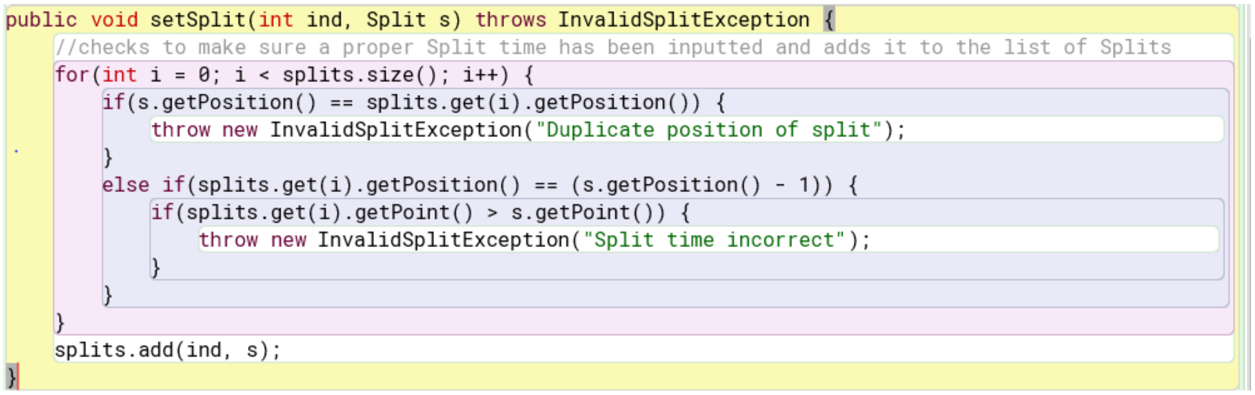
In developing the solution, I used the BlueJ IDE for development.

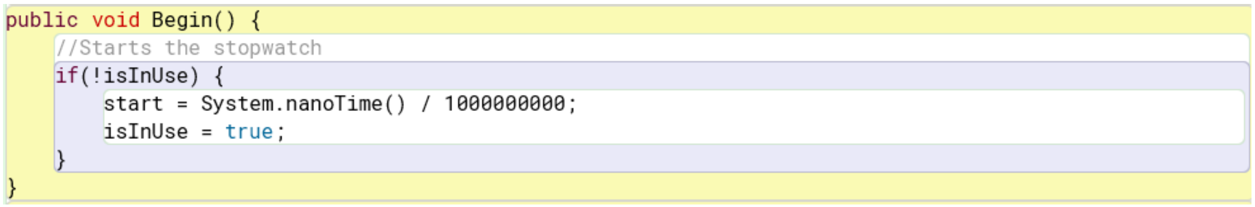
List of techniques

Use of conditional statements like if statements



```java
public void setSplit(int ind, Split s) throws InvalidSplitException {
    //checks to make sure a proper Split time has been inputted and adds it to the list of Splits
    for(int i = 0; i < splits.size(); i++) {
        if(s.getPosition() == splits.get(i).getPosition()) {
            throw new InvalidSplitException("Duplicate position of split");
        }
        else if(splits.get(i).getPosition() == (s.getPosition() - 1)) {
            if(splits.get(i).getPoint() > s.getPoint()) {
                throw new InvalidSplitException("Split time incorrect");
            }
        }
    }
    splits.add(ind, s);
}
```

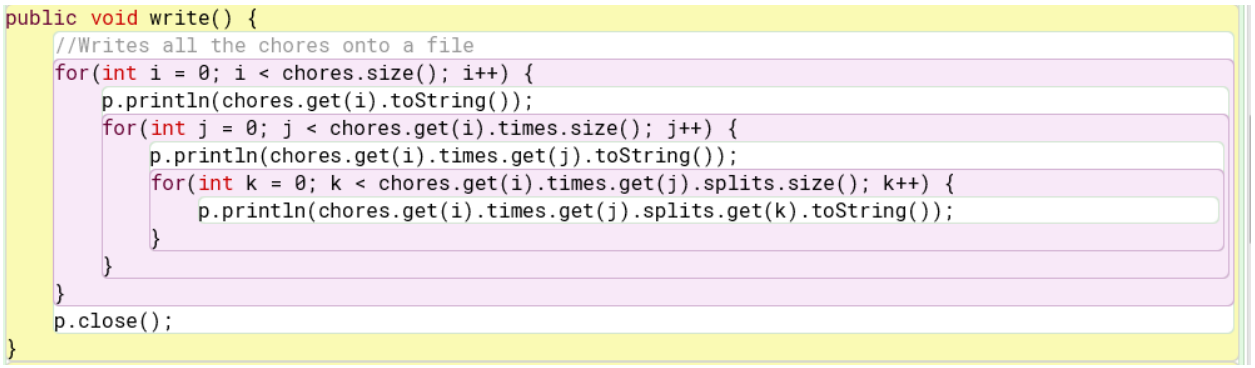
- Shows the use of conditional statements statements because in the method the if statements are used to compare certain parts of Split objects like their positions and points
- The use of these statements are necessary because they allow for comparison between different attributes of objects which lets the program differentiate between values like in this case position and point so that the program can handle the different outcomes properly



```java
public void Begin() {
    //Starts the stopwatch
    if(!isInUse) {
        start = System.nanoTime() / 1000000000;
        isInUse = true;
    }
}
```

- Use of conditional statements can also be seen here in the Stopwatch class. In this case the conditional statement is checking whether the stopwatch is in use or not

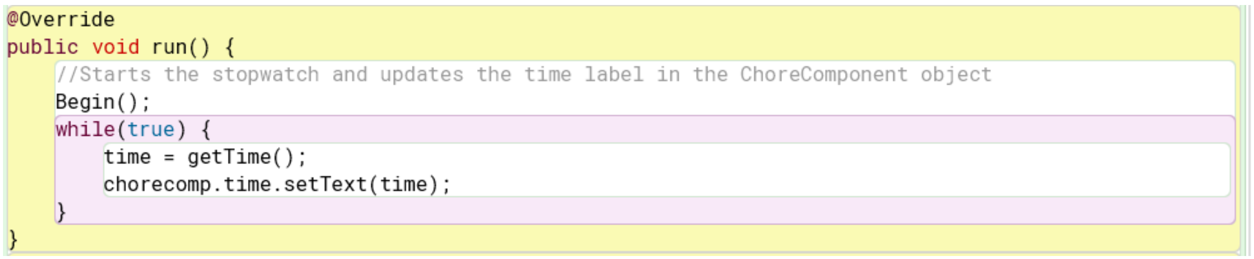
For and while loops including nested loops



```java
public void write() {
    //Writes all the chores onto a file
    for(int i = 0; i < chores.size(); i++) {
        p.println(chores.get(i).toString());
        for(int j = 0; j < chores.get(i).times.size(); j++) {
            p.println(chores.get(i).times.get(j).toString());
            for(int k = 0; k < chores.get(i).times.get(j).splits.size(); k++) {
                p.println(chores.get(i).times.get(j).splits.get(k).toString());
            }
        }
    }
    p.close();
}
```

- Shows the use of for loops and nested for loops because in this case the method is looping through an ArrayList of Chores and at each Chore it loops through the Times of each Chore and at each Time it is looping through the Splits of each chore and writing to the file the ToString methods of all the objects that were looped through

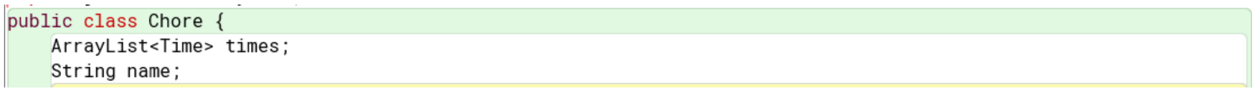
- The use of for loops is necessary because it allows for all the objects of ArrayLists and other data structures to be accessed and processed



```java
@Override
public void run() {
    //Starts the stopwatch and updates the time label in the ChoreComponent object
    Begin();
    while(true) {
        time = getTime();
        chorecomp.time.setText(time);
    }
}
```

