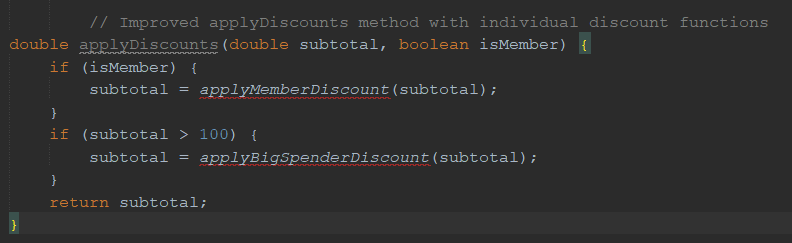
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Category** | | **Criteria** | | **Score – Y/N** | | **Comments** | |
| **Code Formatting** | | Is the code consistently  formatted (indentation, spacing)? | | 5 | | Consistent with proper identation and spacing | |
| Are line lengths reasonable and within guidelines? | | 5 | | Ensures no readability isssues | |
| Are there unnecessary blank  lines or spaces? | | 5 | | NO blank lines, it keeps the code clean | |
| **Naming Conventions** | | Are variable and function names descriptive and  meaningful? | | 5 | | Variable and function names are mostly descriptive. | |
| Are names chosen to reflect the actual role of the  variable, function, or class? | | 5 | | Names generally reflect their roles | |
| Is naming consistent with typical conventions for that language (e.g., camelCase  for variables in Java)? | | 5 | | The naming conventions are consistent with java standards | |
| **Comments and Documentation** | | Is the code self-explanatory, or are comments covering  up poor code readability? | | 3 | | Some methods could use additional documentation to explain logic. | |
| Do comments explain **why**  something is done, not just what is being done? | | 3 | | The comments focus on what is done rather than the reasoning behind decisions | |
| If comments/documentation exist, do they follow industry standards (e.g.,  Javadoc for Java)? | | 2 | | There is no formal documentation comments for methods. Javadoc could be used | |
| **Code Structure** | | Is the code modular and divided into logical  functions/methods? | | 5 | | The code is modular | |
| Are there any long methods/functions that  should be broken down? | | 4 | | ApplyDiscounts could benefit from further decomposition for clarity | |
| Is there any duplicated code  that should be refactored? | | 5 | | There is no noticeable duplicated code | |
| **Logic and Functionality** | | Does the code function as  intended? | | 5 | | The code functions correctly | |
| Are all the requirements  fulfilled? | | 5 | | Most requirements are met | |
| Are there any obvious logic  errors? | | 4 | | The discount logic could be clearer, for handling multiple discounts | |
| Are loop cases handled  properly? | | 5 | | The loops are well handled, iterating correctly the items | |
| **Error Handling** | | Are errors and exceptions properly caught and  handled? | | 3 | | Error handling is minimal. Validation for price and quantity would help prevent incorrect values from being set | |
| Is there any potential for  unhandled exceptions? | | 3 | | Unhandled exceptions may occur if incorrect values are passed.  Adding validation would improve reliability | |
| **Performance and Efficiency** | | Are there any obvious  performance bottlenecks? | | 5 | | There are no noticeable performance bottlenecks | |
| Is the code optimized for performance where  necessary? | | 5 | | There are no unnecessary computations or redundant code | |
| **Security** | | Are there any security  vulnerabilities in the code? | | 3 | | Minimal input validation poses a risk if negative or unrealistic values are entered. | |
| Is input validation properly  implemented? | | 3 | | Input validation is lacking, particularly for quantity and price fields, which could prevent invalid entries | |

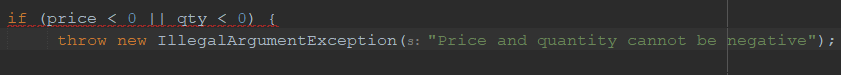
**Key Suggestions:**

* **Error Handling:** Add validation for price and quantity in the Items constructor or setters to prevent unrealistic values (e.g., negative numbers).
* **Discount Logic:** Refactor the applyDiscounts method to split discount logic into individual methods for member and big spender discounts. This approach will clarify which discount is applied and in what order.



