Template based on the Centers for Medicare & Medicaid Services, Information Security & Privacy Management's Assessment

Security Assessment Report

Version 1.0

May 1st, 2023

Table of Contents

1. Summary

The goal of this security assessment was to identify, categorize & remedy any security issues inside the Java project. Some major findings included memory leaks, and mishandling user inputs.

1. Assessment Scope

Github was used for VCS & storage, VSCode for debugging & development, Windows 10 + MacOS. No major limitations were encountered, but I did not have access to other OSes like Linux, and couldn't verify its use in other development environments/terminals.

2. Summary of Findings

Of the findings discovered during our assessment, 3 were considered High risks, 2 Moderate risks, 1 Low, and 1 Informational risks. The SWOT used for planning the assessment are broken down as shown in Figure 2.

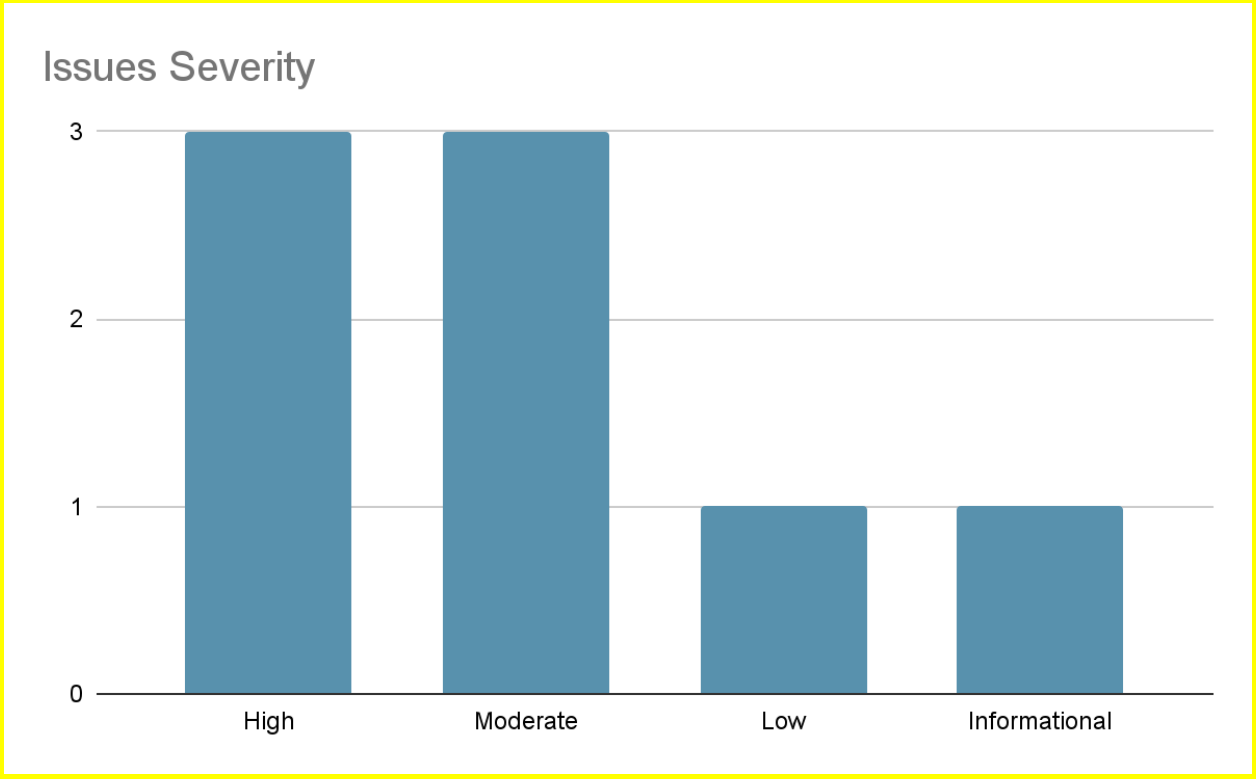


Figure 1. Findings by Risk Level

The major (high) issues found were common, but risky enough to be focused on more heavily than other items. For one, mishandling inputs was a big concern as the program would error when a input type was not what was expected. Second, the program would have unhandled

exceptions like the one above, in addition to going beyond a storage size of a variable. Lastly, the program had risks of malicious code possibly being inserted into code files by the user.

