## Table of Contents


**SOFTWARE REPORT** ..... 1

**ASSIGNMENT COVER SHEET FOR ONLINE SUBMISSION** ..... 2

**PLANNING PROCESS** ..... 2
**INTRODUCTION** ..... 2
**MUSIC.JAVA FILE** ..... 2
Class Builder ..... 2
Getters ..... 3
**MAINFRAME** ..... 3
Java Serialization ..... 3
Deserialization ..... 5
Animation and Graphics ..... 5
Timers ..... 7
**SWING BASED GRAPHIC USER INTERFACE (GUI)** ..... 8
Displaying the data ..... 8
Buttons ..... 8
**THREADS** ..... 11
**CONCLUSION** ..... 12

**REFERENCES:** ..... 12

## Planning Process

When I first started this project, I had never coded in Java. Therefore, the first step for me was to learn the basics of the language. I started by doing the Java course on Code Academy. I decided to focus on key concepts that I would use in my project such as: animation, graphics, use of timers, event handling and java serialisation. I did exercises in NetBeans, which further enhanced my understanding

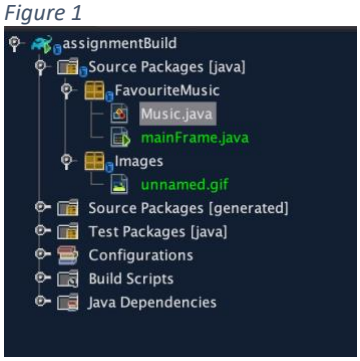
## Introduction

I decided to base my project on my favourite songs. I wanted the program to display my top songs, as well as album is it from, which artist and the year released.

## Music.java file

### Class Builder

*Figure 1*




My Music.java file contains the class builder for my project. This is where I have written code to create an object from my music class.

*Figure 2*

```
import java.io.Serializable;
```

The project brief mentions that I need to serialize my data and deserialize it when the program opens. In-order to do this, I had to import the Serializable interface.

```
public class Music implements Serializable {
    public String Song_Name;
    public String Artist;
    public String Album;
    public int Year_Released;

    Music(String Song_Name,String Artist, String Album, int Year_Released ){
        this.Song_Name = Song_Name;
        this.Artist = Artist;
        this.Album = Album;
        this.Year_Released = Year_Released;
    }
```

*Figure 3*

Next, I created the class for my music and ensured that it is serializable. I declared 3 variables as strings and 1 variable as an integer, this can be seen above.

I created a constructor class; this tells my program how to create a music object. As seen above, to create a music object I have to enter a song name, artist, album and the year released as an integer.

### Getters

```
public String getSongName() {
    return Song_Name;
}

public String getArtist() {
    return Artist;
}

public String getAlbum() {
    return Album;
}
public String getYearReleased() {
    String String_Year_Released = String.valueOf(Year_Released);
    return String_Year_Released;
}

@Override
public String toString() {
    return this.Song_Name + " " + this.Artist + this.Album + this.Year_Released;
}
```

*Figure 4*

Getters have also been coded so that I can return the data to my user interface. Since Java text fields only take in strings, I had to convert the Year Released in my getter function

## MainFrame

### Java Serialization

#### *Method to load objects*

```
private ArrayList<Music> myFavouriteSongs;
private String fileName = "MusicData.ser";
int index = 0;
```

*Figure 5*

```
myFavouriteSongs = new ArrayList();
```

*Figure 6*

The first line of code in Figure 5 means that I create an array list based on my Music class called myFavouriteSongs. The second line of code assigns the value “MusicData.ser” to the

string variable ‘fileName’. I use this for the name of my serialised file. In Figure 6 I have created a new instance of the array list and set it equal to myFavouriteSongs within my mainframe constructor.

```
private void loadDataInnitial() {
    Music m1 = new Music("Do Not Disturb", "Drake", "More Life", 2017);
    myFavouriteSongs.add(m1);
    Music m2 = new Music("Love Yourz", "J Cole", "2014 Forest Hills Drive", 2014);
    myFavouriteSongs.add(m2);
    Music m3 = new Music("Jungle", "A Boogie wit Da Hoodie", "Artist", 2016);
    myFavouriteSongs.add(m3);
    Music m4 = new Music("Every Season", "Roddy Ricch", "Feed Tha Streets 2", 2018);
    myFavouriteSongs.add(m4);
    Music m5 = new Music("Nights", "Frank Ocean", "Blond", 2016);
    myFavouriteSongs.add(m5);
    Music m6 = new Music("wokeuplikethis*", "Playboi Carti & Lil Uzi Vert", "Playboi Carti", 2017);
    myFavouriteSongs.add(m6);
    Music m7 = new Music("Poetic Justice", "Kendrick Lamar", "good kid,m.A.A.d city", 2012);
    myFavouriteSongs.add(m7);
    Music m8 = new Music("Drugs You Should Try it", "Travis Scott", "Days Before Rodeo", 2014);
    myFavouriteSongs.add(m8);
    Music m9 = new Music("Often", "The Weekend", "Beauty Behind The Madness", 2015);
    myFavouriteSongs.add(m9);
    Music m10 = new Music("Picture Me", "Dave", "Six Paths", 2016);
    myFavouriteSongs.add(m10);
    Music m11 = new Music("Signs","1Yush x AbizLoveLiz","Rookie Of The Year",2021);
    myFavouriteSongs.add(m11);
    Music m12 = new Music("PainKiller","AbizLoveLiz","Rookie Of The Year",2021);
    myFavouriteSongs.add(m12);
    Music m13 = new Music("Ready","1Yush","Rookie Of The Year",2021);
    myFavouriteSongs.add(m13);
}
```