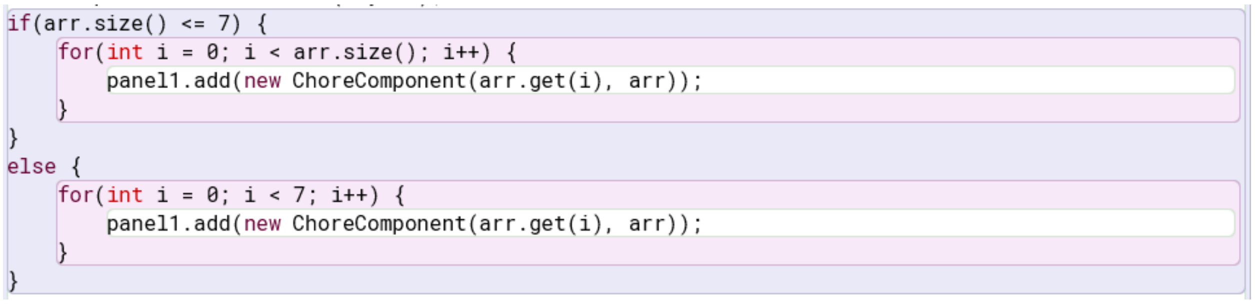
- This method shows the use of a while loop and in this case it is looping through continuously although the while loop can be stopped by pressing the Start/Stop Stopwatch button but the code for that is not shown here
- The use for while loops is necessary because it allows for certain actions to be repeated without having a limit on how many times that action could be repeated while working with varying limits and in this case the limit is how long the user wants the stopwatch to be activated

Use of ArrayLists



```java
public class Chore {
    ArrayList<Time> times;
    String name;
```

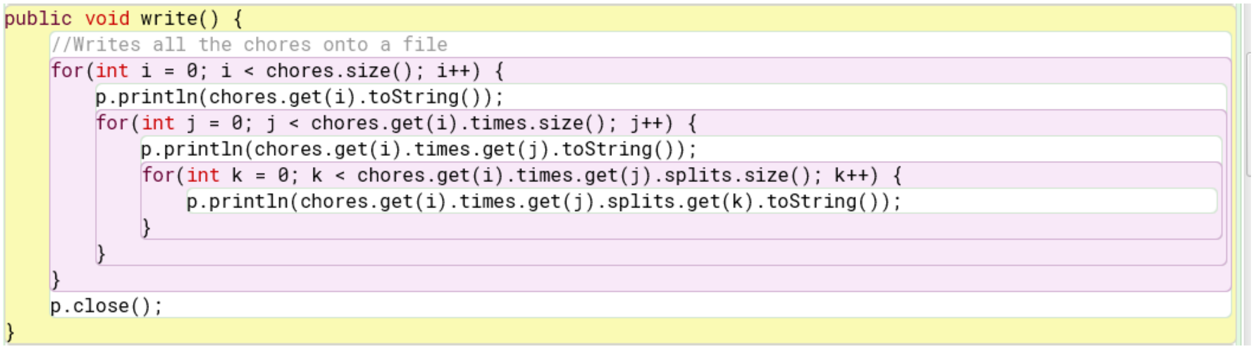
- This screenshot shows the use of ArrayLists because an ArrayList is declared as a field and is used to store Time objects
- I used ArrayLists because an ArrayList is good for iterating through, adding, and removing objects, which I used extensively in the program. Having a changeable size of the ArrayList was also important because there is not a fixed amount of objects like Time and Split that are stored in an ArrayList



```java
if(arr.size() <= 7) {
    for(int i = 0; i < arr.size(); i++) {
        panel1.add(new ChoreComponent(arr.get(i), arr));
    }
}
else {
    for(int i = 0; i < 7; i++) {
        panel1.add(new ChoreComponent(arr.get(i), arr));
    }
}
```

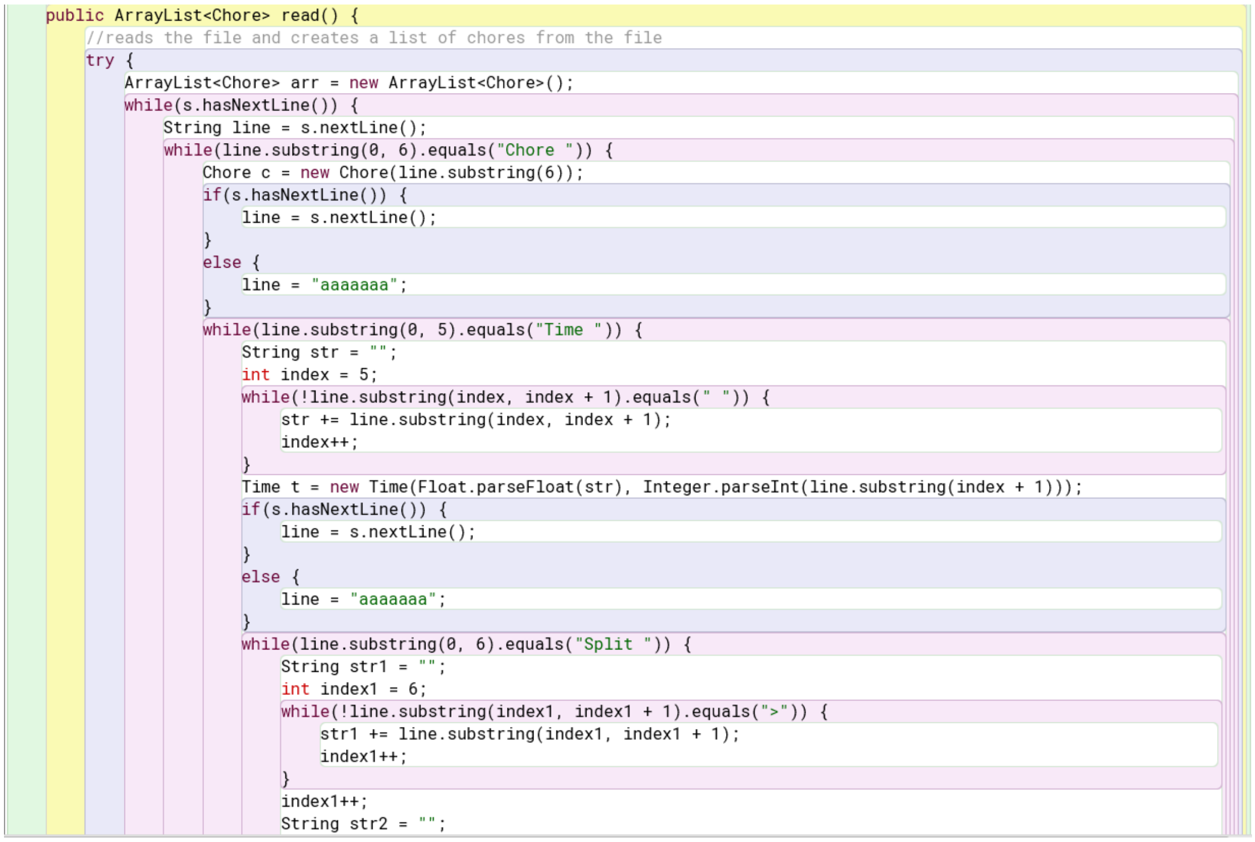
- Another example of ArrayLists being used in the program, in this case the ArrayList arr in the GUI class is being looped through to create new ChoreComponents with the Chores retrieved from the ArrayList

