MIT World Peace University

Object Oriented Programming with Java and C++
Second Year B. Tech, Semester 1

Mini Project with Java - Price Guessing Game
"How Much?"

Project Report

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20

November 22, 2022

Contents

1 Introduction **1**

2 Methodology Used **1**

3 Platform **1**

4 Requirements **1**

5 Installation and Running **1**

6 Database Management **2**

7 Unique Features **2**

7.1 Dark Mode 2

7.2 Data Backup 2

7.3 Web Scrapping 2

7.4 Working Login and Account Creation 2

8 Screenshots of the Project **2**

8.1 The Login Page 2

8.2 The Menu Screen 3

8.3 The Topic Selection Screen 3

8.4 The Highscore Screen 4

8.5 The Help and About 4

8.6 The Game Over Screen 5

9 Output Files Produced **6**

10 Walk-Through of the Files **6**

10.1 TopicsFrame.java 6

10.2 MongoManager.java 6

10.3 MenuFrame.java 6

10.4 Main.java 6

10.5 LoginFrame.java 6

10.6 HighscoreFrame.java 6

10.7 HelpFrame.java 6

10.8 GameOverFrame.java 6

10.9 GameFrame.java 6

10.10 DataBaseManager.java 6

10.11 Colors.java 6

10.12 BackgroundPanel.java 6

10.13 AmazonScrapper.java 6

11 Conclusion and Topics Learnt **6**

12 Code File Main.java **6**

1 Introduction

2 Methodology Used

3 Platform

4 Requirements

5 Installation and Running

Download the .jar file from the releases when it is released that is. Navigate there from your terminal

```
java -jar ./How_Much.jar
```

