Name _____ Period _____

Playlist

Your Tasks (Mark these off as you go)

- ☐ Define key vocabulary
- ☐ Review the *Song* class
- ☐ Instantiate the *Song* class
- ☐ Access methods in the *Song* class
- ☐ Create a Song object from a text file
- ☐ Receive credit for this lab guide

☐ Define key vocabulary

Class

Constructor

Parameter

Argument

Instance

Object

Method

☐ Review the Song class

In a previous lesson we were introduced to classes in java. Classes allow us to model other data types that are not built into the Java programming language. Below is a portion of a class intended to model a Song,

Song.java

```
public class Song {

    private Clip clip;
    private String title;
    private String artist;
    private int playTime; // in seconds
    private String songPath;
    private File songFile;
    private int playCount;
    /**
     * Constructor: Builds a song using the given parameters.
     * @param tempTitle song title
     * @param tempArtist song artist
     * @param tempPlayTime song length in seconds
     * @param tempSongPath path to song file
     */
    public Song(String tempTitle, String tempArtist,
            int tempPlayTime, String tempSongPath) {

        title = tempTitle;
        artist = tempArtist;
        playTime = tempPlayTime;

        songPath = tempSongPath;
        songFile = new File(songPath);
        playCount = 0;

        //Creates a playable song clip from the the songFile
        try {
            AudioInputStream audioStream =
AudioSystem.getAudioInputStream(songFile);
            AudioFormat format = audioStream.getFormat();
            DataLine.Info info = new DataLine.Info(Clip.class, format);

            try {
                clip = (Clip) AudioSystem.getLine(info);
                clip.open(audioStream);
            } catch (LineUnavailableException ex) {}

        } catch (UnsupportedAudioFileException | IOException ex) {
            Logger.getLogger(Song.class.getName()).log(Level.SEVERE, null, ex);
        }

    }
}
```

Instance variables are variables that can be accessed by any member of the class. For this reason, they are declared at the top of the class. The keyword private means they are only accessible by the class in which they are declared.

The *constructor* takes the same name as the class. It allows us to create Song datatypes. *tempTitle, tempArtist*, etc. are placeholders (or parameters) for values we can pass to the constructor. They allow us to create different types of Songs.

To properly describe the song above we need several *instance* variables including: *title*, *artist*, *playTime*, *songPath*, etc. *Instance* variables are variables that can be accessed by any member of the class. For this reason, they are declared at the top of the class.

All classes require a constructor. If you do not define one, Java will create one for you. Notice in the example above, the constructor takes the same name as the class.

A constructor enables us to create objects from the class. When you *instantiate* a class you are creating an *object* (or data type) of that class

Refer to the code snippet below,

```
public class Element{

    private int atomicNumber;
    private double atomicMass;
    private String symbol;
    private boolean isMetal;

    public Element(int atomicNumber, double atomicMass, String symbol, boolean isMe
tal){
        this.atomicNumber = atomicNumber;
        this.atomicMass = atomicMass;
        this.symbol = symbol;
        this.isMetal = isMetal;
    }

}
```

Circle and label the following parts,

(a) Instance variables
(b) Constructor
(c) Constructor parameters

What is this class modeling?

What additional instance variables that could be included to model our datatype with more detail?

☐ Instantiate the Song Class

Instantiating is a fancy word for *creating*. If you *instantiate* a class you are creating an *object* (or data type) of that class. To better understand this, consider the two files below,

Song.java

```
public class Song {

    private Clip clip;
    private String title;
    private String artist;
    private int playTime; // in seconds
    private String songPath;
    private File songFile;
    private int playCount;
    /**
     * Constructor: Builds a song using the given parameters.
     * @param tempTitle song title
     * @param tempArtist song artist
     * @param tempPlayTime song length in seconds
     * @param tempSongPath path to song file
     */
    public Song(String tempTitle, String tempArtist,
            int tempPlayTime, String tempSongPath) {

        title = tempTitle;
        artist = tempArtist;
        playTime = tempPlayTime;

        songPath = tempSongPath;
        songFile = new File(songPath);
        playCount = 0;

        //additional code not shown


    }
}
```

Playlist.java

```
public class Playlist {
    public static void main(String args[]){

      Song mySong = new Song("Cruel Summer", "Taylor Swift", 179,
"Sounds/CruelSummer.wav");
    }
}
```

The keyword *Song* means we are referencing the *Song* class. The name of the *Song* datatype we want to create is called *mySong*.

