

ProductivityPlus

Java-based GUI Windows application used to increase productivity

Austin Ayers

C S4800-102

Project Overview

The goal of ProductivityPlus is to increase the program user's level of productivity while on the computer by analyzing which programs they use and how long they use them. This is aimed at increasing productivity because it allows the user to view where their time was spent, not only graphically but also with a detailed text analysis. ProductivityPlus will also have features to implement custom break intervals and reminders for the user to get back to work, along with prompting the user to enter what they are currently doing for future analysis of their own to see how productive they were.

Feature #1 – Program Time Counter (Priority level A)

The goal of ProductivityPlus is to improve a computer user's productivity while on a computer. This is made possible by the program's features, the more prevalent one being the program time tracker. The main feature of the app is to gather data about what programs a user is currently inside; the app calculates how much time the user spends in each program. When deciding how much time is spent in a program, the app takes whichever window is "in focus" and keeps track of how long it stays in focus. This is accomplished by using the Windows API, more specifically the Java Native Access (JNA) is used to call Windows API functions using method invocation.

The end goal of keeping track of how much time is spent in each application is to output the results graphically so the user understands where their time went while on their computer. This can be used to help a user understand what slows down their work process, help limit distractions, and altogether simply show how much time was spent in each program. This feature can also be used for freelance works to keep track of their timesheets for each project, since the Windows API keeps track of individual processes such as a single Chrome tab or a single Eclipse project. The results will not only be displayed graphically, but also in a text format such as a table for detailed viewing.

Feature #2 – Custom Break Timer (Priority level B)

Other features include popup reminders the user can set to limit breaks. Techniques have been developed such as the Pomodoro Technique (blocks of working where the worker spends 25 minutes working with a 5-minute break) which breaks down a person's time management into intervals where they work and take breaks. The key idea here is maintaining consistency with break times, which is where the popup reminder for when a break is ending comes in handy. A user can simply create their own break interval and be reminded when it is time to break or go back to work. This feature helps the user stick to a regular break schedule such as the Pomodoro to increase their productivity.

Feature #3 – Get Back to Work (Priority level C)

Lastly, a feature like the break-timer feature will be implemented so the user can keep track of individual events they are doing at chosen intervals. A user can set this notification to notify them every X minutes to keep track of what they were doing. For example, the user sets this feature to notify them every 30 minutes. At the 30-minute mark, the notification is triggered, and the user is prompted to answer the question "What are you working on?" to which the user states their current work. This is used to keep people on track and to keep a record of what they were doing.

Possible Users

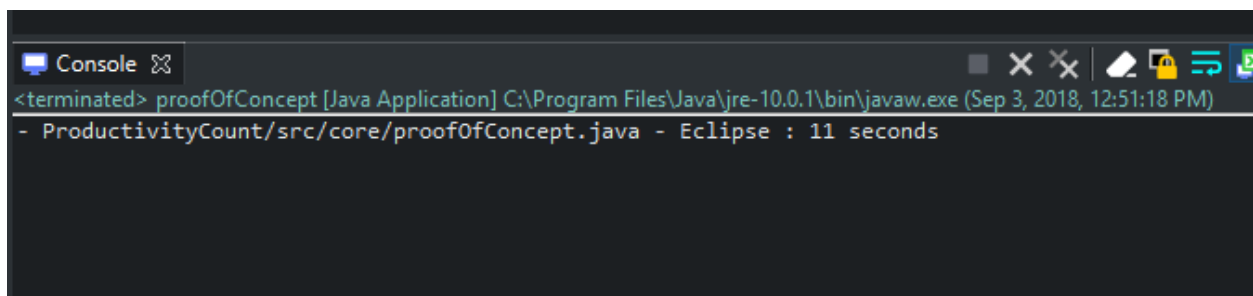
Anyone who uses a computer is a possible user of my project. Since the Windows API can be used to monitor individual processes, time spent in Chrome or any other application can be monitored. This opens the scope of who can and would want to use my application to almost anyone that wants to track their time for free. Freelancers that want to track where their time is spent is likely one of the largest groups of people that would use this software.

Similar Existing Work

Existing programs that measure how much time is spent in applications already exists, examples include Toggl (<https://toggl.com/>) and RescueTime (<https://www.rescuetime.com/>). These programs both measure active time in programs and output graphical results and analysis of time spent. I would not say my project is an improvement because my project branches off from the mentioned examples and focuses on maximizing work with the use of properly time breaks and has an option for “Get back to work” timers. Toggl and RescueTime are meant to analyze time spent and essentially maximize monetization for employees, while my project is more for personal use to stay productive on any assignment.

Previous Experience

I have had a fair amount of experience working with graphical components of programs in outside projects, so designing the GUI of this application will be like the processes I have used in the past. I have used JavaFX Scene Builder to build a few projects in the past, such as a GUI calculator and exercises to learn how to attach buttons and other GUI components to listeners. The language I am using to write this project is Java, which is the language I have had the most experience in. I have had to troubleshoot countless Java programs and find workarounds, so I feel confident that if any obstacle arose my previous experience at finding solutions will come into play. I will be using various data structures such as hash maps, all of which I have had experience with in my previously taken Data Structures class and with outside projects. To prove to myself that my risk is minimized, I wrote a very rudimentary version of my program using the Windows API function to see if I could time how long I spent in my Eclipse IDE. The result is barebones, but functional.



```
<terminated> proofOfConcept [Java Application] C:\Program Files\Java\jre-10.0.1\bin\javaw.exe (Sep 3, 2018, 12:51:18 PM)
- ProductivityCount/src/core/proofOfConcept.java - Eclipse : 11 seconds
```

Technology

As stated before, my language of choice for this project is Java and I will be using the newest version of Eclipse as my IDE of choice. I also need to use the Windows API to be able to extract information about what is currently running in the Windows system. To be able to use the Windows API inside my Java program, I also need to use the Java Native Access libraries. By using this library, I can access which programs are running, which program is in focus, and other program attributes like program name. As for the GUI, I will be using JavaFX Scene Builder to create the layout and style sheets. As for displaying the data collected graphically, I will be using a library called JFreeChart which works well with JavaFX components and allows for exporting the chart as an image for later use.

Risk Areas

Working with the Windows API is a high-risk area for my project. I will need to search the Windows API for specific function calls to be able to use the JNA. I have attempted to minimize this risk area by writing a simple mockup of my priority feature to see if I could locate the current window in focus and time how long it stays in focus. As the project moves on, I likely will have to recreate this process with different aspects of the WinAPI to prove that future steps and features are possible.

Even though I have experience with GUIs, they can still be challenging to create, and I know this project will push my experience and skills with GUIs further than I have before, which is why I have opted to instead of writing the GUI by hand to use Scene Builder to assist with the XML and style sheets.

I had never created a chart or graph in Java before, so I found a library I liked (JFreeChart) and constructed basic pie charts with dummy data to prove I could get it working.