6 Database Management

7 Unique Features

7.1 Dark Mode

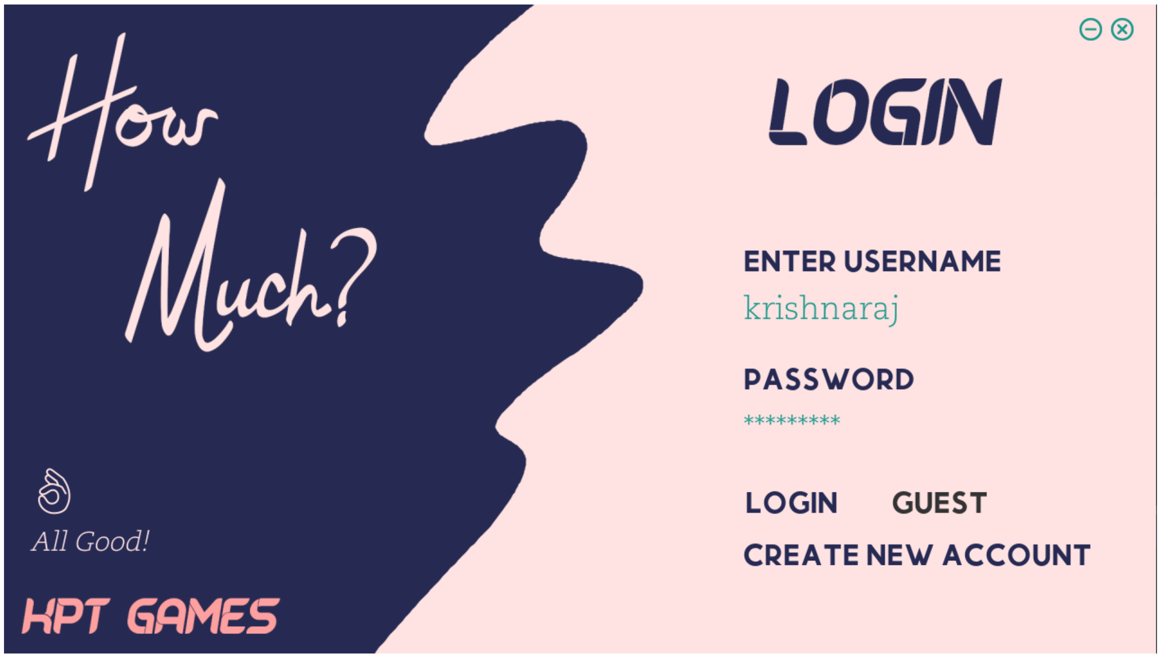
7.2 Data Backup

7.3 Web Scrapping

7.4 Working Login and Account Creation

8 Screenshots of the Project

8.1 The Login Page



ENTER USERNAME
krishnaraj

PASSWORD
****

Figure 1: The Login page after a successful login

8.2 The Menu Screen

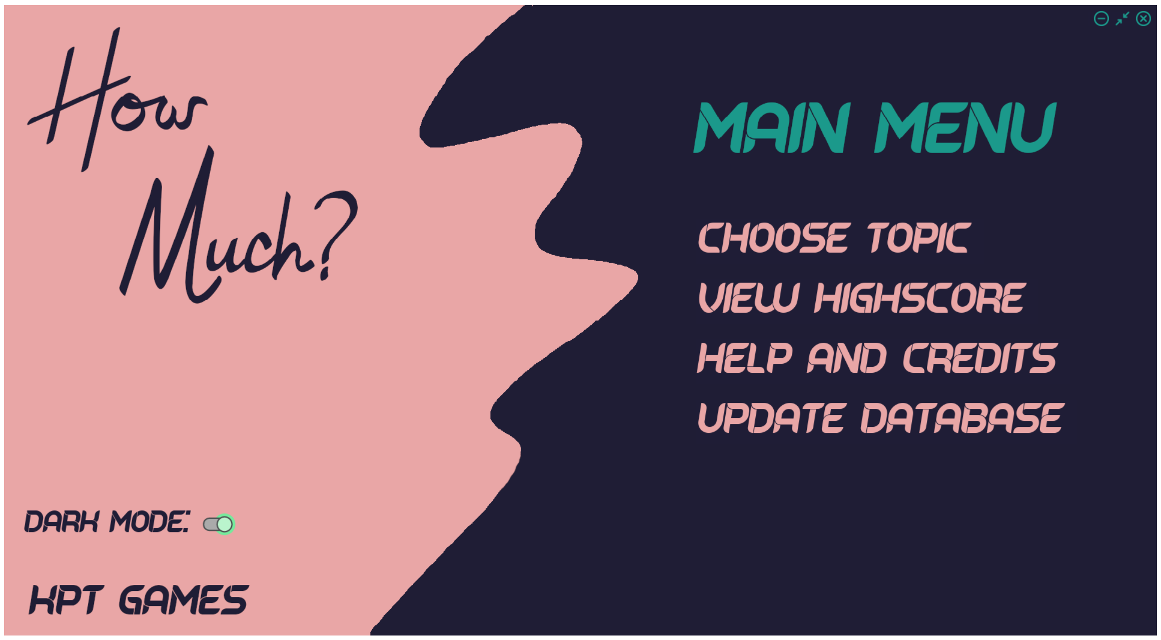


Figure 2:

8.3 The Topic Selection Screen

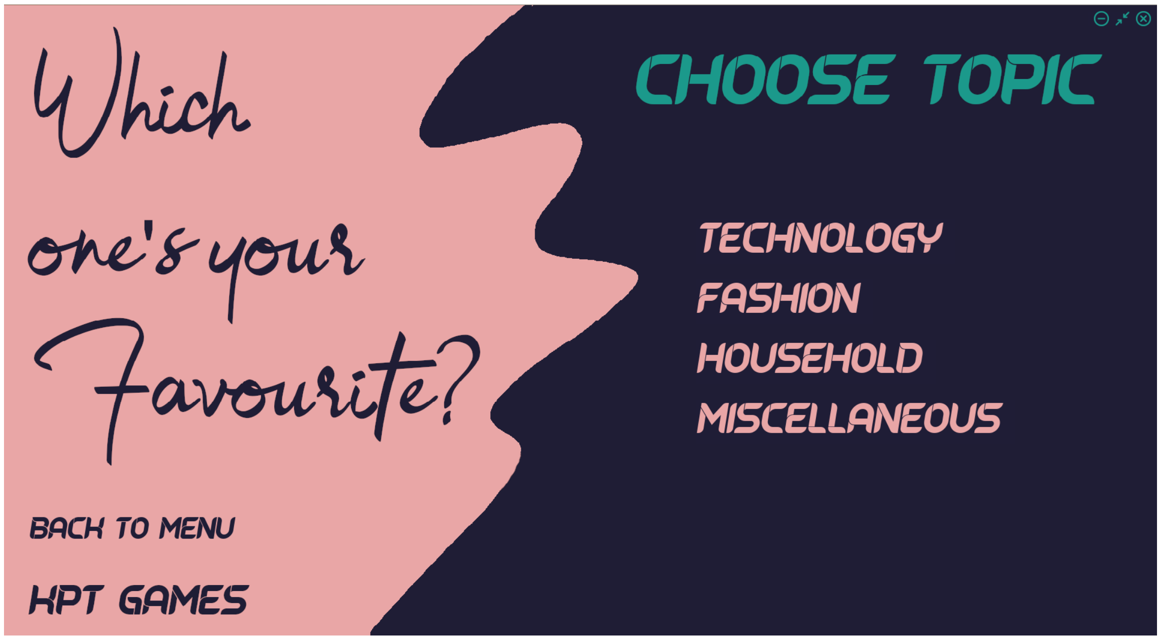
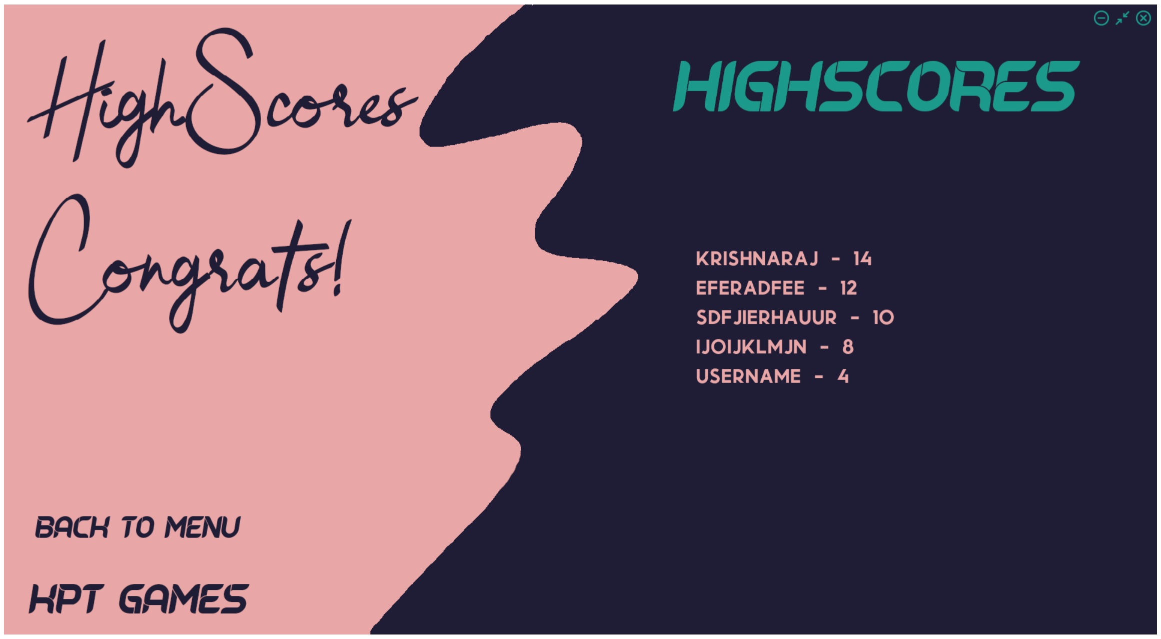