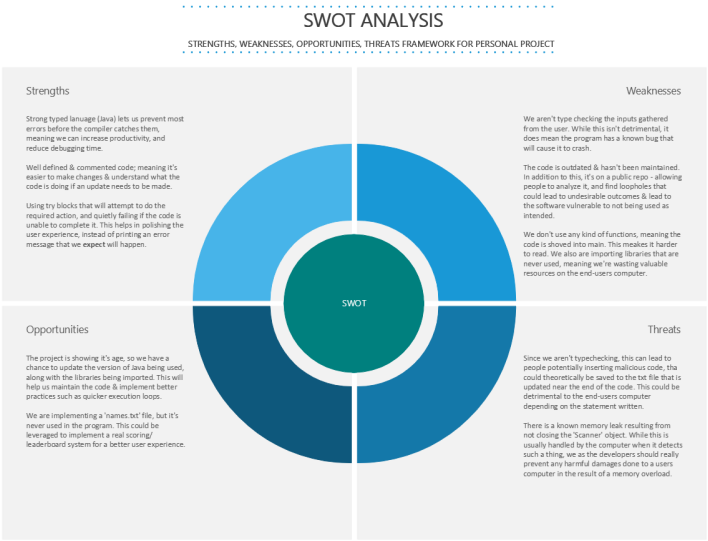


Figure 2. SWOT

Multiple issues resulted from this SWOT analysis, such as mishandling input types, error checking and the lack of classes/functions.

3. Summary of Recommendations

During the security updates, I issued some recommendations to the following issues:

- Upgrading the physical security of the computer (antivirus + bitlocker)
- Update & verify third party java libraries used
- Adding a license to the repository for use
- Implementing std testing with JUnit + extract code into classes
- Handle input & type checking

These fixes were quite simple to go through, although extracting the existing code into classes was a big undertaking. I still have a small amount more for security items to address in the future:

- Investing in a backup policy
- Implement an accounting process - logging errors

2. Goals, Findings, and Recommendations

1. Assessment Goals

The purpose of this assessment was to do the following:

- Demonstrate best practices & proper planning/maintenance by reviewing existing code to identify any security holes.
- Learning best security practices with handling common code vulnerabilities

2. Detailed Findings

Name	Severity	Explanation
1. Add License	Moderate	The github repo is public, so someone could have access to the code, but not make changes to the source that's stored on github itself. I don't have any plans to mitigate any damage if someone were to take my code & pirate it as their own, redistribute it under my alias, or anything of the sort. For starters, I'd need to add a license for distributability to cover myself against any accidents, and research how to better protect myself in the case of a leak. This determined to be an opportunity to ensure the code is properly formed & well-kept by adding a license. (Table 1 - Moderate)
2. Handle invalid inputs & type checking	High	This was a weakness, because although rare & not a huge issue as the program would just bomb out, the code clearly demonstrated a lack of understanding & professionalism by letting users input something invalid without handling it. (Table 1 - High)
3. Add unit testing	Moderate	This was another opportunity for the program to follow best practices & demonstrate well-formed code. With good testing, it ensures the code passes an initial QA phase by the developer for an integrity check. (Table 1 - Moderate)
4. Data/Memory leak	High	In java, the scanf() function can be used to open

Name	Severity	Explanation
		files, but the original program never closed it, so when it's left open it can lead to a memory leak for the users PC. Of course, it ends when the program ends, but it was a threat because it could cause potential harm to the device. (Table 1 - High)
5. Third party libs out of date	Low	In the original program, the libraries used had the potential to be out of date. After review however, the libraries seemed to have not been updated since they are mostly std libs. This was a low risk, and was a weakness of the project. (Table 1 - Low)
6. Encapsulate/Extract program into classes/functions	High	In order to follow best practices and gain some extra security by extracting the code, the program needed to be formed into functions for most of the code such as UI, updates, etc. This was a weakness of the code, and was dealt with in an update. (Table 1 - High)
7. Accounting process	Moderate	This isn't a huge priority, but it would be very nice for the program to have some kind of error logging process that didn't interrupt the main program, and rather silently record errors to a file. This was a good opportunity for the program to have more polish, and better security insights to error logging. (Table 1 - Moderate)
8. Invest in backup policy	Informational	The backup policy wasn't an immediate push since mostly everything the computer does is stored in the cloud, but it was listed as I don't currently have any physical backups for the code. This is a good security opportunity to ensure I have multiple copies of this code if something were to happen to my PC/Github repo. (Table 1 - Informational)

3. Recommendations

Here's where your fixes go (ensure you reference Table 2 for your ease of fix evaluation and explain why it matches that category).

