a mobile from the ArrayList and then call its makeAPhoneCall method

try{ //If java will throw a NumberFormatException error after trying to call get or makeAPhoneCall methods, than the user typed something in the incorect format or/and left the field(s) empty. If it will throw the IndexOutOfBoundsException after trying to call the get method, then there is no such index in the ArrayList

Mobile current = allMobiles.get(Integer.parseInt(displayNumber.getText())); //Here we are getting Mobile object form the ArrayList allMobiles by the display number from the text field

current.makeAPhoneCall(Integer.parseInt(phoneNo.getText()), Integer.parseInt(duration.getText())); //Calling the makeAPhoneCall Method of the Mobile class

JOptionPane.showMessageDialog(frame, "You've successfuly made a call. Remaining calling minutes: " + current.getRemainingMinutes());

clear();

}

catch(NumberFormatException e)

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

catch(IndexOutOfBoundsException e)

{

JOptionPane.showMessageDialog(frame, "Mobile phone not found");

}

break;

case "Download Music": //If user clicks the button "Download Music", we are trying to get an MP3 from the ArrayList and then call its download method

try{// try/catch statements work the same as with "Make A Call" button block

MP3 current = allMP3.get(Integer.parseInt(displayNumber.getText()));

boolean checkDownload = current.download(trackName.getText(), Double.parseDouble(download.getText())); //The download method returns boolean value. If our download was successful, than it returns true, otherwise - false

if(checkDownload == true) //We check whether the download was successful, if it was, we notify the user and show him the amount of space left

{

JOptionPane.showMessageDialog(frame, "You've successfuly downloaded music. Remaining memory: " + current.getMemory());

}

clear();

}

catch(NumberFormatException e)

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

catch(IndexOutOfBoundsException e)

{

JOptionPane.showMessageDialog(frame, "MP3 player not found");

}

break;

case "Delete Music": //This one works the same as "Download Music", but calls the delete method and instead of checking remaining memory, we actually check if the track's name exist in the HashMap of the MP3 class. There is a try/catch block for that inside the delete method

try{

MP3 current = allMP3.get(Integer.parseInt(displayNumber.getText()));

boolean checkTrackAvailability = current.delete(trackName.getText());

if(checkTrackAvailability == true)

{

JOptionPane.showMessageDialog(frame, "You've successfuly deleted music. Remaining memory: " + current.getMemory());

}

clear();

}

catch(NumberFormatException e)

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

catch(IndexOutOfBoundsException e)

{

JOptionPane.showMessageDialog(frame, "MP3 player not found");

}

break;

}

}

}

# Command Prompt Test

# A screenshot of a computer Description automatically generated

# GUI

A screenshot of a computer

Description automatically generated

# Testing

**Test 1(Adding a mobile to the array list):**

Succesfully added mobile:

A screenshot of a computer

Description automatically generatedA screenshot of a message box

Description automatically generated

A screenshot of a computer

Description automatically generated Wrong Input:

**Test 2(Adding an MP3 to the array list):**

A screenshot of a computer

Description automatically generatedA screenshot of a message

Description automatically generatedSuccesfully added MP3:

**A screenshot of a computer screen

Description automatically generated**

Wrong Input:

**A screenshot of a phone number

Description automatically generatedTest 3(Displaying the details of all of the gadgets in the array list):**

**Test 4(Making a call):**

Call with insufficient credit:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedEntering 0:Entering “q”:

A screenshot of a phone

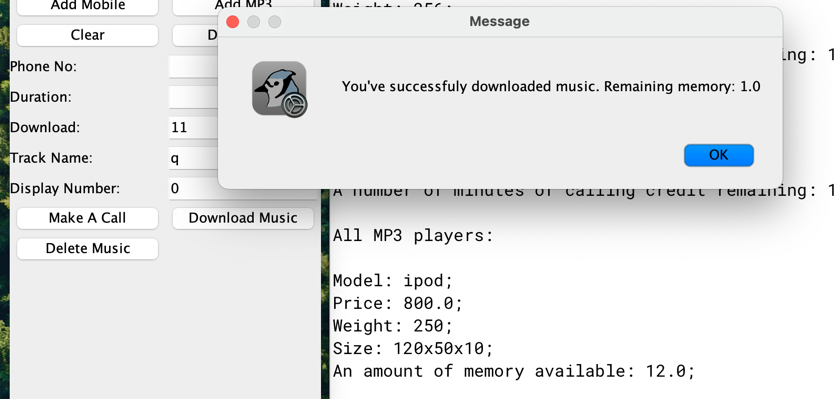
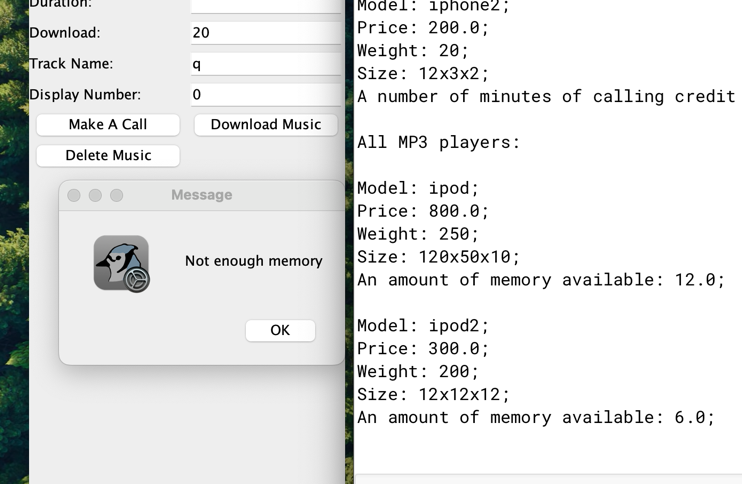
Description automatically generatedA screenshot of a message

Description automatically generatedEntering right value:

A screenshot of a phone

Description automatically generatedEntering display number that doesn’t exist:

**Test 5(Downloading music):**

Not enough memory: Successful download:



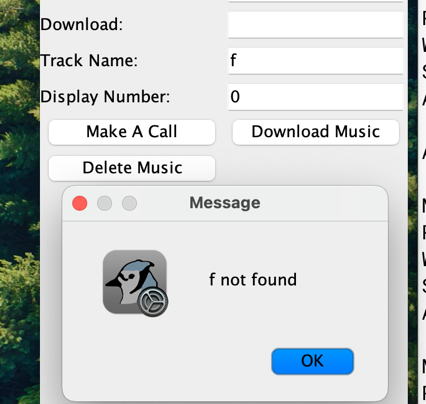
MP3 Not Found:

**Test 6(Command Prompt Test):**

*Was covered on page 26*

**Test 7(Delete music):**

**A screenshot of a computer

Description automatically generated**Track Not Found: Success:

# Error detection

1. The most major error I’ve encountered was when at the very start I’ve tried to make the user input with a Scanner object through a console. The GUI window would just freeze without responding, terminal would print message about insufficient credit, but wouldn’t allow any input.

A screenshot of a computer

Description automatically generatedI realized that the problem was because of single thread and that’s why I’ve implemented multi-threaded solution. *I will provide a new class for the thread, changed make a call case statement, changed errorPrompt method and changed makeAPhoneCall method(full code will be on the github).*

New class:

public class callThread extends Thread //creating callThread class for overriding run method of a Thread class

{

public void run() //method overriding

{

Mobile current = allMobiles.get(Integer.parseInt(displayNumber.getText())); //getting Mobile from ArrayList

current.makeAPhoneCall(Integer.parseInt(phoneNo.getText()), Integer.parseInt(duration.getText())); //Making a call

System.out.println("You've successfuly made a call. Remaining calling minutes: " + current.getRemainingMinutes());

clear();

}

}

Case block:

case "Make A Call": //If user clicks the button "Make A Call", we are trying to get a mobile from the ArrayList and then call its makeAPhoneCall method

try{ //If java will throw a NumberFormatException error after trying to call get or makeAPhoneCall methods, than the user typed something in the incorect format or/and left the field(s) empty. If it will throw the IndexOutOfBoundsException after trying to call the get method, then there is no such index in the ArrayList

Thread callAction = new callThread(); //Creating new callThread object

callAction.start(); //Creating new thread by using start method

callAction.join(); //Main thread would wait for the callAction thread to end, so user wouldn't be able to call any methods that require console input

