eclipse-workspace - Portfolio Project Part II/src/concurrencyThreads.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- Calculate Average Witholding
- Capture Grade Statistics Using For-lo
- Car Data and MPG
- Creating a GUI Application
- Creating an Abstract Class Shape I
- Creating a User Interface II
- Final Project - Student Data Loop
- Get Weekly Temperatures
- HelloWorld
- Implementing a Superclass Employee
- Implementing Recusion to Provide a
- Method with Scanner Input
- Name and Address
- Portfolio Project Part II
  - JRE System Library [JavaSE-17]
  - src
    - (default package)
      - concurrencyThreads.java
- Storing an ArrayList
- Test Frame
- Test Program
- Vehicle Inventory Program

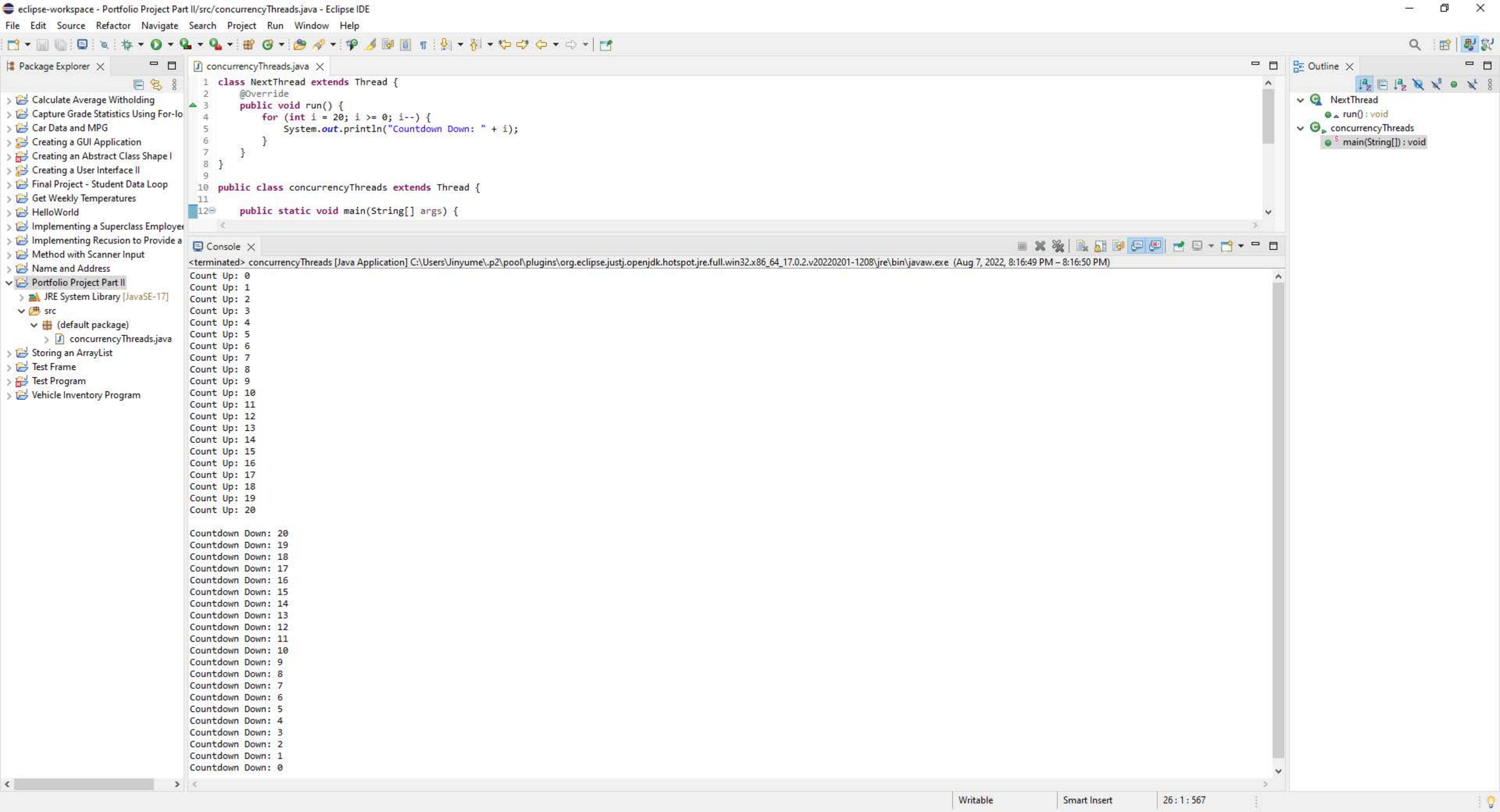
concurrencyThreads.java

```
1  class NextThread extends Thread {
2      @Override
3      public void run() {
4          for (int i = 20; i >= 0; i--) {
5              System.out.println("Countdown Down: " + i);
6          }
7      }
8  }
9
10 public class concurrencyThreads extends Thread {
11
12     public static void main(String[] args) {
13         Thread increasingThread = Thread.currentThread();
14         increasingThread.setPriority(MAX_PRIORITY);
15
16         for (int k = 0; k <= 20; k++) {
17             System.out.println("Count Up: " + k);
18         }
19
20         System.out.println("");
21         NextThread decreasingThread = new NextThread();
22         decreasingThread.start();
23     }
24 }
25
26
```