1. Invest in a backup policy by using external device drives such as thumbdrives, hard drives, or any other physical storage device (#8) (Table 2 - Easy)
2. Invest in an accounting process such as adding a txt file or email system that tracks any errors that happen with the program (#7) (Table 2 - Very Difficult)
3. Encapsulate/extract the program into classes & functions to increase data hiding, along with handling any memory leaks by closing read functions (#6, #4) (Table 2 - Moderately Difficult)
4. Update & verify third party libraries integrity, along with adding unit testing to anywhere appropriate AFTER the above issue is resolved (#5, #3) (Table 2 - Easy)
5. Add a license to the github repository for ensured accountability and distributability (#1) (Table 2 - Easy)
6. Handle type checking & input checking from the user to prevent program crashes (#2) (Table 2 - Easy)

3. Methodology for the Security Control Assessment

3.1.1 Risk Level Assessment

Each Business Risk has been assigned a Risk Level value of High, Moderate, or Low. The rating is, in actuality, an assessment of the priority with which each Business Risk will be viewed. The definitions in Table 1 apply to risk level assessment values (based on probability and severity of risk). While Table 2 describes the estimation values used for a risk's "ease-of-fix".

Table 1 - Risk Values

Rating	Definition of Risk Rating
High Risk	Exploitation of the technical or procedural vulnerability will cause substantial harm to the business processes. Significant political, financial, and legal damage is likely to result
Moderate Risk	Exploitation of the technical or procedural vulnerability will significantly impact the confidentiality, integrity and/or availability of the system, or data. Exploitation of the vulnerability may cause moderate financial loss or public embarrassment to organization.
Low Risk	Exploitation of the technical or procedural vulnerability will cause minimal impact to operations. The confidentiality, integrity and availability of sensitive information are not at risk of compromise. Exploitation of the vulnerability may cause slight financial loss or public embarrassment
Informational	An "Informational" finding, is a risk that has been identified during this assessment which is reassigned to another Major Application (MA) or General Support System (GSS). As these already exist or are handled by a different department, the informational finding will simply be noted as it is not the responsibility of this group to create a Corrective Action Plan.
Observations	An observation risk will need to be "watched" as it may arise as a result of various changes raising it to a higher risk category. However, until and unless the change happens it remains a low risk.

Table 2 - Ease of Fix Definitions

Rating	Definition of Risk Rating
Easy	The corrective action(s) can be completed quickly with minimal resources, and without causing disruption to the system or data
Moderately Difficult	Remediation efforts will likely cause a noticeable service disruption • A vendor patch or major configuration change may be required to close the vulnerability • An upgrade to a different version of the software may be required to address the impact severity

Rating	Definition of Risk Rating
	• The system may require a reconfiguration to mitigate the threat exposure • Corrective action may require construction or significant alterations to the manner in which business is undertaken
Very Difficult	The high risk of substantial service disruption makes it impractical to complete the corrective action for mission critical systems without careful scheduling • An obscure, hard-to-find vendor patch may be required to close the vulnerability • Significant, time-consuming configuration changes may be required to address the threat exposure or impact severity • Corrective action requires major construction or redesign of an entire business process
No Known Fix	No known solution to the problem currently exists. The Risk may require the Business Owner to: • Discontinue use of the software or protocol • Isolate the information system within the enterprise, thereby eliminating reliance on the system In some cases, the vulnerability is due to a design-level flaw that cannot be resolved through the application of vendor patches or the reconfiguration of the system. If the system is critical and must be used to support on-going business functions, no less than quarterly monitoring shall be conducted by the Business Owner, and reviewed by IS Management, to validate that security incidents have not occurred

3.1.2 Tests and Analyses

This was completed using standard unit testing provided through JUnit, and through professor feedback on code - pointing out vulnerability with scanf() being dangerous -. Other practices such as basic code review, program demonstrations & pen-testing were used here too.

3.1.2.1 **Pen-testing process**:

For pen-testing, I simply used basic code demonstration by running the code, and using white box testing to target certain sections of code that I believed to be vulnerable.

