|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Issue | Vulnerability | Bug | Code Smell |  |
| Recourses should be closed |  | 🗸 |  |  |
| Cognitive Complexity |  |  | 🗸 | Effect the **Maintain**: because of the difficulty in understanding the code. |
| Duplicated |  |  | 🗸 |  |
| Null Pointer Exception |  | 🗸 |  | Effect **Integrity** |
| Nested Catch-Try Blocks |  |  | 🗸 | Effect **Reusability**: when using nested blocks it will make it complex to understand the code then to reuse it .  Effect **Maintain** : it will be hard to maintain the code if it was too complex |
| Conditionally Executed is Unreachable |  | 🗸 |  | Effect **Reliability** : If the code is buggy code it will make the user uncomfortable to use this kind of code |
| Comment Block |  |  | 🗸 | Effect **Correctness** **and Maintainability:** by not following the required standers and code convention it will lead to make the code hard to maintain and complex to understand |
| Issue | Vulnerability | Bug | Code smell |  |
| Methods are Empty |  |  | 🗸 | Effect **Reliability**: an unexpected behavior in production will appear in empty methods |
| Child class fields shouldn’t shadow parent class Fields |  |  | 🗸 | Effect **Reusability** : it confuse the developer and make it hard to reuse |
| Serializable class |  |  | 🗸 | Effect **Efficacy** : this process is inefficient when it comes to memory utilization |
| Throw Generic Exception |  |  | 🗸 | Effect **Testability** :generic exception blocks a high-level class like Exception can obscure exceptions that deserve special treatment or that should not be caught at this point in the program |

4. false positive and false negative issues

SonarQube it has limitations and may not catch all false positives or false negatives.

***False positives*** are cases where SonarQube reports an issue that is not actually a problem in the code. This can happen when SonarQube is not configured correctly, or when it encounters a code construct that it doesn't fully understand. In some cases, false positives may also occur due to the complexity of the code being analyzed, or due to limitations in the analysis engine itself.

***False negatives***, on the other hand, are cases where SonarQube fails to report an issue that is actually present in the code. This can happen when SonarQube's rules do not cover all possible cases of a particular issue, or when the code being analyzed is too complex for SonarQube to fully understand.

False Positive :

* Here when I was using java JDK 11version SonarQube give me this :



But the code work perfectly with it .

False Negative :

* Magic Number