*Figure 7*

I wrote a method to load my data which I go on to serialise later. In the method above, I am creating a new instance of the Music class by passing 4 pieces of data; 3 strings and an integer. This is because, as I mentioned above in the music.java file, the music class requires 3 strings and an integer. After creating a new instance, I added it to my array list called ‘myFavouriteSongs’.

### *Serialization*

```
public void serializeMusic() {
    try {
        FileOutputStream fileOut = new FileOutputStream(fileName);
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        out.writeObject(myFavouriteSongs);
        out.close();
        fileOut.close();
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

*Figure 8*

This is my method that I use to serialises my data. I create a file output stream that writes to a file that is specified in the filename variable (‘MusicData.ser’). The data from the array

myFavouriteSongs is written to the file and then ‘out.close’ and ‘fileOut.close’ is used to close and save the file. If there are any errors, then the error message is caught by e.

## Deserialization

```
public ArrayList<Music> doInBackground() {
    final ArrayList<Music> dataLoading = deserializeMusic();
    deserializeMusic();
```

*Figure 9*

I have called the deserialize function in my ‘doInBackgroundThread’. The code in Figure 9 is saying that ‘dataLoading’ is expecting an array list based on the music class which it is going to get from the ‘deserializeMusic’ method.

```
public ArrayList<Music> deserializeMusic() {
    ArrayList<Music> songs = new ArrayList();
    try {
        FileInputStream fileIn = new FileInputStream(fileName);
        ObjectInputStream in = new ObjectInputStream(fileIn);
        songs = (ArrayList<Music>) in.readObject();
        in.close();
        fileIn.close();
    } catch (IOException i) {
        i.printStackTrace();

    } catch (ClassNotFoundException c) {
        System.out.println("Music class not found");
        c.printStackTrace();

    }
    return songs;
}
```

*Figure 10*

Figure 10 shows the ‘deserializeMusic’ method. A new variable called ‘songs’ is initialised that is expecting an array list from the music class. Once this is done, File input stream makes it possible to read the contents of the file as a stream of bytes. The object input stream allows me to read java objects from the input stream. Next im telling java that the stream of bytes is an array list that holds objects from the Music class. Once the file has been deserialised, the variable songs is returned. This is to ensure thread safety for the thread above, as shown in Figure 9.

## Animation and Graphics

```
// animation position related
private BufferedImage ImgCd;
private int startx = 100, starty = 200;
private int xPos = 400;
private int degrees =  0;
```

*Figure 11*

At the start of my program, I have declared 'ImgCd' as a Buffered Image, this is the bitmap image I'm going to be using for parts of my animation. Additionally, I have declared and set x and y positions for where the cd's are going to start on the screen. I have also declared 'degrees' to 0, which I will use to have a rotating effect on the screen.

```
try {
        ImgCd = ImageIO.read(new File("cd.png"));
}       catch (Exception ex) {
        ex.printStackTrace();
}
```

*Figure 12*

'ImgCd' is then populated with the bitmap image of the cd. The code above is checking the location the code for a file called 'cd.png' If it can't find it, then it gives out an error.

```
private AffineTransformOp getRotateOp(int degrees) {
    double radians = Math.toRadians(degrees);
    double xCentre = ImgCd.getWidth() / 2;
    double yCentre = ImgCd.getHeight() / 2;

    AffineTransform tx = AffineTransform.getRotateInstance(radians, xCentre, yCentre);
    AffineTransformOp op = new AffineTransformOp(tx, AffineTransformOp.TYPE_BILINEAR);

    return op;

}
```

*Figure 13*

Next I had to write a method called Affine transformations which rotates the CD. The centre of the image is taken and that is used to rotate it.