3.1.2.2 **JUnit**:

I used JUnit to verify multiple formats of functions to ensure each function could receive two different variations of cases to account for pass/fail conditions.

3.1.2.3 **Analysis:**

“If you do not close the Scanner then Java will not garbage collect the Scanner object and you will have a memory leak in your program” (“Using Java's Scanner and Console Class to Read Input”)

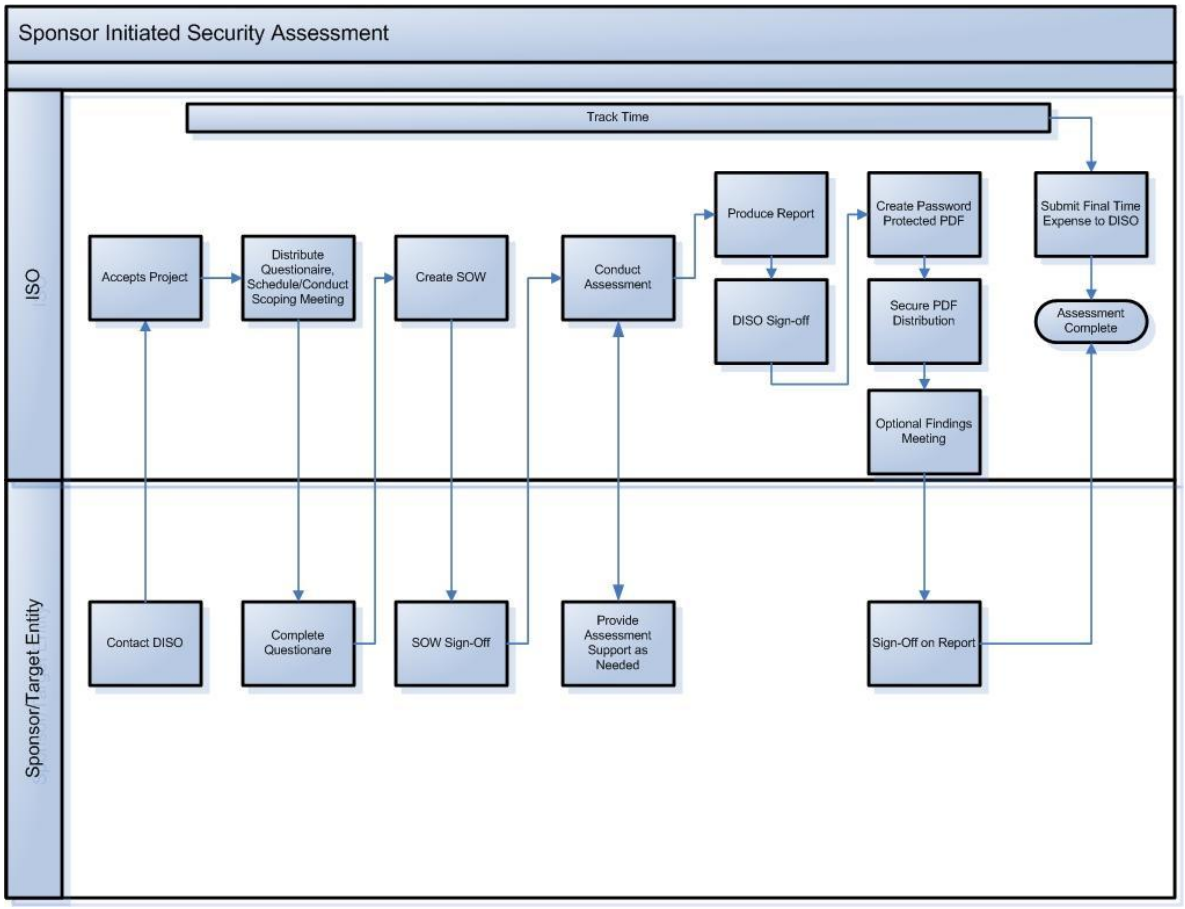
It was concluded that most errors, excluding the one above, were caused by the programmer’s lack of experience. The scanf() function is unique in that it is specific to Java, but the programmer should’ve still read the documentation before utilizing it.

3.1.3 Tools

This was completed using the VSCode internal debugging tool to set breakpoints, testing inputs with quitting, and verify no background codes were thrown. I also used githubs integration checks to verify new code would merge with the old code without conflicts.