The *new* keyword means we are instantiating the *Song* class – or creating a *Song* datatype.

These are *arguments* - values that map to the parameters in the Song constructor.

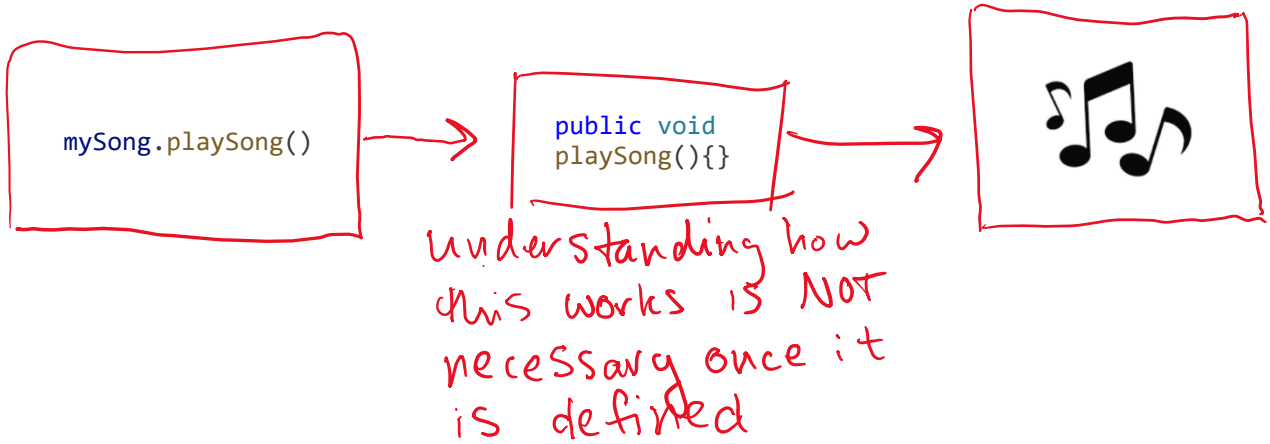
Refer to the Song and Playlist classes above. Write code that could be used to instantiate two new Songs in Playlist.

Each of the following code snippets are intended to instantiate the Song class. Identify the errors.
`Song badSong = new Song("Cruel Summer", "Taylor Swift", "2:58", "sounds/CruelSummer.wav");`
`Song badSong = new Song("Cruel Summer", "Taylor Swift", 158.5 , "sounds/CruelSummer.wav");`
`Song badSong = new Song("Cruel Summer", "Taylor Swift", 158.5);`
`Song badSong = Song("Cruel Summer", "Taylor Swift", 158 , "sounds/CruelSummer.wav");`

☐ Access methods in the Song class

In programming, we often use code to perform a specific task multiple times. Instead of rewriting the same code, we can group a block of code together and associate it with one task, then we can reuse that block of code whenever we need to perform the task again. We achieve this by creating a ***function***. A function is a reusable block of code that groups together a sequence of statements to perform a specific task.

It is possible to implement a function without understanding the underlying complexity. For example, once the *playSong* function is defined, it should be possible to play many songs without understanding how it takes place. This process is illustrated below.



When functions are defined in a class they are referred to as *methods*. Consider a portion of the Song class below,

Song.java

```
public class Song {

    private Clip clip;
    private String title;
    private String artist;
    private int playTime; // in seconds
    private String songPath;
    private File songFile;
    private int playCount;

    public Song(String tempTitle, String tempArtist,
            int tempPlayTime, String tempSongPath) {

        title = tempTitle;
        artist = tempArtist;
        playTime = tempPlayTime;

        songPath = tempSongPath;
        songFile = new File(songPath);
        playCount = 0;

        //additional code not shown

    }
}

/**
 * Plays this song
 */
public void playSong(){

   //implementation not shown

}
```

playSong is function. Also referred to as a method because it is part of the Song class. The keyword public means that it can be accessed from other

To access methods in the Song class we can use the "dot" notation. Below illustrates this process.