Outline

- NextThread
  - run() : void
- concurrencyThreads
  - main(String[]) : void

Portfolio Project Part II


eclipse-workspace - Portfolio Project Part II/src/concurrencyThreads.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Calculate Average Witholding
Capture Grade Statistics Using For-lo
Car Data and MPG
Creating a GUI Application
Creating an Abstract Class Shape I
Creating a User Interface II
Final Project - Student Data Loop
Get Weekly Temperatures
HelloWorld
Implementing a Superclass Employee
Implementing Recusion to Provide a
Method with Scanner Input
Name and Address
Portfolio Project Part II
JRE System Library [JavaSE-17]
src
(default package)
concurrencyThreads.java
Storing an ArrayList
Test Frame
Test Program
Vehicle Inventory Program
concurrencyThreads.java
1 class NextThread extends Thread {
2 @Override
3 public void run() {
4 for (int i = 20; i >= 0; i--) {
5 System.out.println("Countdown Down: " + i);
6 }
7 }
8 }
9
10 public class concurrencyThreads extends Thread {
11
12 public static void main(String[] args) {
Console
<terminated> concurrencyThreads [Java Application] C:\Users\Jinyume\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Aug 7, 2022, 8:16:49 PM – 8:16:50 PM)
Count Up: 0
Count Up: 1
Count Up: 2
Count Up: 3
Count Up: 4
Count Up: 5
Count Up: 6
Count Up: 7
Count Up: 8
Count Up: 9
Count Up: 10
Count Up: 11
Count Up: 12
Count Up: 13
Count Up: 14
Count Up: 15
Count Up: 16
Count Up: 17
Count Up: 18
Count Up: 19
Count Up: 20
Countdown Down: 20
Countdown Down: 19
Countdown Down: 18
Countdown Down: 17
Countdown Down: 16
Countdown Down: 15
Countdown Down: 14
Countdown Down: 13
Countdown Down: 12
Countdown Down: 11
Countdown Down: 10
Countdown Down: 9
Countdown Down: 8
Countdown Down: 7
Countdown Down: 6
Countdown Down: 5
Countdown Down: 4
Countdown Down: 3
Countdown Down: 2
Countdown Down: 1
Countdown Down: 0
Outline
NextThread
run() : void
concurrencyThreads
main(String[]) : void
Writable
Smart Insert
26 : 1 : 567

Clarissa Kuter

CSC450 Programming III

Reginald Haseltine

7 August 2022

Portfolio Project: Part 2

For the second half of this course's portfolio project, I was asked to once again demonstrate my understanding regarding concurrency. Unlike the previous week where I was tasked with creating a C++ program that would create two threads, this week I was tasked with creating a Java application that would create two threads. Like last week's program, these threads would act as counters. One thread would count to 20; after it reached 20, the second thread would start counting down to 0. Below, I cover the same concurrency concepts I addressed with the C++ version of my portfolio project; however, this time, the concepts are discussed in the context of Java and a Java application.

To start, I will be addressing the performance issues that may occur whilst a program is using concurrency. Performance issues can crop up whether it is a C++ or a Java program being run, and the reasons for the issues are often the same. As I stated with my C++ program, an application's performance can incur a drop in execution quality due to concurrency. This is because when multiple threads are running and executing several different processes, the core executing the program will constantly be swapping between each of the processes, slowing down the execution. Deadlocking is another performance issue that can occur in both languages; it

occurs when one or more threads block other threads and are unable to get out of it—thus, the threads are blocked forever. Because the threads are unable to make progress in a deadlock, starvation may occur. This will cause the program to perform incorrectly.

Next is the topic of vulnerabilities exhibited with the use of strings. While both C++ and Java are prone to performance issues when using concurrency, things are not quite the same with string vulnerabilities. Due to how the languages are written, format string vulnerabilities and buffer overflows—a common result of string vulnerabilities—are more often a C++ occurrence than a Java one. This is because Java and other languages like it allow programmers to maintain dynamic arrays and strings that help the languages to prevent and remain immune to common format errors. That said, while string vulnerabilities are more common to occur in a C++ program, they are not impossible to have in a Java program. If strings are left unprotected and open, they can still be controlled and manipulated in a Java program just like in a C++ program. Attackers could use it to write and read information to and from memory.

The final topic to address is that of the security of the data types exhibited. Java is a language that employs a large array of data types, program components, and modifiers. In some Java programs, there can be classes, functions, constructors, and more that may share the same names. As such, just as it was important with C++, it is equally important in Java that any threads accessing the same data types do not change the data or mistake it for a different data type of the same name. Unlike C++, Java also has the added security of four different access modifiers: default, public, private, and protected. By using these in their applications, programmers have an added layer of security that will help prevent data leakage.

The above issues can occur to anyone who is using concurrency and being unaware of the issues will make solving the errors that much harder. Thus, no matter what language a

programmer chooses to code in, it is important that they follow security protocols and familiarize themselves with the issues and errors that can arise. That said, if I were to choose a language, I believe I would choose Java, as it has extra measures written into it that make it less prone to some of the above issues, especially when compared to C++. One of the biggest being format string vulnerabilities. For me, it is the easier and slightly more secure program to write in, and the more beginner friendly one.