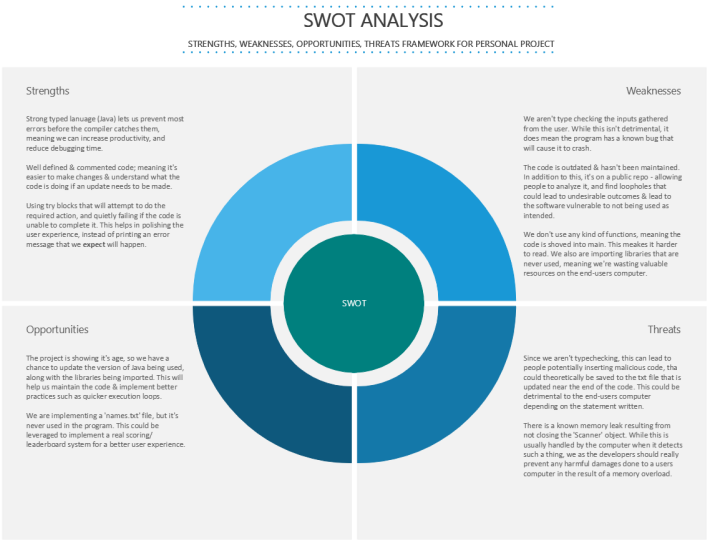
4. Figures and Code

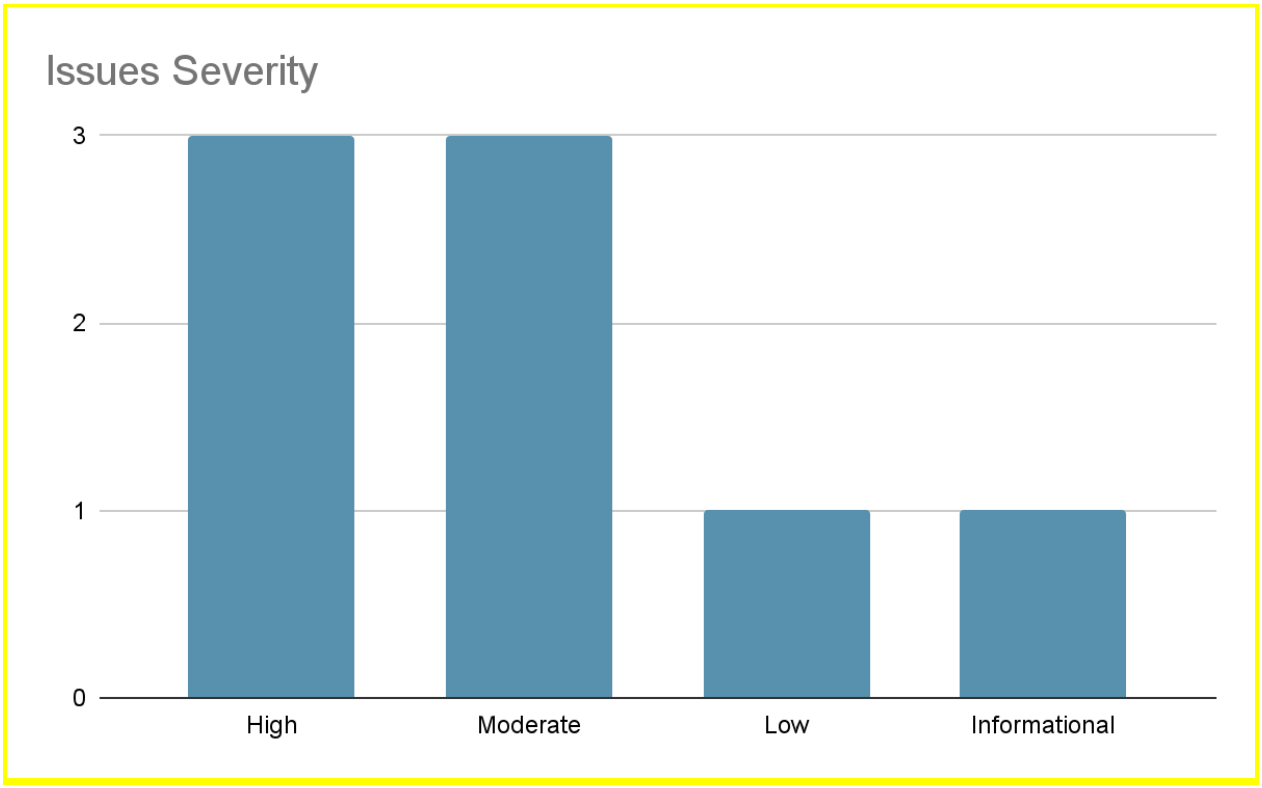
4.1.1 Process or Data flow of System (this one just describes the process for requesting), use-cases, security checklist, graphs, etc.