Playlist.java

```
public class Playlist {
    public static void main(String args[]){

      Song mySong = new Song("Cruel Summer", "Taylor Swift", 179,
"Sounds/CruelSummer.wav");

      mySong.playSong();

    }
}
```

Refer to the song class below.

```
public class Song {

    public Song(String tempTitle, String tempArtist,
            int tempPlayTime, String tempSongPath) {

        title = tempTitle;
        artist = tempArtist;
        playTime = tempPlayTime;

        songPath = tempSongPath;
        songFile = new File(songPath);
        playCount = 0;

        //additional code not shown
    }

    /**
    * Returns the title of the song
    * @return the title
    */
    public String getTitle(){
         return title;
    }

    /**
    *  Returns the artist of the song
    * @return the artist
    */
    public String getArtist(){
         return artist;
    }

    /**
    * Returns the play time of the song
    * @return the playTime
    */
    public int getPlayTime(){
         return playTime;
    }


    /**
    * Returns the file path of the song
    * @return the songPath
    */
    public String getSongPath(){
         return songPath;
    }

    /**
    * Returns the number of times this song has been played.
    * @return the count
    */
    public int getPlayCount(){
         return playCount;
    }
```

```
    /**
    * Plays this song
    */
    public void playSong(){
       //implementation not shown
    }

    /**
    * Stops this song from playing.
    */
    public void stop(){
       //additional code not shown
    }

    /* (non-Javadoc)
    * returns information about the song
    */
    @Override
    public String toString(){
         return String.format("%-20s %-20s %-25s %10d",
            title, artist, songPath, playTime);
    }
}
```

In the Playlist class below, write code that could be used to create two different songs.

```
public class PlayList {




}
```

For each song, write code that could be used to play the songs.

For each song, write code that could be used to print information about the song.

☐ Create a Song object from a text file

In the code portion of this lab you will be getting your playlist data from a text file. This can be done with a Scanner as follows,

Playlist.java

```
public class PlayList {
    public static void main(String args[]){
        File file = new File(args[0]);//get the file of songs from the user
        //The try-catch statement is required to load files
        try {
            Scanner fileScan = new Scanner(file);
        } catch (FileNotFoundException ex) {
            System.out.println("File cannot be located.");
        }
}
```

Once the file is loaded in our scanner we can access the contents just as we have before. Below illustrates how we can access our Playlist data from our file.

Input.txt

```
Last Tango in Paris
Gotan Project
05:50
sounds/newAgeRhythm.wav
Where You End
Moby
3:18
sounds/classical.wav
Big Jet Plane
Julia Stone
3:54
sounds/westernBeat.wav
```

Playlist.java

```
public class PlayList {
    public static void main(String args[]){
        File file = new File(args[0]);//get the file of songs from the user
        //The try-catch statement is required to load files
        try {
            Scanner fileScan = new Scanner(file);
            Scanner fileScan = new Scanner(file);
            String title = fileScan.nextLine();
            String artist = fileScan.nextLine();
            String playtime = fileScan.nextLine();
            String path = fileScan.nextLine();
            System.out.println(artist + "\n"
                               + title + "\n"
                               + playtime + "\n"
                               + path + "\n");
        } catch (FileNotFoundException ex) {
            System.out.println("File cannot be located.");
        }
}
```

Output

```
Last Tango in Paris
Gotan Project
05:50
sounds/newAgeRhythm.wav
```

Refer to the example above which illustrates how to get playlist data from a text file. To create a Song datatype requires that the playtime be formatted as an int datatype. In the space below write code that could be used to convert the time which is in mm:ss format to an int which represents the playtime of the song in seconds.
Now that the time is formatted correctly, write code that could be used to create a Song object.

☐ Receive credit for this lab guide

Submit this portion of the lab to Pluska to receive credit for the lab guide. Once received, your completed code challenges will also be graded and will count towards your final lab grade.