Figure 3: The Login page after a successful login

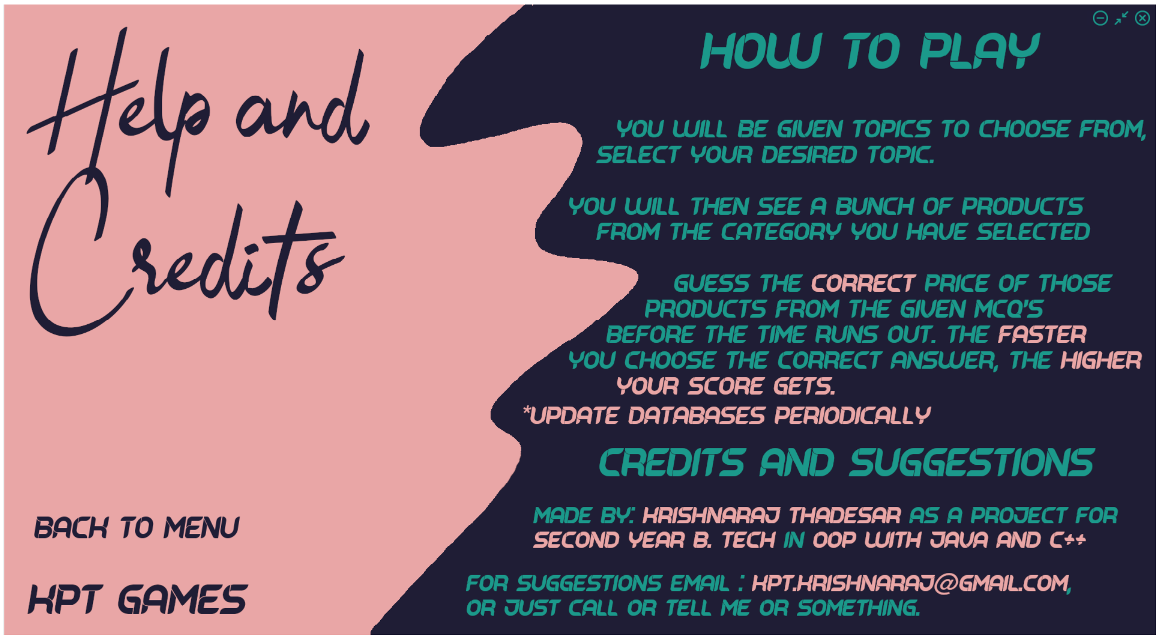
8.4 The Highscore Screen



KRISHNARAJ - 14
EFERADFEE - 12
SDFJIERHAUUR - 10
IJOIJKLMJN - 8
USERNAME - 4

Figure 4: The Login page after a successful login

8.5 The Help and About



HOW TO PLAY

YOU WILL BE GIVEN TOPICS TO CHOOSE FROM, SELECT YOUR DESIRED TOPIC.

YOU WILL THEN SEE A BUNCH OF PRODUCTS FROM THE CATEGORY YOU HAVE SELECTED

GUESS THE CORRECT PRICE OF THOSE PRODUCTS FROM THE GIVEN MCQ'S BEFORE THE TIME RUNS OUT. THE FASTER YOU CHOOSE THE CORRECT ANSWER, THE HIGHER YOUR SCORE GETS.

*UPDATE DATABASES PERIODICALLY

CREDITS AND SUGGESTIONS

MADE BY: KRISHNARAJ THADESAR AS A PROJECT FOR SECOND YEAR B. TECH IN OOP WITH JAVA AND C++

FOR SUGGESTIONS EMAIL : KPT.KRISHNARAJ@GMAIL.COM, OR JUST CALL OR TELL ME OR SOMETHING.