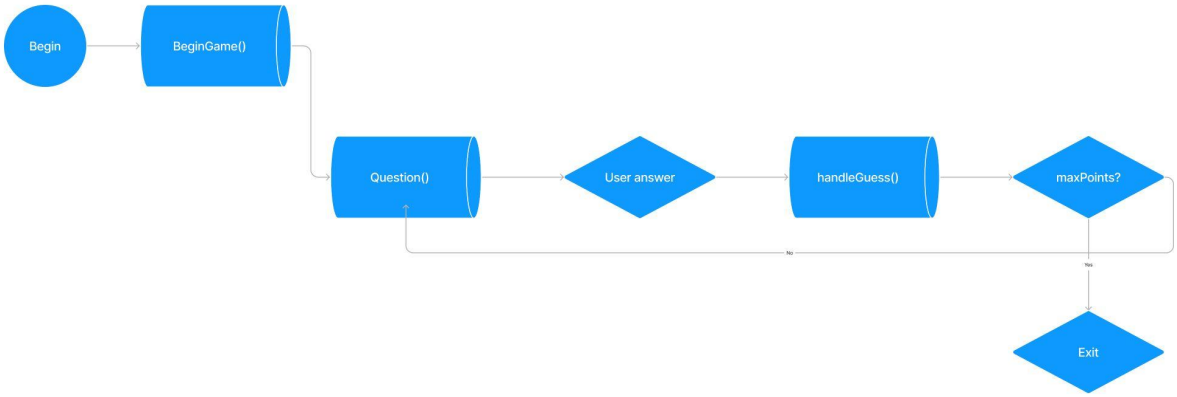


		Probability ---------->				
Severity		Frequent	Probable	Likely	Possible	Rare
|	Emergency			Unhandled exceptions		
|	Major		Wrong input type entered	Malicious code inserted into txt files		
|	Moderate		No permissions to read/write to txt files		Script to enter numbers too fast to console	
|	Minor		Invalid characters entered	Duplicate names on scores.txt	Memory leak	Too many entries in txt file (too large)
|	Negatable	Unused library resources				

Security Assessment – Java Guessing Game







tjmacphee/Cop-2006: Guessing game made with java (github.com)

1. Begin game
    a. Init points
    b. Init question
2. Prompt user for input
3. Take input
4. Check input
5. Update scores for correct/incorrect
6. Check if game over
    a. Exit
7. Repeat

4.1.2 Other figure of code

```
// try block gives scan() a chance to close
try (Scanner scan = new Scanner(System.in)) {
```

```
import org.junit.Before;
import org.junit.Test;
```

5. Works Cited

Greenwell, Josiah. *Canvas Comment/Suggestion*. "Checkin looks good, other things to think about: proper input handling (scan.nextInt(); should always scanNextLine() after to get rid of \n)". 17 April 2023.

Greenwell, Josiah. *Canvas Comment/Suggestion*. "Checkin looks good, other things to think about: proper input handling (scan.nextInt(); should always scanNextLine() after to get rid of \n)". 17 April 2023.

“Java User Input (Scanner class).” *W3Schools*, https://www.w3schools.com/java/java_user_input.asp. Accessed 1 May 2023.

Li, Vickie. “Common vulnerabilities in Java and how to fix them.” *Security Boulevard*, 30 November 2021, https://securityboulevard.com/2021/11/common-vulnerabilities-in-java-and-how-to-fix-them/. Accessed 1 May 2023.

McKenzie, Cameron. “Java Scanner next() vs nextLine() methods: What's the difference?” *TheServerSide*, 10 September 2022, https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Java-Scanner-next-vs-nextLine-methods-Whats-the-difference. Accessed 1 May 2023.

“Using Java's Scanner and Console Class to Read Input.” *UTK EECS*, http://web.eecs.utk.edu/~bvanderz/teaching/cs365Sp17/examples/datacheck.html. Accessed 1 May 2023.