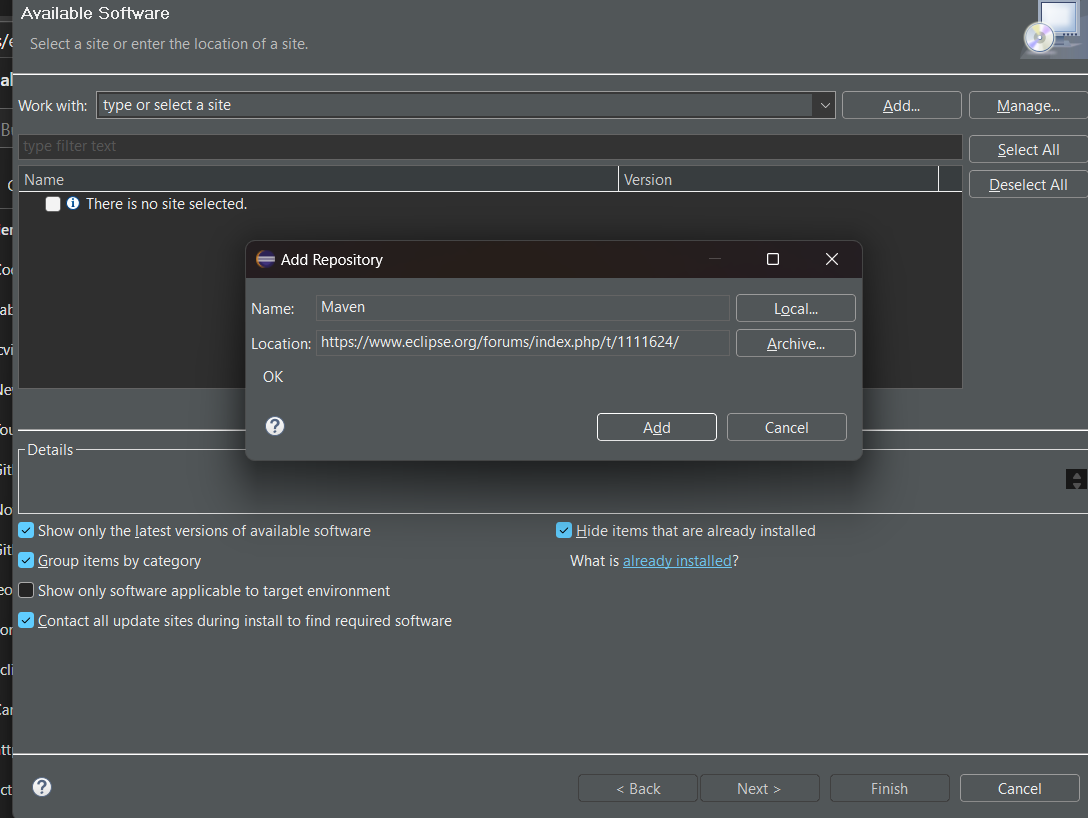
Breaking down applyDiscounts into smaller methods enhances readability by isolating each discount rule. Each method has a single responsibility, making the code more readable and easier to follow.

Implement basic validation for Items class:



By adding the check if (price < 0 || qty < 0), we ensure that any attempt to create an Items object with invalid negative values for price or qty will throw an IllegalArgumentException, preventing the creation of objects with incorrect state.