Figure 5: The Login page after a successful login

8.6 The Game Over Screen

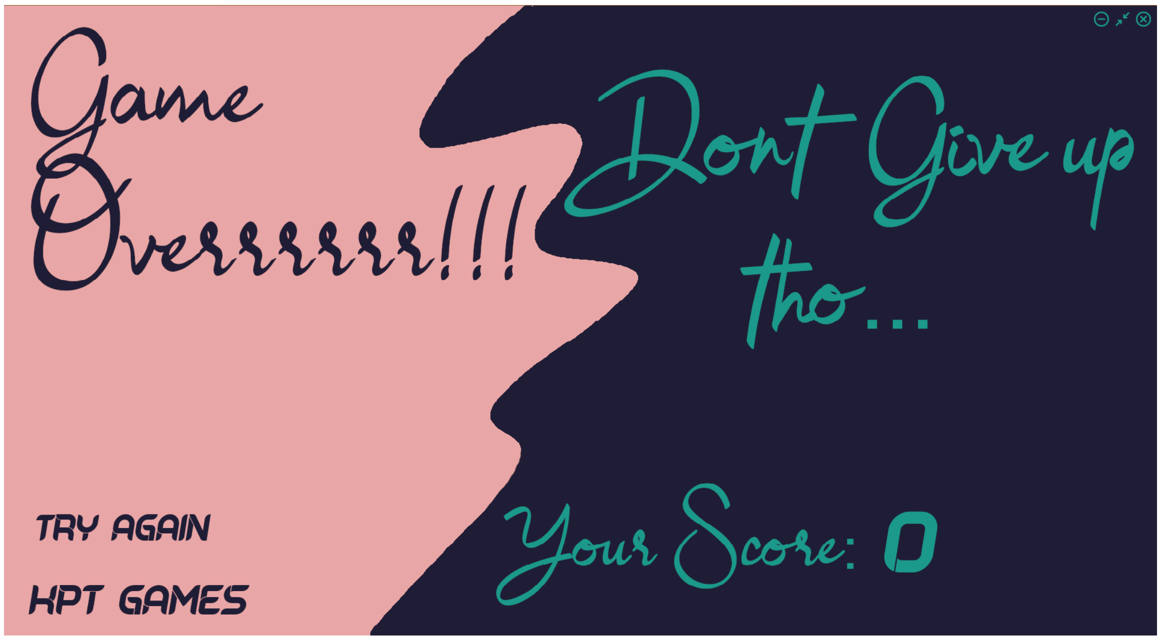


Figure 6: The Login page after a successful login

9 Output Files Produced

10 Walk-Through of the Files

10.1 TopicsFrame.java

10.2 MongoManager.java

10.3 MenuFrame.java

10.4 Main.java

10.5 LoginFrame.java

10.6 HighscoreFrame.java

10.7 HelpFrame.java

10.8 GameOverFrame.java

10.9 GameFrame.java

10.10 DataBaseManager.java

10.11 Colors.java

10.12 BackgroundPanel.java

10.13 AmazonScrapper.java

11 Conclusion and Topics Learnt

12 Code File Main.java

```java
package org.howmuch;

import org.xml.sax.SAXException;
import javax.swing.*;
import javax.xml.parsers.ParserConfigurationException;
import java.awt.*;
import java.io.*;
import java.time.LocalDate;

// You have to extend the thread class to create a new thread for running the
    databases.
public class Main extends Thread {

    // Statically defining important variables used throughout the game. They are
    // statically defined coz they are used by other classes very often.
    public static String[] Topics = new String[] { "Technology", "Fashion", "
    Household", "Miscellaneous" };
    public static String currentTopic = Topics[0];
    static final int WIDTH = 1280, HEIGHT = 720;
    static boolean maximized = false, isGuest = true, grantAccess = false,
    isLocalDatabaseUpToDate = false,
            isMongoUpToDate = false, usingMongo = false;

    // Declaring Objects of other classes that we are going to call from main.
    static LoginFrame loginFrame;
    static MenuFrame menuFrame;
    static HelpFrame helpFrame;
    static HighscoreFrame highscoreFrame;
    static TopicsFrame topicsFrame;
    static GameFrame gameFrame;
    static GameOverFrame gameOverFrame;

    static Font buttonFont, textFont, password_font, options_font, emoji_font;
    static JButton exit_btn, resize_btn, minimize_btn;
    static JPanel basicButtons_pnl;

    // These are the icons from where we get the resize, exit and the minimize
    // button. They are custom made coz they look better,
    // eliminate the need for the titlebar making the UI look cleaner, albeit less
    // useful.
    // They also let you have full control over what you want to do when they are
    // pressed, and what you wanna call, which you cant do without them.
    // You can also control now exactly the resizing behaviour of your software.
    static ImageIcon exit = new ImageIcon("src/main/resources/icons/circle_delete.
png");
    static Image exit_image = exit.getImage().getScaledInstance(25, 25, Image.
SCALE_SMOOTH);
    static ImageIcon minimize = new ImageIcon("src/main/resources/icons/
circle_minus.png");
    static Image minimize_image = minimize.getImage().getScaledInstance(25, 25,
Image.SCALE_SMOOTH);
    static ImageIcon resizeUp = new ImageIcon("src/main/resources/icons/resize_3.
png");
    static Image resizeUp_image = resizeUp.getImage().getScaledInstance(25, 25,
Image.SCALE_SMOOTH);
    static ImageIcon resizeDown = new ImageIcon("src/main/resources/icons/resize_4
.png");
    static Image resizeDown_image = resizeDown.getImage().getScaledInstance(25,
25, Image.SCALE_SMOOTH);

    /**
     * Creates fonts by instantiating the font objects with their respective fonts
     * stored locally. Static and used everywhere. Its an important function and
     * gets called in almost every class constructor.
     */
    public static void createFonts() {
        try {
            GraphicsEnvironment ge = GraphicsEnvironment.
getLocalGraphicsEnvironment();

            // Used for Buttons Almost everywhere.
            buttonFont = Font
                    .createFont(Font.TRUETYPE_FONT, new File("src/main/resources/
Fonts/BelgradoItalic-OVArd.ttf"))
                    .deriveFont(50f);
            // Used Mostly on the Login Page.
            textFont = Font.createFont(Font.TRUETYPE_FONT, new File("src/main/
resources/Fonts/MomcakeBold-WyonA.otf"))
                    .deriveFont(50f);
            // Used for password Entering
            password_font = Font
                    .createFont(Font.TRUETYPE_FONT, new File("src/main/resources/
Fonts/CaeciliaLTPro45Light.TTF"))
                    .deriveFont(35f);
            // Used only for Emojis
            emoji_font = Font.createFont(Font.TRUETYPE_FONT,
                    new File("src/main/resources/Fonts/NotoEmoji-VariableFont_wght
.ttf")).deriveFont(35f);
            // Used to show the Price, needs to contain the Rupee symbol
            options_font = Font
                    .createFont(Font.TRUETYPE_FONT, new File("src/main/resources/
Fonts/ProductSans-Regular.ttf"))
                    .deriveFont(35f);

            // registering them locally, not required.
            ge.registerFont(textFont);
            ge.registerFont(buttonFont);
            ge.registerFont(password_font);
            ge.registerFont(emoji_font);
            ge.registerFont(options_font);

        } catch (FontFormatException | IOException e) {
            e.printStackTrace();
            System.out.println("Couldnt create the fonts. ");
        }
    }

    /*
     * Function to Create the resize, minimize and the exit button, they are all