}

catch(NumberFormatException e)

{

JOptionPane.showMessageDialog(frame, "Please, enter all the values that are needed in a correct format!");

}

catch(IndexOutOfBoundsException e)

{

JOptionPane.showMessageDialog(frame, "Mobile phone not found");

}

catch(InterruptedException e) // we need this block becase java makes you handle interuption exception, if you use join method

{

}

break;

makeAPhoneCall method:

//Method to make a call

public void makeAPhoneCall(int phoneNumber, int duration)

{

if(remainingMinutes >= duration) //If remaininingMinutes value is bigger than or equal tothe duration of a call, we decrease the remainingMinutes variable by the duration variable and print the phone number and the duration of a call

{

remainingMinutes -= duration;

JOptionPane.showMessageDialog(GadgetShop.frame, "Number: " + phoneNumber + "\nDuration: " + duration + " minute(s)");

}

else //If not - we call the errorPrompt function giving a number 1 as na argument's value in a loop, while the remainingMinutes value is less than duration and only then we decrease the remainingMinutes variable by the duration variable and print the phone number and the duration of a call

{

while(remainingMinutes < duration)

{

errorPrompt(1);

}

remainingMinutes -= duration;

System.out.println("Number: " + phoneNumber + "\nDuration: " + duration + " minute(s)");

}

}

errorPrompt method:

private void errorPrompt(int errorCode)

{

Scanner scanNum; //Creating Scanner variable

int remainingMinutesPrompt = 0; //This variable will be used for the added minutes. I've used a separate variable instead of adding a value straight to the remainingMinutes to check whether the user entered 0

try //For the case when user enters not an integer value, we have a try-catch block

{

switch(errorCode) //We use switch statement on the errorCode value to compare it with values that go after case operator. There are 3 possible error codes: 0, 1 and 2. 0 is used when the user entered a 0, 1 is used when there is insufficient credit for a call, 2 is used when user entered the value which is not an integer

{

case 0:

System.out.println("Enter the value greater than 0");

scanNum = new Scanner(System.in); //Creating Scanner object for user input from console

remainingMinutesPrompt = scanNum.nextInt(); //Getting number from console user input

break;

case 1:

System.out.println("Insufficient credit. Please, enter the desired number of minutes to top up");

scanNum = new Scanner(System.in);

remainingMinutesPrompt = scanNum.nextInt();

break;

case 2:

System.out.println("Please, enter correct value");

scanNum = new Scanner(System.in);

remainingMinutesPrompt = scanNum.nextInt();

break;

}

if(remainingMinutesPrompt > 0) //If user topped up the credit, we add those remaining minutes to the main remainingMinutes variable

{

remainingMinutes += remainingMinutesPrompt;

}

else //If not, we have a recursive call of this function with an errorCode value equal to 0

{

errorPrompt(0);

}

}

catch(Exception e)

{

errorPrompt(2); //Here we have a recursive call of this function with an errorCode value equal to 2

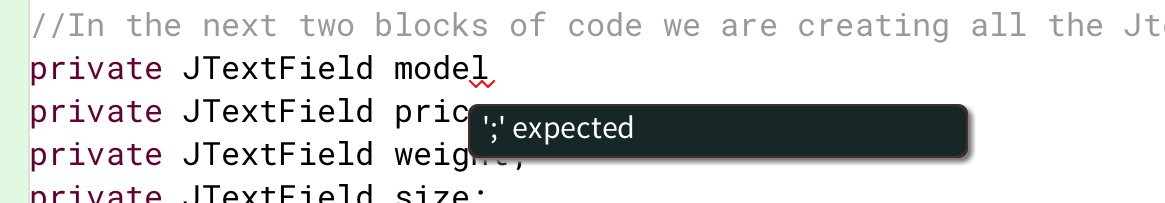
}

}

A screen shot of a computer

Description automatically generated2. I often forgot semicolons

3. Spelling errors:



# Conclusion

Overall I haven’t learned anything new, because I was familiar with all of those concepts already, but looking into multi-threading was a bit fun. Also I’ve practiced my documentation reading skills

# Full code