```java
public void drawCd(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    AffineTransformOp rotateOp = getRotateOp(degrees += 2);
    g2d.drawImage(ImgCd, rotateOp, startx += 1, starty);
    if (startx > 430) {
        g2d.drawImage(ImgCd, rotateOp, startx -= 2, starty);
    }
    Toolkit.getDefaultToolkit().sync();
}

public void drawCd2(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    AffineTransformOp rotateOp = getRotateOp(degrees += 2);
    g2d.drawImage(ImgCd, rotateOp, xPos -= 1, starty);
    if (xPos < 5) {
        g2d.drawImage(ImgCd, rotateOp, xPos += 2, starty);
    }
    Toolkit.getDefaultToolkit().sync();
}
```

*Figure 14*

These two methods use Java Graphics. First they rotate the image by incrementing the degrees by two. Java graphics is used to draw the cd's onto the screen. One of the cd's x position is incremented by two while the others is decremented, this is how the cd moves across the screen. Furthermore, once the cd's reach the border, the cd's go back.

```java
public void paint(Graphics g) {
    super.paint(g);
    if (IntroPannel.isVisible()) {
        drawCd(g);
        drawCd2(g);
    }
    //paintCircle(g);

}

 /*public void paintCircle(Graphics g) {
    g.drawOval(++startx, ++starty, 20, 20);
    Toolkit.getDefaultToolkit().sync();
}
*/
```

*Figure 15*

The method in figure 15 calls the method in figure 14 to actually draw the cds onto the screen, but only when the intro panel is set to visible, which is my animation panel. 'superpaint' repaints the screen every twenty milliseconds to make it seem like the picture is moving which gives it the animation effect.

## Timers

```java
// animation timer related
private Timer animationTimer;
private int animationTimerInterval = 10; // in milliseconds
```

*Figure 16*

I have made use of Java Timer for my animation. Every 20 milliseconds, I update the animation screen to give the effect of an animation.

## Swing Based Graphic User Interface (GUI)


*Figure 17*

Figure 16 is my user Interface (UI). This is made using Javas Swing components. I have implemented 'jTextFields' to display the songs from the array. Additionally, I have also made use of the 'jLabels' to label what type of data is being displayed such as 'Song Name' etc.

### Displaying the data

```
private void DisplayData(int i) {
    SongNameText.setText("");
    ArtistNameText.setText("");
    AlbumText.setText("");
    YearReleasedText.setText("");
    Music currentMusic = myFavouriteSongs.get(i);
    SongNameText.setText(currentMusic.getSongName());
    ArtistNameText.setText(currentMusic.getArtist());
    AlbumText.setText(currentMusic.getAlbum());
    YearReleasedText.setText(currentMusic.getYearReleased());
    index = i;

}
```

*Figure 18*

The method shown in Figure 17 is responsible for displaying the deserialised data onto the GUI. The jTextFields are first emptied and then filled with the deserialised data using getter methods from my Music Class, which I explained earlier in the report.

### Buttons

Figure 16 shows my GUI, the buttons are either neon green to navigate through the songs or red to exit the program.

#### *First Song Button*

```
private void FirstMusicActionPerformed(java.awt.event.ActionEvent evt) {
    DisplayData(0);
}
```

*Figure 19*

When the First song button is pressed, the first item on the array list is displayed using the above line of code.

*Previous Button*

```
private void PrevMusicActionPerformed(java.awt.event.ActionEvent evt) {
    int min = 0;
    int max = myFavouriteSongs.size() - 1;
    int updatedIndex = index-1;
    if (updatedIndex < min) {
            updatedIndex = max;
    }
    DisplayData(updatedIndex);
}
```

*Figure 20*

The code above shows how the previous button works. The program checks to see, when the button is pressed, is the updated array pointer less than 0. If it is, then the array pointer is set to the last item in the array. If it isn't then the updated index is just decremented. With this, the array pointer loops through the array so the data is always circled around.

*Random Button*

```
private void RandomMusicActionPerformed(java.awt.event.ActionEvent evt) {
    Random rand = new Random();
    int randomIndex = rand.nextInt(myFavouriteSongs.size());
    DisplayData(randomIndex);

}
```

*Figure 21*

Rand is an instance of the random class. A random number between 0 and the size of the array is generated and put into the variable randomIndex, which then displays the data associated with that number. Therefore, when the random button is pressed, a random song in the array will be displayed.

*Next Button*