     * placed in a panel, so that you can move them around easily without the
hassle
     * of moving each thing. Just move the panel. Here we define them.
     */
    public static void createBasicButtonPanel() {
        basicButtons_pnl = new JPanel();
        FlowLayout fl = new FlowLayout(FlowLayout.LEFT, 10, 0);
        basicButtons_pnl.setLayout(fl);

        exit_btn = new JButton();
        exit_btn.setIcon(new ImageIcon(exit_image));
        exit_btn.setAlignmentY(Box.CENTER_ALIGNMENT);
        exit_btn.setAlignmentX(Box.CENTER_ALIGNMENT);
        exit_btn.setBounds(new Rectangle(25, 25));
        exit_btn.setFont(buttonFont.deriveFont(44f));
        exit_btn.setFocusPainted(false);
        exit_btn.setContentAreaFilled(false);
        exit_btn.setOpaque(true);
        exit_btn.setBorder(null);

        resize_btn = new JButton();
        if (Main.maximized) {
            resize_btn.setIcon(new ImageIcon(resizeDown_image));
        } else {
            resize_btn.setIcon(new ImageIcon(resizeUp_image));
        }
        resize_btn.setAlignmentY(Box.CENTER_ALIGNMENT);
        resize_btn.setAlignmentX(Box.CENTER_ALIGNMENT);
        resize_btn.setBounds(new Rectangle(25, 25));
        resize_btn.setFont(buttonFont.deriveFont(44f));
        resize_btn.setFocusPainted(false);
        resize_btn.setContentAreaFilled(false);
```

```java
        resize_btn.setOpaque(true);
        resize_btn.setBorder(null);

        minimize_btn = new JButton();
        minimize_btn.setIcon(new ImageIcon(minimize_image));
        minimize_btn.setAlignmentY(Box.CENTER_ALIGNMENT);
        minimize_btn.setAlignmentX(Box.CENTER_ALIGNMENT);
        minimize_btn.setBounds(new Rectangle(25, 25));
        minimize_btn.setFont(buttonFont.deriveFont(44f));
        minimize_btn.setFocusPainted(false);
        minimize_btn.setContentAreaFilled(false);
        minimize_btn.setOpaque(true);
        minimize_btn.setBorder(null);

        // Adding them to the panel here.
        basicButtons_pnl.add(minimize_btn);
        basicButtons_pnl.add(resize_btn);
        basicButtons_pnl.add(exit_btn);
    }

    /**
     * @param status = 1: Call Main Menu <br>
     *               status = 2: Call Topic Selection<br>
     *               status = 3: Call Help and Credits<br>
     *               status = 4: View Highscores<br>
     *               status = 5: Update Database<br>
     *               status = 6: Start Game<br>
     *               status = 7: Game over Screen<br>
     *               status = 0: Exit Game<br>
     *
     *               Important Function as it decides to change to another frame,
     *               provides some security with grantedAccess boolean,
     *               and Also does the mandatory things that need to be done if the
     *               close button is pressed.
     **/
    public static void changeFrame(int status) {
        // Status is 0 when you wanna quit, so we gotta do some stuff before you quit,
        // like creating the backup.
        if (status == 0) {

            // Create a local backup of the users file irrespective of what was done during
            // gameplay.
            DataBaseManager.createLocalDatabaseBackupOfUsers();

            // If the user is a guest, then the index is less than 0, in which case dont
            // update anything.
            if (DataBaseManager.USER_INDEX < 0) {
                System.out.println("You are a guest, so not updating anything. \n");
            } else {
                // If its a user, then update the user score. The index is known already in a
                // static variable.
                DataBaseManager.updateUserScore();
            }

            // This is what keeps track of when the last time was that the database was
            // updated. You dont need to update it every time you run the game.
            String lastUpdateDate = "";

            // Opening the Backup Date file and checking the last time it was backed up.
            File dateFile = new File(DataBaseManager.LOCAL_BACKUP_DATEFILE);
            if (dateFile.exists()) {
                try (BufferedReader br = new BufferedReader(new FileReader(dateFile))) {
                    lastUpdateDate = br.readLine();

                    // If the database was backed up last today, then dont do it.
                    if (lastUpdateDate.equals(String.valueOf(LocalDate.now()))) {
                        System.out.println("Backup DataBases are Up to Date!");
                    } else {
                        // Else update it.
                        DataBaseManager.createLocalDatabaseBackup();
                    }
                } catch (IOException e) {
                    throw new RuntimeException(e);
                } catch (NullPointerException exception) {
                    System.out.println("Nothing in the Date File. ");
                }
            } else {
                // If the file itself doesnt exist, then clearly there doesnt exist any backup,
                // so we better back up at that point.
                try {
                    DataBaseManager.createLocalDatabaseBackup();
                } catch (Exception e) {
                    System.out.println("You havent really created the database yet, so not creating backup either. ");