Reading and Writing to a file

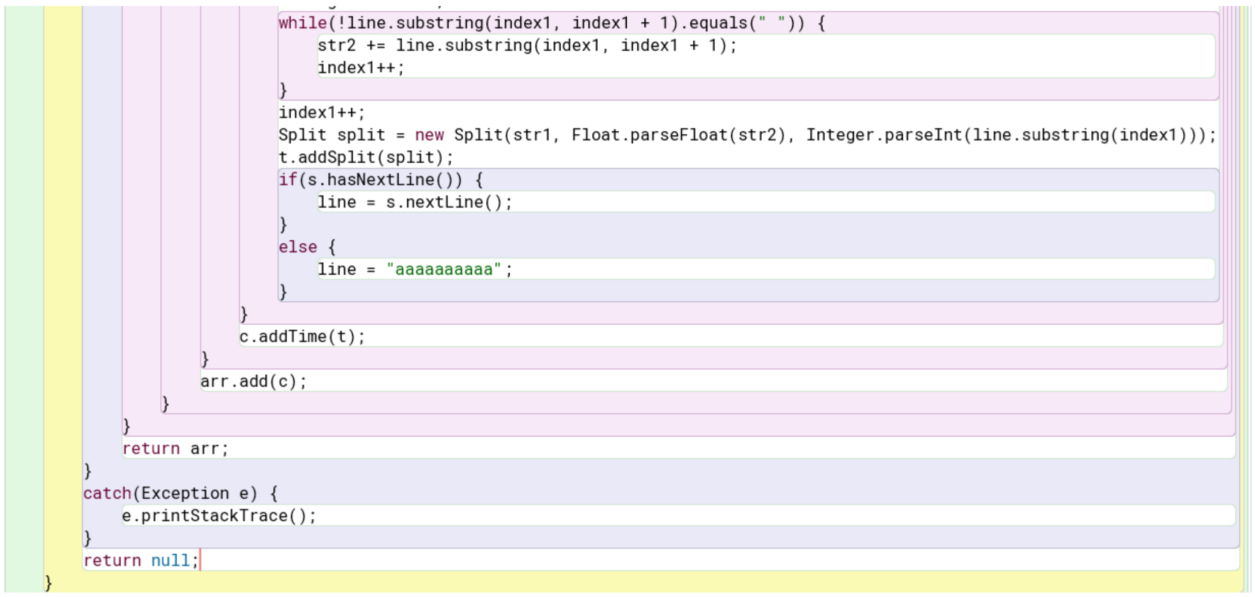


```java
public void write() {
    //Writes all the chores onto a file
    for(int i = 0; i < chores.size(); i++) {
        p.println(chores.get(i).toString());
        for(int j = 0; j < chores.get(i).times.size(); j++) {
            p.println(chores.get(i).times.get(j).toString());
            for(int k = 0; k < chores.get(i).times.get(j).splits.size(); k++) {
                p.println(chores.get(i).times.get(j).splits.get(k).toString());
            }
        }
    }
    p.close();
}
```

- This method is writing to a file by using a PrintWriter object by looping through an ArrayList of Chores and for each Chore loops through an ArrayList of Times and for each Time loops through an ArrayList of Splits and prints every object's toString method onto the file
- It is useful to read and write to files because the file allows for long term storage of Chores and their components

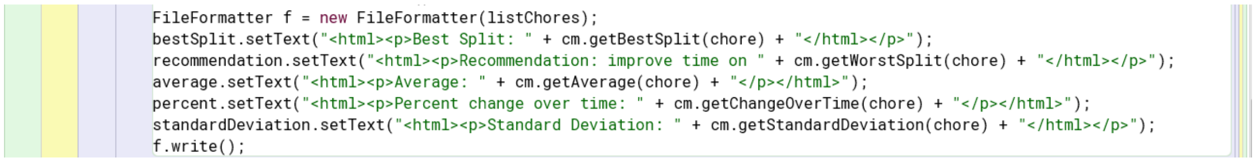


```java
public ArrayList<Chore> read() {
    //reads the file and creates a list of chores from the file
    try {
        ArrayList<Chore> arr = new ArrayList<Chore>();
        while(s.hasNextLine()) {
            String line = s.nextLine();
            while(line.substring(0, 6).equals("Chore ")) {
                Chore c = new Chore(line.substring(6));
                if(s.hasNextLine()) {
                    line = s.nextLine();
                }
                else {
                    line = "aaaaaaa";
                }
                while(line.substring(0, 5).equals("Time ")) {
                    String str = "";
                    int index = 5;
                    while(!line.substring(index, index + 1).equals(" ")) {
                        str += line.substring(index, index + 1);
                        index++;
                    }
                    Time t = new Time(Float.parseFloat(str), Integer.parseInt(line.substring(index + 1)));
                    if(s.hasNextLine()) {
                        line = s.nextLine();
                    }
                    else {
                        line = "aaaaaaa";
                    }
                    while(line.substring(0, 6).equals("Split ")) {
                        String str1 = "";
                        int index1 = 6;
                        while(!line.substring(index1, index1 + 1).equals(">")) {
                            str1 += line.substring(index1, index1 + 1);
                            index1++;
                        }
                        index1++;
                        String str2 = "";
```



```
                        while(!line.substring(index1, index1 + 1).equals(" ")) {
                            str2 += line.substring(index1, index1 + 1);
                            index1++;
                        }
                        index1++;
                        Split split = new Split(str1, Float.parseFloat(str2), Integer.parseInt(line.substring(index1)));
                        t.addSplit(split);
                        if(s.hasNextLine()) {
                            line = s.nextLine();
                        }
                        else {
                            line = "aaaaaaaaaa";
                        }
                    }
                    c.addTime(t);
                }
                arr.add(c);
            }
        }
        return arr;
    }
    catch(Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

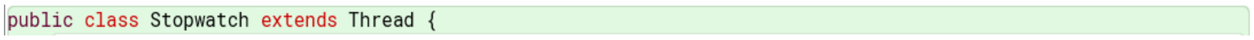
- This method reads from a file and will be about talked more in depth in the algorithms section



```
FileFormatter f = new FileFormatter(listChores);
bestSplit.setText("<html><p>Best Split: " + cm.getBestSplit(chore) + "</html></p>");
recommendation.setText("<html><p>Recommendation: improve time on " + cm.getWorstSplit(chore) + "</html></p>");
average.setText("<html><p>Average: " + cm.getAverage(chore) + "</p></html>");
percent.setText("<html><p>Percent change over time: " + cm.getChangeOverTime(chore) + "</p></html>");
standardDeviation.setText("<html><p>Standard Deviation: " + cm.getStandardDeviation(chore) + "</html></p>");
f.write();
```

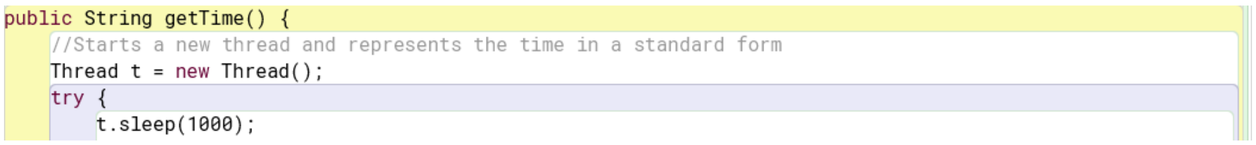
- Instance of the write method of being called

Multithreading



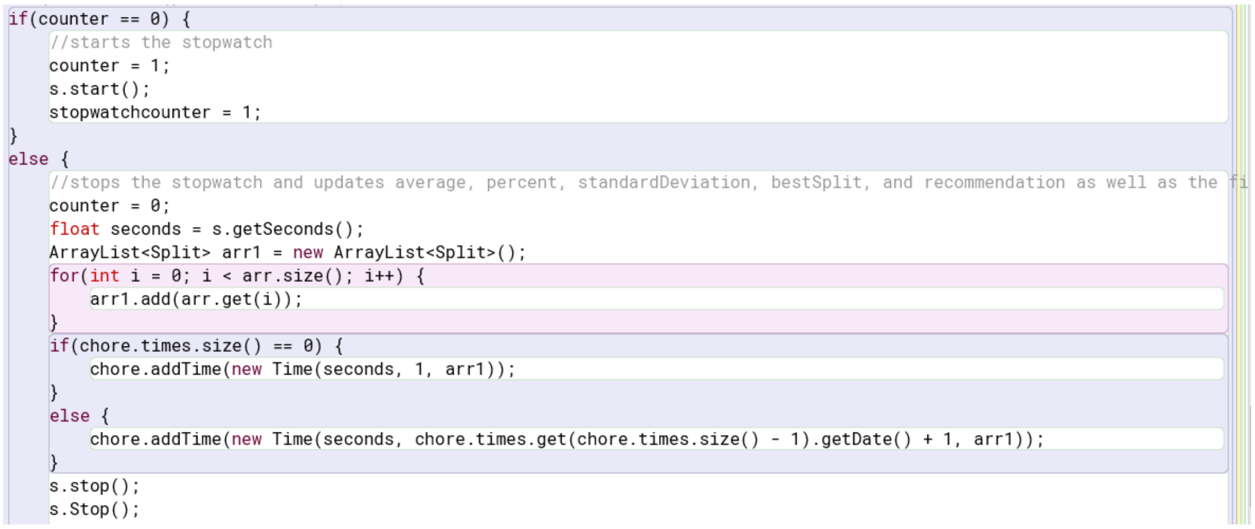
```
public class Stopwatch extends Thread {
```

- By Stopwatch extending Thread, multithreading is used because the Stopwatch can be run on the program while other actions/interactions could be done simultaneously
- Multithreading is important because it allows the user to interact with other components while the Stopwatch is running



