Spencer Erickson

Milestone 2

04/10/23

The artifact chosen is from my CS-250 Software Development Lifecycle Course, created in September 2021. During the course, I worked on a top five vacation destinations slide show that was created using java; the code produces a simple java swing application that creates a slideshow of images with brief descriptions.

I chose this artifact as it allowed me to make enhancements that effectively displayed my skills in the software design and engineering category. The enhancements made to this artifact were first refactoring the Java code to improve quality and make it easier to read and maintain. This will reduce the likely hood of bugs and errors and make it easier for new features to be implemented down the road. Refactoring code demonstrates software design/ engineering skills by showcasing my ability to improve and maintain existing code without changing its external behaviors, which can lead to many benefits, such as increased performance and reduced technical debt. The second enhancement was the addition of two simple error-handling features. The error handling added to the original code was a try-catch block to handle any exceptions that may occur during the initialization. If an exception is caught, the catch block will print the stack trace of the exception to the console, which provides a detailed list of the method calls that led to the exception being thrown. This information can help diagnose and fix the error. By incorporating these error-handling features, I could display my code maintainability and user experience enhancement skills. Error handling will require careful consideration of the possible error scenarios and the appropriate actions to take in each case. This forces developers to think through the logic of the program and ensure that it is well-structured and easy to understand. This will make the code more maintainable in the long run, making it easier for other developers to understand and modify. In addition, an excellent error-handling system can enhance the user experience by providing helpful feedback when crashes or errors occur, explaining what went wrong making it more user-friendly in the long run. The third enhancement is a timer function that I implemented into the slideshow. The timer function in this Java class is used to control the automatic advancement of slides in a slideshow GUI application. The timer is initialized with the duration of each slide, which is set to 5000 milliseconds. The Timer object is created in the initComponent() method and has an ActionListener attached to it. The ActionListener calculates the elapsed time since the timer started, updates the timer label, and advances to the next slide if the timer has expired. The timer is started using the start() method of the Timer object. The start time of the timer is initialized to the current system time using the System.currentTimeMillis() method. The elapsed time is calculated by subtracting the start time and the pause time from the current system time.The timer is paused when the user clicks the btnPause button and resumed when the user clicks the btnResume button. The pause time is calculated by subtracting the current system time from the pause time.The timer function is used in conjunction with the CardLayout to display the next slide when the timer expires. This is done by calling the goNext() method, which increments the index of the current slide and displays the next slide.

I enhanced this artifact using my software development/ engineering skillset, which meets the course objectives. Moving forward, I will now look to do the same with the final category using this same artifact as discussed.

During the enhancements of this artifact, I faced a few challenges that I had to overcome, but doing so taught me a few things as well. When brainstorming ways to incorporate some error handling, I struggled since the actual code itself was relatively simple, and there were not too many issues to arise when executing the code. I ended up looking into the try-catch block as an option, as even though the program's current iteration was relatively simple and had not displayed any signs of instability, I figured adding this feature would protect the application from crashing. If errors occur, it can handle the exception more effectively. The following process I embarked on was refactoring the original code; my goal was to write the code more cleanly and optimize it as much as possible, making it easier to maintain and scalable, if need be, in the future. A lot was learned regarding error handling and, surprisingly, refactoring code on my behalf and improving in these areas will lead to more efficient high, performing code that's easy to read and maintain as well as protected from bugs, errors, and other faults they may lead to potential problems for the end user.