                }
            }
            // Exit game
            System.out.println("Thanks for Playing! ");
            System.exit(0);
        }
        if (grantAccess) {
            System.out.println("Access Granted!");
            switch (status) {
                case 1 -> {
                    // Showing Main Menu
                    grantAccess = false;
                    menuFrame = new MenuFrame();
                }
                case 2 -> {
                    // Showing the TopicsFrame
                    grantAccess = false;
                    topicsFrame = new TopicsFrame();
                }
                case 3 -> {
                    // Showing the Help Screen
                    grantAccess = false;
                    helpFrame = new HelpFrame();
                }
                case 4 -> {
                    // Showing Highscores
                    grantAccess = false;
                    highscoreFrame = new HighscoreFrame();
                }
                case 5 -> {
                    System.out.println("Updating Database");

                    // Instead of overwriting the files, or appending to them, as they contain old
                    // data,
                    // we will just erase them altogether and create them again.
                    DataBaseManager.clearLocalDatabase();
                    MongoManager.clearMongoDb();

                    // Scrap everything and Start Saving
                    AmazonScrapper obj = new AmazonScrapper();
                    try {
                        AmazonScrapper.scrapAndSave();
                    } catch (Exception e) {
                        System.out.println("Couldnt update the database, there was some problem. It was");
                        System.out.println(e.getMessage());
                    }
                    // Just copy everything to the backup either way.
                    DataBaseManager.createLocalDatabaseBackup();

                    File dateFile;

                    // Updating the Mongo and Local Database File.
                    dateFile = new File(DataBaseManager.LOCAL_DATEFILE);
                    try (FileWriter f = new FileWriter(dateFile, false)) {
                        f.write(String.valueOf(LocalDate.now()));
                    } catch (IOException e) {
                        throw new RuntimeException(e);
                    }
                    dateFile = new File(DataBaseManager.LOCAL_MONGODATEFILE);
                    try (FileWriter f = new FileWriter(dateFile, false)) {
                        f.write(String.valueOf(LocalDate.now()));
                    } catch (IOException e) {
                        throw new RuntimeException(e);
                    }
                }
                case 6 -> {
                    // Showing Game Screen
                    grantAccess = false;
                    gameFrame = new GameFrame();
                }
                case 7 -> {
                    // This is only called by the gameframe, which has a timer, which is what calls
                    // this function, and as its in a different class,
                    // you have to close the things from here coz that timer cant access its parent
                    // class properties.
                    gameFrame.setVisible(false);
                    gameFrame.dispose();

                    // Show GameOverScreen
```

```
                    gameOverFrame = new GameOverFrame();
                }
                default -> {
                    // In Case something goes really wrong, just backup and exit.
                    DataBaseManager.createLocalDatabaseBackup();

                    // Exit game
                    System.out.println("Thanks for Playing! ");
                    System.exit(0);
                }
            }
        } else {
            System.out.println("Access Denied Who are you? What are you trynna do
 here? ");
            System.exit(0);
        }
    }

    /*
     * This function is overridden from the Thread class, coz its empty there, and
     * thread.start calls this function.
     * And this is where you put loops or something in case you wanna do something
     * for ever as a game Loop and access data members stored somewhere else and
     * written to by some other classes.
     * The Job of this function here is important in that its the first function
     * that is real multithread. It checks the database, and if they are not up to
     * date, it updates them.
     */
    public void run() {

        // Just establish the connection, and if thats not possible, then we
 clearly
        // arent gonna be using mongo.
        usingMongo = MongoManager.establishConnectionWithMongo();

        // Same logic as demod in changeFrame()
        String lastUpdateDate = "";

        // Checking the Local CSV Files
        File dateFile = new File(DataBaseManager.LOCAL_DATEFILE);
        if (dateFile.exists()) {
            try (BufferedReader br = new BufferedReader(new FileReader(dateFile)))
  {
                lastUpdateDate = br.readLine();
                System.out.println(lastUpdateDate);
                if (lastUpdateDate.equals(String.valueOf(LocalDate.now()))) {