```
public String getTime() {
    //Starts a new thread and represents the time in a standard form
    Thread t = new Thread();
    try {
        t.sleep(1000);
```

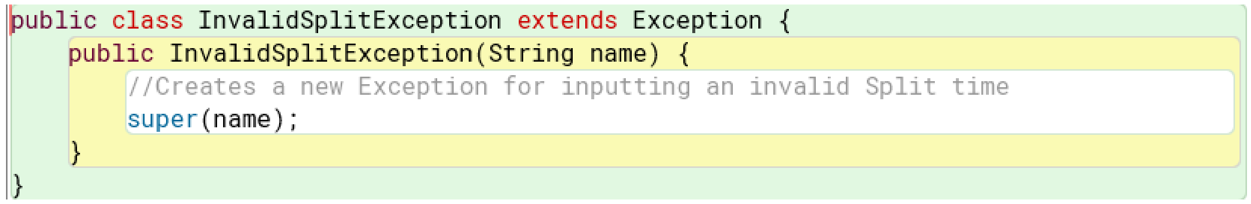
- This screenshot shows another example of multithreading as in this case the thread is used to delay part of the method for 1 second



```java
if(counter == 0) {
    //starts the stopwatch
    counter = 1;
    s.start();
    stopwatchcounter = 1;
}
else {
    //stops the stopwatch and updates average, percent, standardDeviation, bestSplit, and recommendation as well as the fi
    counter = 0;
    float seconds = s.getSeconds();
    ArrayList<Split> arr1 = new ArrayList<Split>();
    for(int i = 0; i < arr.size(); i++) {
        arr1.add(arr.get(i));
    }
    if(chore.times.size() == 0) {
        chore.addTime(new Time(seconds, 1, arr1));
    }
    else {
        chore.addTime(new Time(seconds, chore.times.get(chore.times.size() - 1).getDate() + 1, arr1));
    }
    s.stop();
    s.Stop();
```

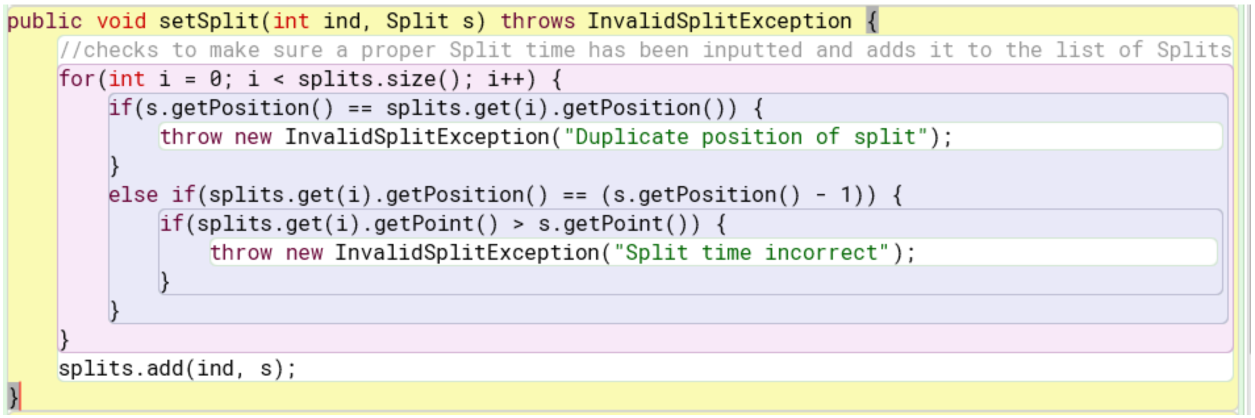
- Example of the Stopwatch being used while the program is running. The use of multithreading here is important because it allows the Stopwatch to run while being able to click the Start/Stop Stopwatch button

Error handling



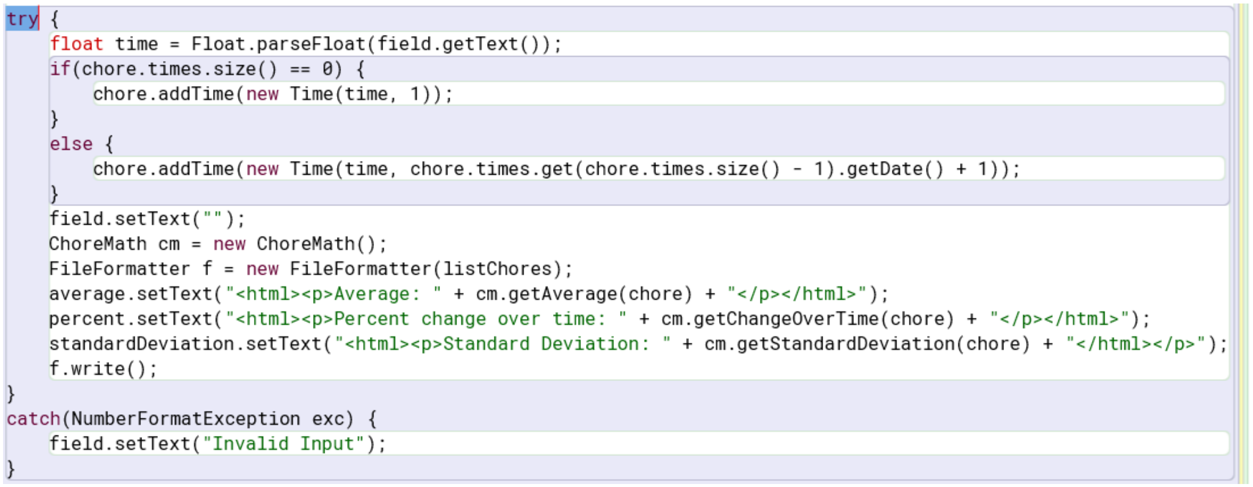
```java
public class InvalidSplitException extends Exception {
    public InvalidSplitException(String name) {
        //Creates a new Exception for inputting an invalid Split time
        super(name);
    }
}
```

- Example of a class creating a custom error
- Error handling is important because it allows the user to see what errors have occurred and it allows the programmer to decide what happens when an error occurs



```java
public void setSplit(int ind, Split s) throws InvalidSplitException {
    //checks to make sure a proper Split time has been inputted and adds it to the list of Splits
    for(int i = 0; i < splits.size(); i++) {
        if(s.getPosition() == splits.get(i).getPosition()) {
            throw new InvalidSplitException("Duplicate position of split");
        }
        else if(splits.get(i).getPosition() == (s.getPosition() - 1)) {
            if(splits.get(i).getPoint() > s.getPoint()) {
                throw new InvalidSplitException("Split time incorrect");
            }
        }
    }
    splits.add(ind, s);
}
```

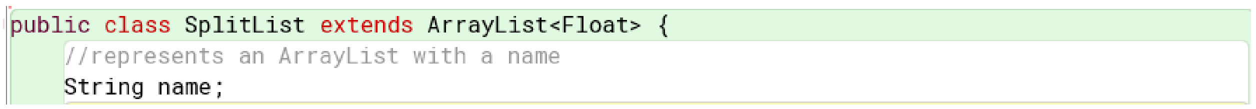
- Example of deciding what the name of the error is going to be in a method to differentiate what went wrong in inputting a Split



```
try {
    float time = Float.parseFloat(field.getText());
    if(chore.times.size() == 0) {
        chore.addTime(new Time(time, 1));
    }
    else {
        chore.addTime(new Time(time, chore.times.get(chore.times.size() - 1).getDate() + 1));
    }
    field.setText("");
    ChoreMath cm = new ChoreMath();
    FileFormatter f = new FileFormatter(listChores);
    average.setText("<html><p>Average: " + cm.getAverage(chore) + "</p></html>");
    percent.setText("<html><p>Percent change over time: " + cm.getChangeOverTime(chore) + "</p></html>");
    standardDeviation.setText("<html><p>Standard Deviation: " + cm.getStandardDeviation(chore) + "</html></p>");
    f.write();
}
catch(NumberFormatException exc) {
    field.setText("Invalid Input");
}
```

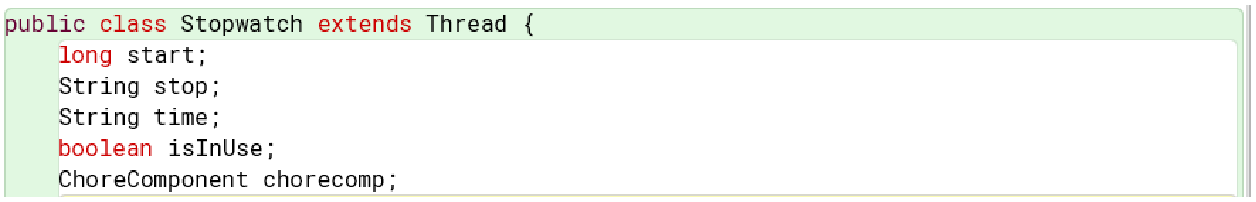
- Example of error handling because if there is a NumberFormatException the program will make set the text of JTextField to "Invalid Input"

Inheritance



