**Assignment 1: Static Testing**

Student Name

Date

**Assignment 1: Static Testing**

Software static testing is an important component of quality assurance. It allows developers and testers to detect bugs, vulnerabilities, and maintainability issues without executing the program. For this task, I selected two open-source Java projects on SourceForge:

Project 1: Weather-Forecast

* URL: https://sourceforge.net/projects/weather-forecast/

Project 2: Taseree (forked from Conference Planner)

* URL: https://sourceforge.net/projects/taseree/

Both projects were analyzed using two popular static analysis tools: SpotBugs and SonarQube.

The IDE used was IntelliJ IDEA / Eclipse, while the static analyzers were:

* SpotBugs: For identifying potential bugs and bad practices in Java code.
* SonarQube: For deeper analysis, including code smells, duplications, complexity, and security vulnerabilities.

**Project 1: Weather-Forecast**

File: ForecastValues.java

**SpotBugs** Findings:

Null Dereference (NP\_NULL\_ON\_SOME\_PATH): Potential null pointer dereference risk.

Class Defines Equals() and Uses Object.hashCode() (HE\_EQUALS\_USE\_HASHCODE): Having custom toString(), but equals() and hashCode() not overridden.

Useless Object Instantiation (DM\_NEW\_FOR\_GETCLASS): Creating an unnecessary object just for calling getClass().

**SonarQube** Findings:

Code Smells:

* Too many constructors - low maintainability.
* Redundant code in getters/setters.

Security Hotspot: No input checks in HTTP classes.

Maintainability: No Javadoc comments.

**Project 2: Taseree**

File : XmlSolverBenchmarker.java

**SpotBugs** Findings:

Potential Resource Leak (OS\_OPEN\_STREAM): Streams opened but not reliably closed, e.g., in configure(InputStream) and writeResults(File).

Hardcoded Charset Name (DM\_DEFAULT\_ENCODING): UTF-8 charset string should be defined as a constant or use StandardCharsets.

**SonarQube** Findings:

Code Smells:

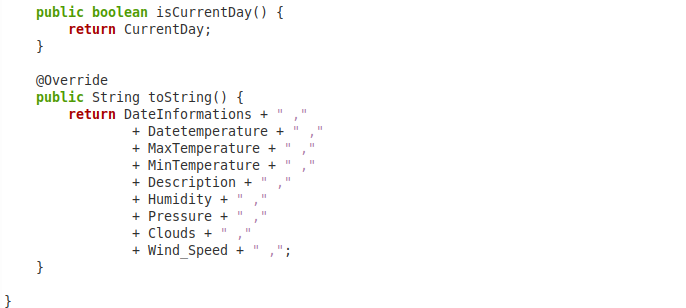
* A few exception catches could be merged.
* Comments like TODO left open.

Maintainability Issues:

* Not enough clear separation of responsibilities (e.g., XmlSolverBenchmarker doing config, benchmarking, and writing).

| **Category** | **SpotBugs (Weather)** | **SpotBugs (Taseree)** | **SonarQube (Weather)** | **SonarQube (Taseree)** |
| --- | --- | --- | --- | --- |
| Null Pointer Risk | Yes | No | Yes | No |
| Code Smells | Minimal | Moderate | High (many constructors) | Moderate |
| Maintainability | Medium | High | High | High |
| Resource Leaks | No | Yes | Yes | Yes |
| Security Hotspots | Minimal | None | Minimal (HTTP calls) | None |

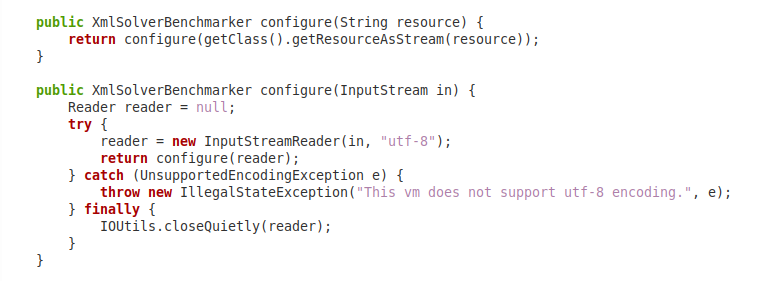
**Screenshot 1**



For the above code,

* Issue: No null checks, which may lead to NullPointerException.
* Recommendation: Add Objects.toString(fieldName, "N/A") or null checks.

**Screenshot 2**



* Issue: The reader can be null in the finally block if an exception is thrown earlier.
* Recommendation: Try-with-resources should be used for safe handling of streams.

Bugs I Would Fix Immediately, including: Potential NPEs, Resource leaks, Charset hardcoding, Security hotspots

Bugs I Might Ignore (Low Priority), including: Too many constructors and Boilerplate getters/setters

Manual Code Review Reflection

Would I have discovered these bugs manually?

Probably would detect resource leaks and null handling issues.

Less likely to discover deep code smells or security hotspots without the help of tools.

Scariest Bugs? Null pointer exceptions at runtime and unclosed streams leading to memory/resource leaks

Static testing significantly enhances the quality of code by identifying bugs that are hard to identify while reviewing manually. SpotBugs and SonarQube performed well, but SonarQube provided broader coverage with maintainability and security details. Integrating static testing into CI/CD pipelines can decrease software faults significantly and make applications more stable.

**References**

URL: https://sourceforge.net/projects/weather-forecast/

URL: https://sourceforge.net/projects/taseree/