                    System.out.println("Local DataBases are Up to Date!");
                    isLocalDatabaseUpToDate = true;
                }
            } catch (IOException e) {
                throw new RuntimeException(e);
            } catch (NullPointerException exception) {
                System.out.println("Nothing in the Local Date File. ");
            }
        }

        // Now check the mongodb database date file to check when was the last
 time it
        // was updated. Same Logic tho.
        dateFile = new File(DataBaseManager.LOCAL_MONGODATEFILE);
        if (dateFile.exists()) {
            try (BufferedReader br = new BufferedReader(new FileReader(dateFile)))
 {
                lastUpdateDate = br.readLine();
                System.out.println(lastUpdateDate);
                if (lastUpdateDate.equals(String.valueOf(LocalDate.now()))) {
                    System.out.println("Mongo DataBases are Up to Date!");
                    isMongoUpToDate = true;
                }
            } catch (IOException e) {
                throw new RuntimeException(e);
            } catch (NullPointerException exception) {
                System.out.println("Nothing in the mongo Date File. ");
            }
        }

        // If say one of them is not updated, then we gotta scrap amazon.
        if (!isLocalDatabaseUpToDate || (usingMongo && !isMongoUpToDate)) {

            System.out.println("Beginning to Scrap Data From Amazon, as one of the
 DataBases isnt updated. ");
            if (!isLocalDatabaseUpToDate) {
                // As an edge case, if mongo isnt up to date, we dont wanna clear
the local one.
                DataBaseManager.clearLocalDatabase();
            }
            if (usingMongo && !isMongoUpToDate) {
                // If the local one isnt up to date we dont wanna clear mongo.
                MongoManager.clearMongoDb();
            }

            // Scrap and save, as at this point we already know what works and
what doesnt,
            // and what is updated and what isnt,
            // AmazonScrapper class can figure out where to save stuff. After that
            // everything would have to be updated.
            AmazonScrapper obj = new AmazonScrapper();
            try {
                AmazonScrapper.scrapAndSave();
                isLocalDatabaseUpToDate = true;

                // writing to the date file coz we must have updated at this point
                dateFile = new File(DataBaseManager.LOCAL_DATEFILE);
                try (FileWriter f = new FileWriter(dateFile, false)) {
                    f.write(String.valueOf(LocalDate.now()));
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
                System.out.println("Updated the local database, no need to depend
on the backup anymore");

                if (usingMongo) {
                    // Coz at this point it has to be, as we just scrapped and
didnt get any erros.
                    isMongoUpToDate = true;

                    // writing to the date file coz we must have updated at this
point
```

```
                    dateFile = new File(DataBaseManager.LOCAL_MONGODATEFILE);
                    try (FileWriter f = new FileWriter(dateFile, false)) {
                        f.write(String.valueOf(LocalDate.now()));
                    } catch (IOException e) {
                        throw new RuntimeException(e);
                    }
                    System.out.println("Updated the Mongo database, no need to depend on the local one anymore");
                }
            } catch (Exception e) {
                System.out.print("Couldnt update one of the databases, in the case that one of them wasnt updated. ");
                System.out.println(e.getMessage());
            }

            // This has to happen at this point as a forced minimum.
            isLocalDatabaseUpToDate = true;
        }
    }

    public static void main(String[] args) {

        // This is so that the fonts are rendered correctly in Swing gui.
        System.setProperty("awt.useSystemAAFontSettings", "on");
        System.setProperty("swing.aatext", "true");

        // This is to call the thread, so we can check the databases.
        Main t1 = new Main();
        t1.start();

        // As the thread starts, we start the game. Usually it has to read from the
        // backup file if the database isnt updated yet. After which it would start
        // reading from there. As downloading the images and putting them in the
        // database takes time and we cant wait that long, that job is multithreaded.
        // The use of the backup database is :
        // 1. It has some basic images that are shipped with the jar file so in case
        // someone doesnt have internet, atleast they have something.
        // 2. It is the fallback in case something goes wrong while doing or reading
        // something from one of the files.
        // 3. It serves as the Primary database when we are updating the local database,
        // and we still need to show stuff to the user so they can play the game. This
        // is the most important one.
        loginFrame = new LoginFrame();
    }
}
```

Listing 1: Main Java file