```
public class SplitList extends ArrayList<Float> {
    //represents an ArrayList with a name
    String name;
```

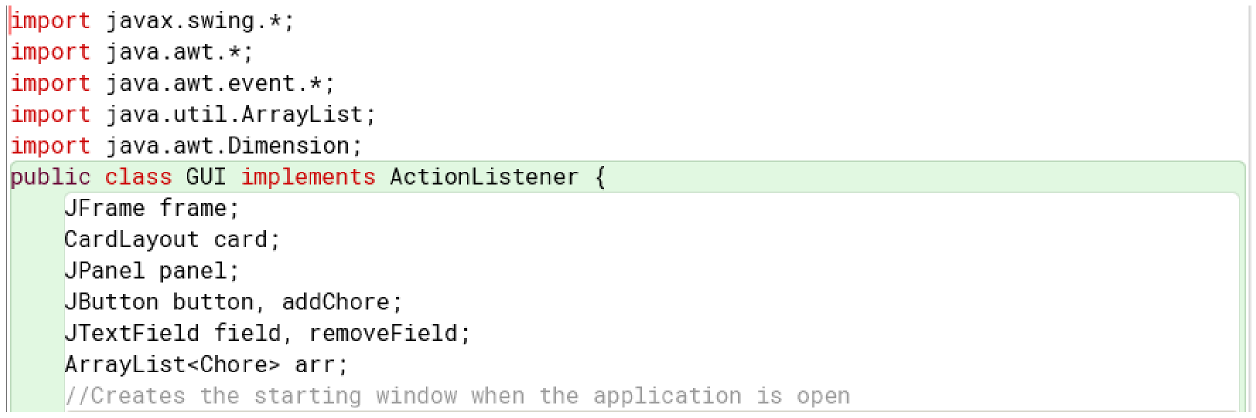
- Example of inheritance and in this case SplitList inherits ArrayList<Float>
- Inheritance is useful because it eliminates redundant code and tells the reader that this class is an ArrayList and has the functionality and methods of an ArrayList



```
public class Stopwatch extends Thread {
    long start;
    String stop;
    String time;
    boolean isInUse;
    ChoreComponent chorecomp;
```

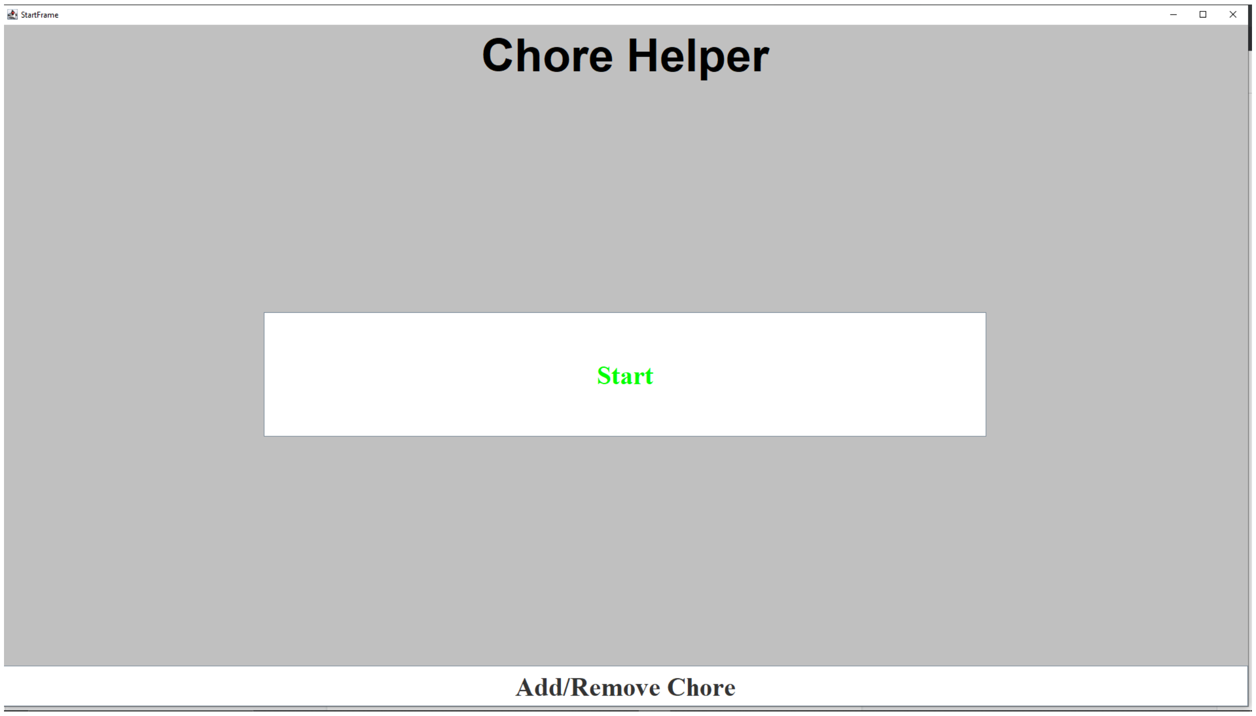
- Another example of inheritance

Use of Java Swing to make a GUI



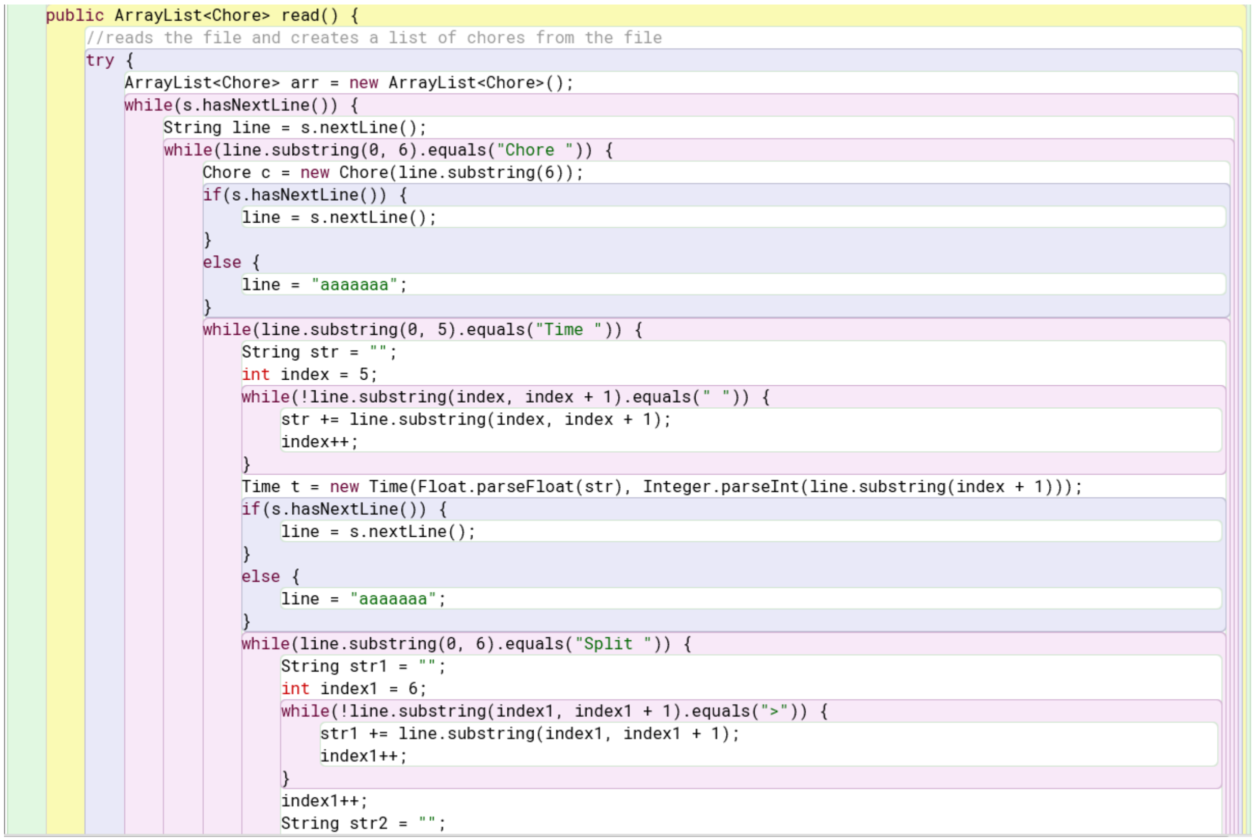
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.awt.Dimension;
public class GUI implements ActionListener {
    JFrame frame;
    CardLayout card;
    JPanel panel;
    JButton button, addChore;
    JTextField field, removeField;
    ArrayList<Chore> arr;
    //Creates the starting window when the application is open
```