```
private void NextMusicActionPerformed(java.awt.event.ActionEvent evt) {
    int max = myFavouriteSongs.size() - 1;
    int updatedIndex = index + 1;
    if (updatedIndex > max) {
        updatedIndex = 0;
    }
    DisplayData(updatedIndex);
}
```

*Figure 22*

The next button is similar to the previous button. It also uses a selection statement to ensure that the array is going in circles. When the next button is pressed, it usually displays the next item in the array by updating the index. However, if the current index is on the last item in the array then the updated index is set to the first item in the array then displayed.

*Last Button*

```
private void LastMusicActionPerformed(java.awt.event.ActionEvent evt) {
    int max = myFavouriteSongs.size() - 1;
    DisplayData(max);

}
```

*Figure 23*

The last button just displays the last item in the array. Above you can see that the variable max is set to the size of the array – 1, this is because the first item is saved at 0 and not 1. So 1 needs to be decremented from myFavouriteSongs.size().

*Exit Button*

```
private void ExitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    loadDataInnitial();
    serializeMusic();
    System.exit(0);
}
```

*Figure 24*

When the exit button is pressed, the data is serialised again just like the project brief asks me to do and then the window is closed.

*Escape Button*

```
private void formKeyPressed(java.awt.event.KeyEvent evt) {
    if (evt.getKeyCode() == KeyEvent.VK_ESCAPE) {

        animationTimer.stop();
        loadDataInnitial();
        serializeMusic();
        JOptionPane.showMessageDialog(this, "You have chosen to exit the program!");
        System.exit(0);
    }
}
```

*Figure 25*

If the escape key on the keyboard is pressed at any point in my program, the animation timer is stopped, the data is serialised and a small box pops up saying that the window is closing, before actually closing it.

## Threads

```
private void createAndRunLoadingThread() {
    IntroPannel.setVisible(true);
    MainPannel.setVisible(false);
    animationTimer = new javax.swing.Timer(animationTimerInterval, this);
    animationTimer.start();

    worker = new SwingWorker<ArrayList<Music>, Void>() {
        @Override
        public ArrayList<Music> doInBackground() {
            final ArrayList<Music> dataLoading = deserializeMusic();
            deserializeMusic();
            try {
                for (int c = 0; c <= 100;c++) {
                    Thread.sleep(100);
                    setProgress(c);


                }
            } catch (InterruptedException e) {
                System.out.println("Interrupted exception");
            }
            return dataLoading;
        }

        @Override
        public void done() {

            try {
                // code separated out to make things easier
                doThisWhenFinished();
             } catch (InterruptedException ignore) {
             } catch (java.util.concurrent.ExecutionException e) {
                 System.out.println("System error: concurrent exception: " + e.getMessage());
             }
        }

        private void doThisWhenFinished() throws InterruptedException, java.util.concurrent.ExecutionException {
            System.out.println("Task completed");
            IntroPannel.setVisible(false);
            animationTimer.stop();
            MainPannel.setVisible(true);
            myFavouriteSongs = get();
            DisplayData(0);
        }

    };

    worker.addPropertyChangeListener((PropertyChangeEvent evt) -> {
        if ("progress".equals(evt.getPropertyName())) {
            jProgressBar1.setValue((Integer)evt.getNewValue());
            Progress_Label.setText(""+evt.getNewValue()+"%");
        }
    });

    worker.execute();
}
```

*Figure 26*

Mainframe is one thread; it handles the animation in my program. The pictures above are another thread in my program which handles loading the data. At the start of the thread, the visibility of the panels is set, and the animation timer is started.
A new swing worker is initialised in this thread, I have defined it and an array list is callable. The background thread deserialises the data then sleeps for 101 times for 100 milliseconds each, in this time, the animation is drawn and run. I have had to create a delay in my code as the computer deserialises my file too quick for my animation to run. I have also used the thread.sleep to increment my progress bar, which is thread safe as I am using swing worker to listen to changes rather than allowing both mainframe and this thread to access and alter the variable at the same time. After the animation is done and the data is loaded, the animation panel is set to invisible, and my main panel is set to visible, and the data is displayed. The last line “worker.execute” starts this whole thread.

## Conclusion

I am really pleased with the program I have coded and learned quite a lot while doing so. If I had more time and more knowledge, I would have liked to make my program more complicated by using APIs from streaming platforms such as Spotify to load data directly from their servers rather than having to code it in. Additionally, I would have liked to implement a mp3 file onto the array and allow the user to play the music that is being shown on the screen. However, I am not sure how this would work as I don't know what would happen if a song were playing and the next button is pressed. Would the song continue playing or stop?

# References:

Jenkov, J., 2021. [online] Tutorials.jenkov.com. Available at: <http://tutorials.jenkov.com/java-io/fileinputstream.html> [Accessed 7 June 2021].