- Example of how the GUI class imports Java Swing classes and uses Java Swing components in its fields
- Using Java Swing is necessary because without Java Swing, building the GUI would have been impossible because the Java Swing components are the objects that create a GUI in the first place

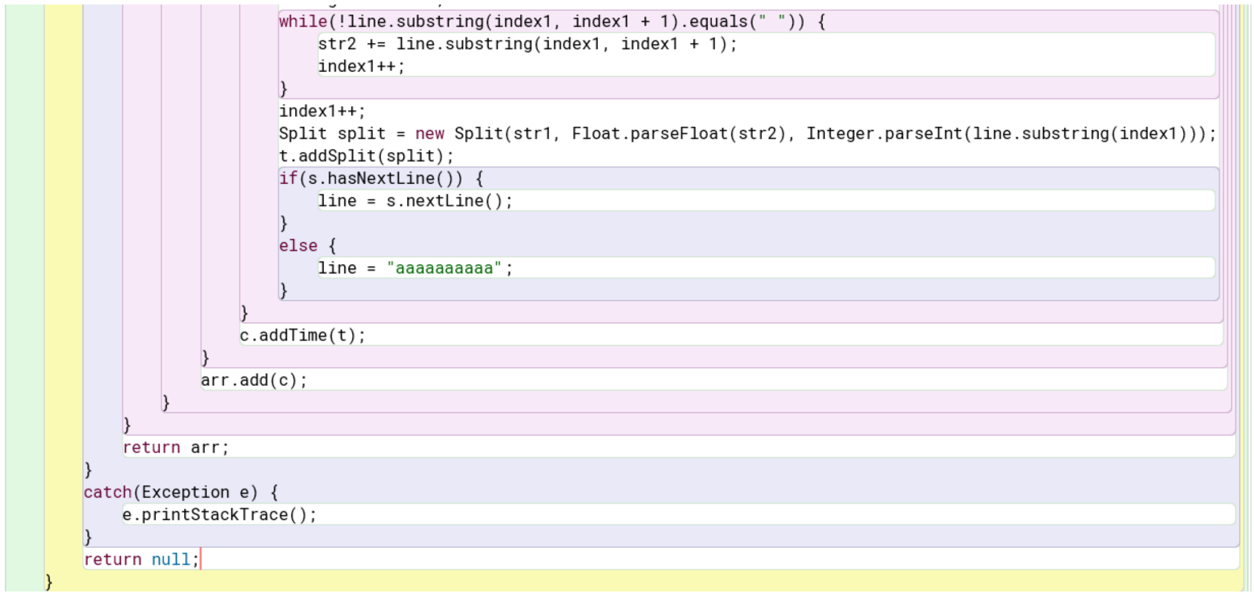


- Visual representation of Java Swing components

Algorithms



```java
public ArrayList<Chore> read() {
    //reads the file and creates a list of chores from the file
    try {
        ArrayList<Chore> arr = new ArrayList<Chore>();
        while(s.hasNextLine()) {
            String line = s.nextLine();
            while(line.substring(0, 6).equals("Chore ")) {
                Chore c = new Chore(line.substring(6));
                if(s.hasNextLine()) {
                    line = s.nextLine();
                }
                else {
                    line = "aaaaaaa";
                }
                while(line.substring(0, 5).equals("Time ")) {
                    String str = "";
                    int index = 5;
                    while(!line.substring(index, index + 1).equals(" ")) {
                        str += line.substring(index, index + 1);
                        index++;
                    }
                    Time t = new Time(Float.parseFloat(str), Integer.parseInt(line.substring(index + 1)));
                    if(s.hasNextLine()) {
                        line = s.nextLine();
                    }
                    else {
                        line = "aaaaaaa";
                    }
                    while(line.substring(0, 6).equals("Split ")) {
                        String str1 = "";
                        int index1 = 6;
                        while(!line.substring(index1, index1 + 1).equals(">")) {
                            str1 += line.substring(index1, index1 + 1);
                            index1++;
                        }
                        index1++;
                        String str2 = "";
```

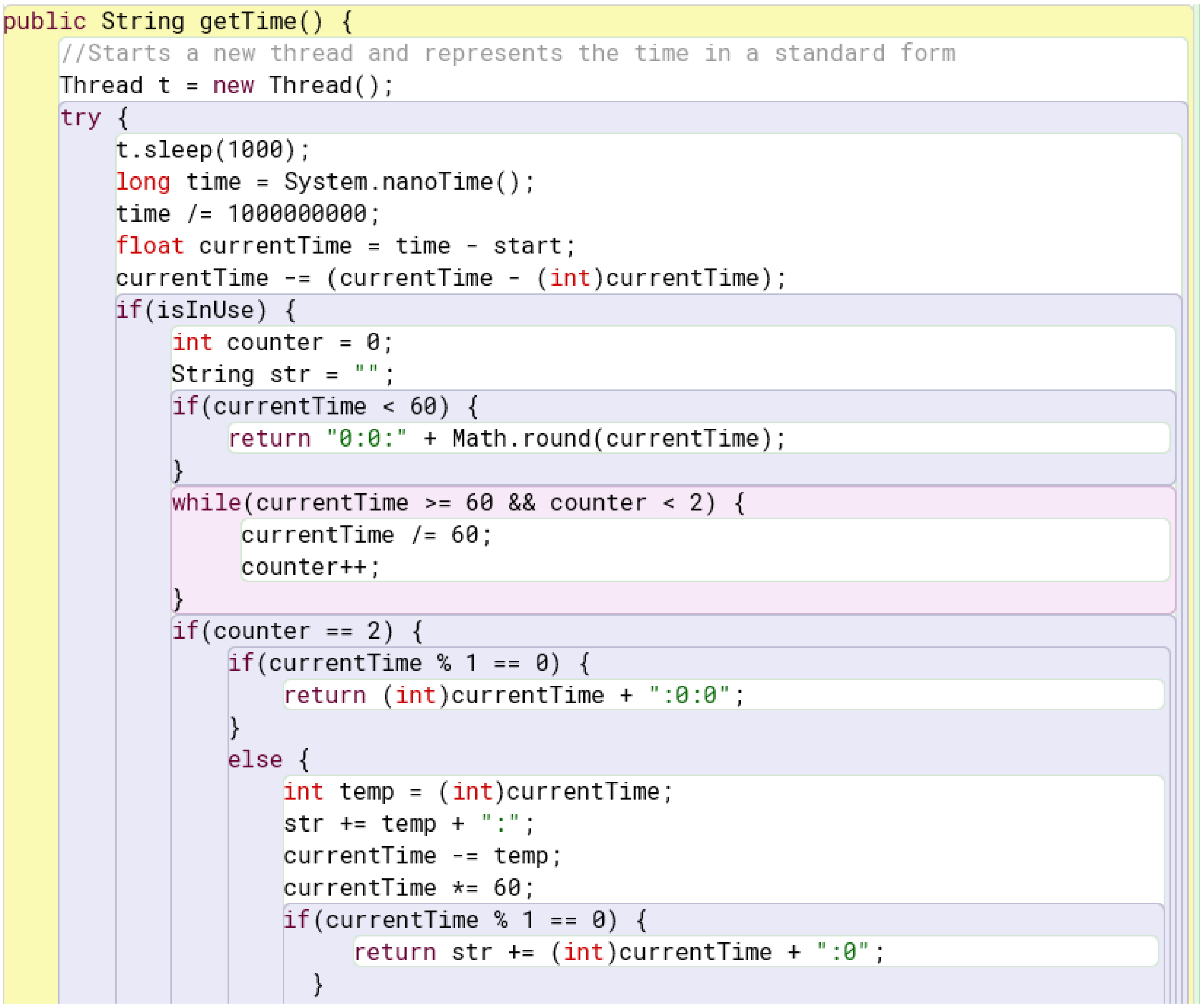


```java
                        while(!line.substring(index1, index1 + 1).equals(" ")) {
                            str2 += line.substring(index1, index1 + 1);
                            index1++;
                        }
                        index1++;
                        Split split = new Split(str1, Float.parseFloat(str2), Integer.parseInt(line.substring(index1)));
                        t.addSplit(split);
                        if(s.hasNextLine()) {
                            line = s.nextLine();
                        }
                        else {
                            line = "aaaaaaaaaa";
                        }
                    }
                    c.addTime(t);
                }
                arr.add(c);
            }
        }
        return arr;
    }
    catch(Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

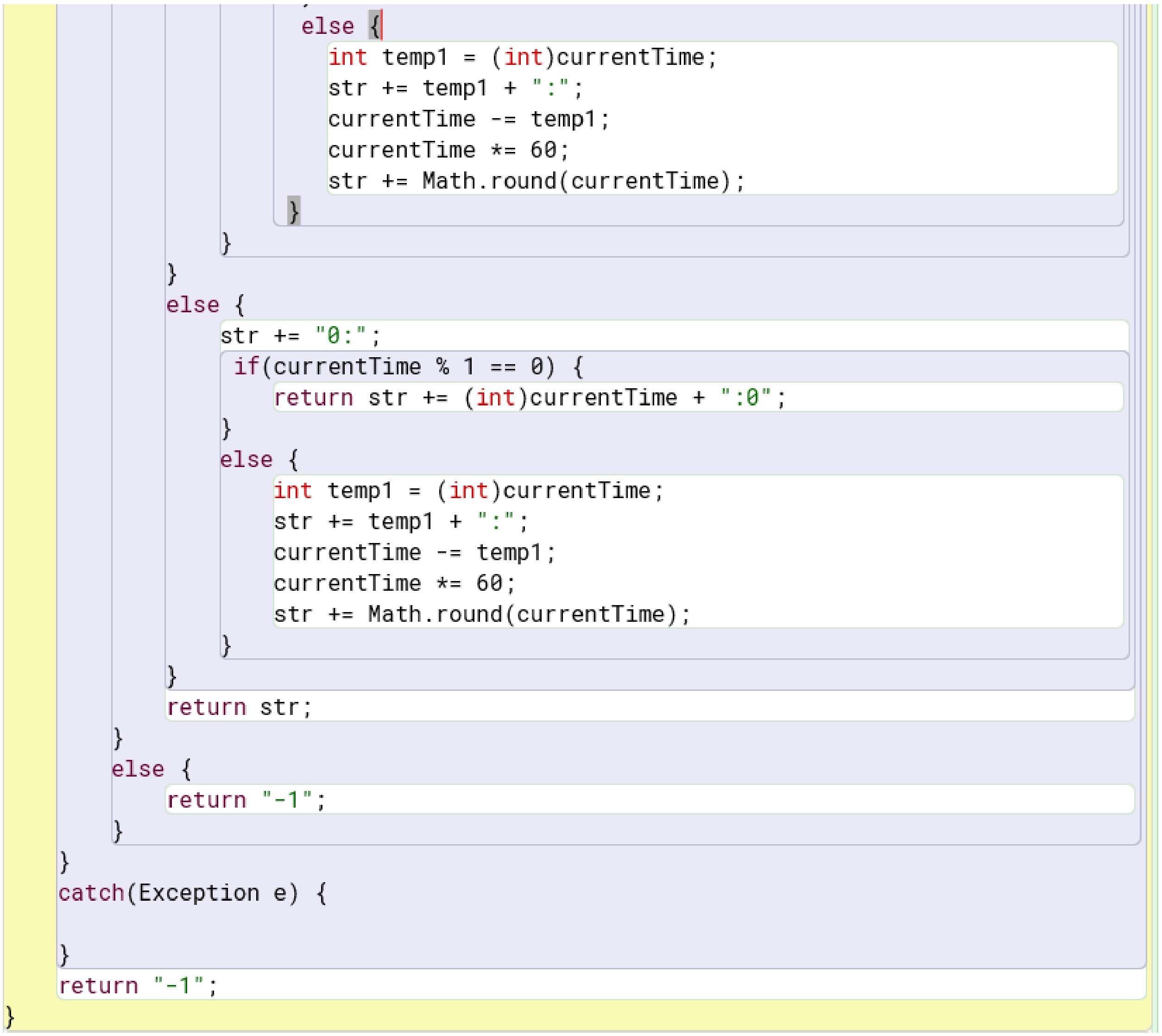
Algorithm that reads from a text file to return an ArrayList of Chores

- Uses file handling and nested while loops
- Loops through each line of the program and depending on the beginning determines if that line is a Time, Split, or Chore and creates objects that correspond to each line and adds them to the appropriate places to create Chores
- Handles files because the algorithm uses a Scanner object and reads each line of the file

- Uses nested while loops in order for the algorithm to work properly because it needs to continuously loop through each line in order to determine what object the line represents



```java
public String getTime() {
    //Starts a new thread and represents the time in a standard form
    Thread t = new Thread();
    try {
        t.sleep(1000);
        long time = System.nanoTime();
        time /= 1000000000;
        float currentTime = time - start;
        currentTime -= (currentTime - (int)currentTime);
        if(isInUse) {
            int counter = 0;
            String str = "";
            if(currentTime < 60) {
                return "0:0:" + Math.round(currentTime);
            }
            while(currentTime >= 60 && counter < 2) {
                 currentTime /= 60;
                 counter++;
            }
            if(counter == 2) {
                if(currentTime % 1 == 0) {
                    return (int)currentTime + ":0:0";
                }
                else {
                    int temp = (int)currentTime;
                    str += temp + ":";
                    currentTime -= temp;
                    currentTime *= 60;
                    if(currentTime % 1 == 0) {
                         return str += (int)currentTime + ":0";
                      }
```

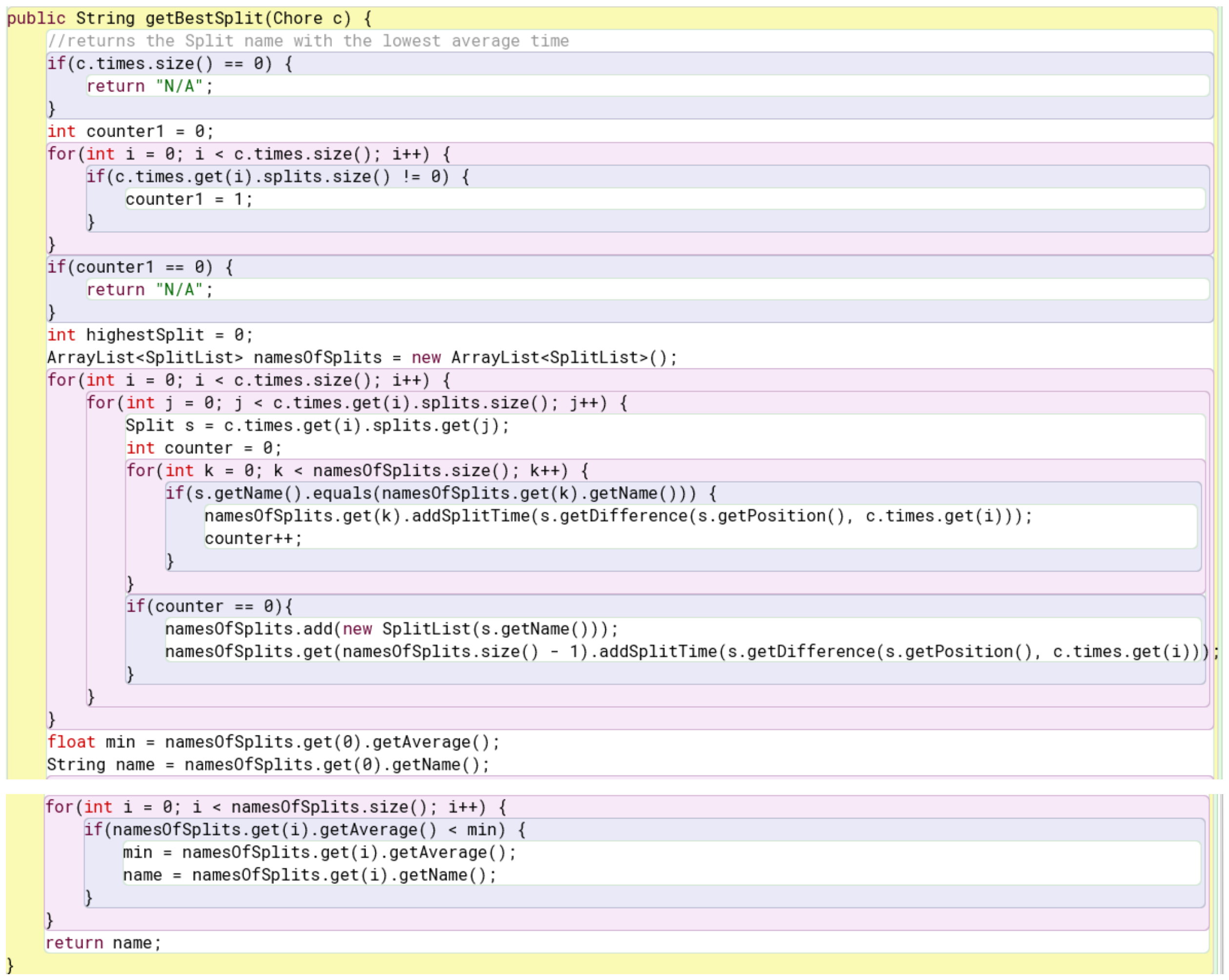


```
                      else {
                        int temp1 = (int)currentTime;
                        str += temp1 + ":";
                        currentTime -= temp1;
                        currentTime *= 60;
                        str += Math.round(currentTime);
                      }
                }
            }
            else {
                str += "0:";
                 if(currentTime % 1 == 0) {
                    return str += (int)currentTime + ":0";
                }
                else {
                    int temp1 = (int)currentTime;
                    str += temp1 + ":";
                    currentTime -= temp1;
                    currentTime *= 60;
                    str += Math.round(currentTime);
                }
            }
            return str;
        }
        else {
            return "-1";
        }
    }
    catch(Exception e) {

    }
    return "-1";
}
```

Algorithm that turns seconds into a standard form of time (hours:minutes:seconds). It also gets the time that has elapsed since the beginning of the activation of the stopwatch after 1 second

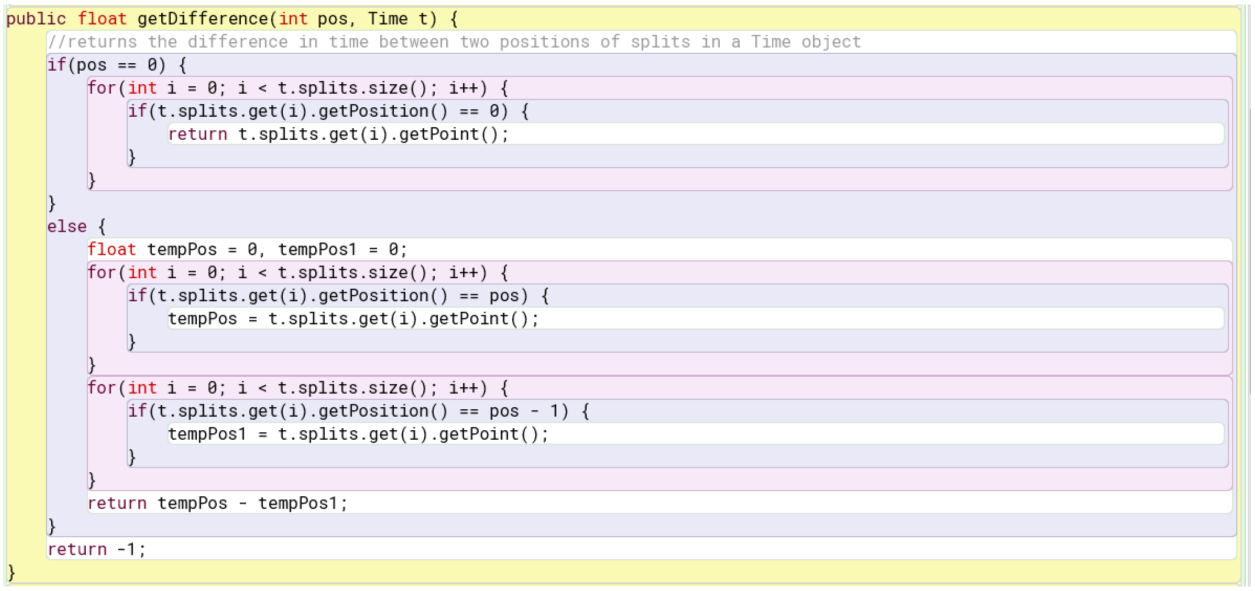
- Uses multithreading for the delay and uses conditional loops from converting seconds to the more standard time
- Also implements exception handling by catching an exception when t.sleep is called



```java
public String getBestSplit(Chore c) {
    //returns the Split name with the lowest average time
    if(c.times.size() == 0) {
        return "N/A";
    }
    int counter1 = 0;
    for(int i = 0; i < c.times.size(); i++) {
        if(c.times.get(i).splits.size() != 0) {
            counter1 = 1;
        }
    }
    if(counter1 == 0) {
        return "N/A";
    }
    int highestSplit = 0;
    ArrayList<SplitList> namesOfSplits = new ArrayList<SplitList>();
    for(int i = 0; i < c.times.size(); i++) {
        for(int j = 0; j < c.times.get(i).splits.size(); j++) {
            Split s = c.times.get(i).splits.get(j);
            int counter = 0;
            for(int k = 0; k < namesOfSplits.size(); k++) {
                if(s.getName().equals(namesOfSplits.get(k).getName())) {
                    namesOfSplits.get(k).addSplitTime(s.getDifference(s.getPosition(), c.times.get(i)));
                    counter++;
                }
            }
            if(counter == 0){
                namesOfSplits.add(new SplitList(s.getName()));
                namesOfSplits.get(namesOfSplits.size() - 1).addSplitTime(s.getDifference(s.getPosition(), c.times.get(i)));
            }
        }
    }
    float min = namesOfSplits.get(0).getAverage();
    String name = namesOfSplits.get(0).getName();
    for(int i = 0; i < namesOfSplits.size(); i++) {
        if(namesOfSplits.get(i).getAverage() < min) {
            min = namesOfSplits.get(i).getAverage();
            name = namesOfSplits.get(i).getName();
        }
    }
    return name;
}
```

Algorithm that takes in a Chore and loops through the Times and Splits of the Times and determines which Split is the fastest by taking the Splits with the same names and finding the difference between the Split before it and then storing it in a SplitList and taking the average of the floats of the SplitList to determine which Split is the fastest

- Uses nested for loops to loop through a Chore object to retrieve its Times and the Times' Splits



```java
public float getDifference(int pos, Time t) {
    //returns the difference in time between two positions of splits in a Time object
    if(pos == 0) {
        for(int i = 0; i < t.splits.size(); i++) {
            if(t.splits.get(i).getPosition() == 0) {
                return t.splits.get(i).getPoint();
            }
        }
    }
    else {
        float tempPos = 0, tempPos1 = 0;
        for(int i = 0; i < t.splits.size(); i++) {
            if(t.splits.get(i).getPosition() == pos) {
                tempPos = t.splits.get(i).getPoint();
            }
        }
        for(int i = 0; i < t.splits.size(); i++) {
            if(t.splits.get(i).getPosition() == pos - 1) {
                tempPos1 = t.splits.get(i).getPoint();
            }
        }
        return tempPos - tempPos1;
    }
    return -1;
}
```

Algorithm that determines the differences between Splits

- Uses conditional statements and for loops to determine the difference between Splits with differing positions

Word Count: 1090