Installing Maven

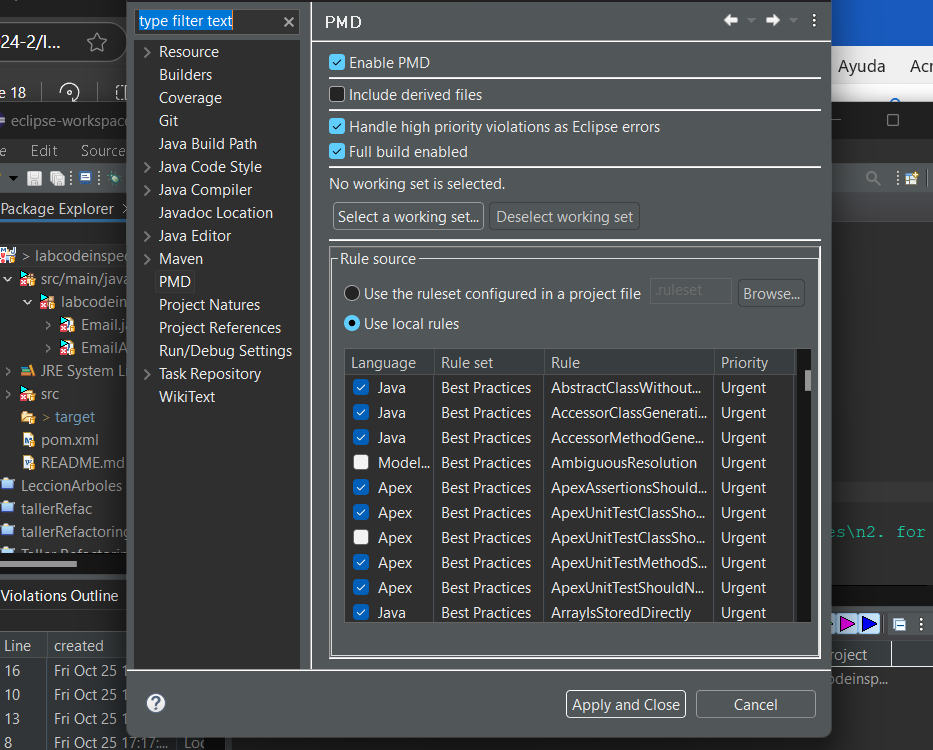


Installing PMD

Captura de pantalla de un celular

Descripción generada automáticamente

Enabling PMD

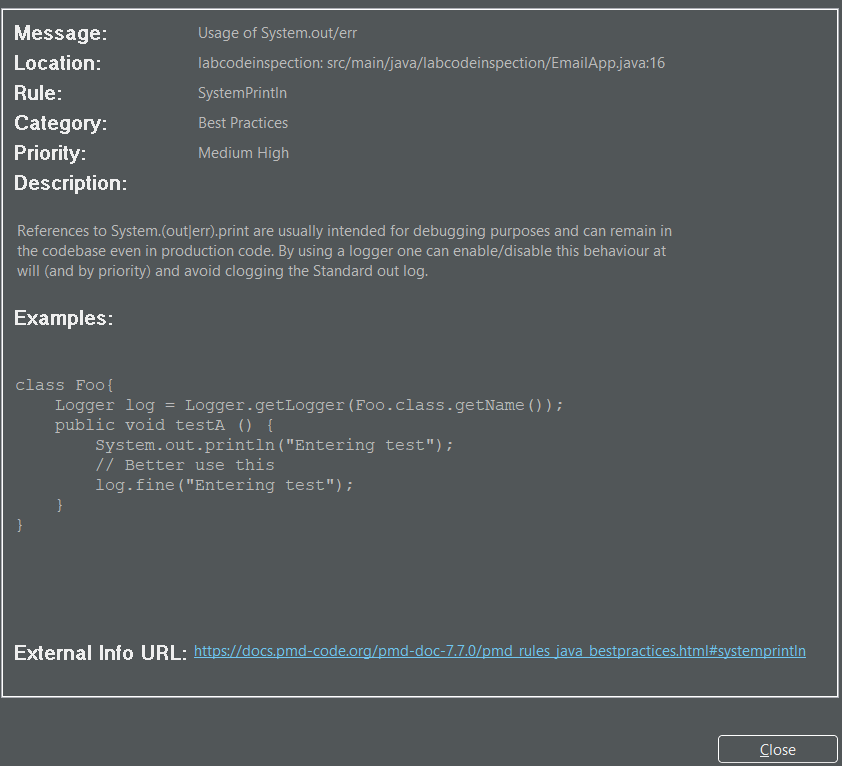


Running PMD

Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

Descripción generada automáticamente

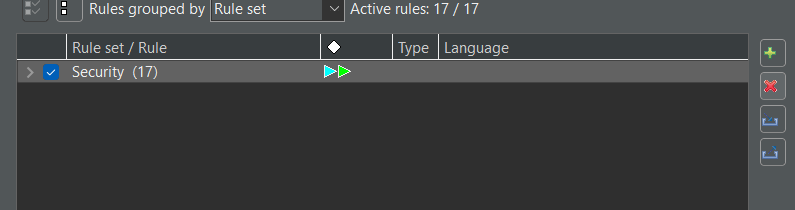
Showing details of violations



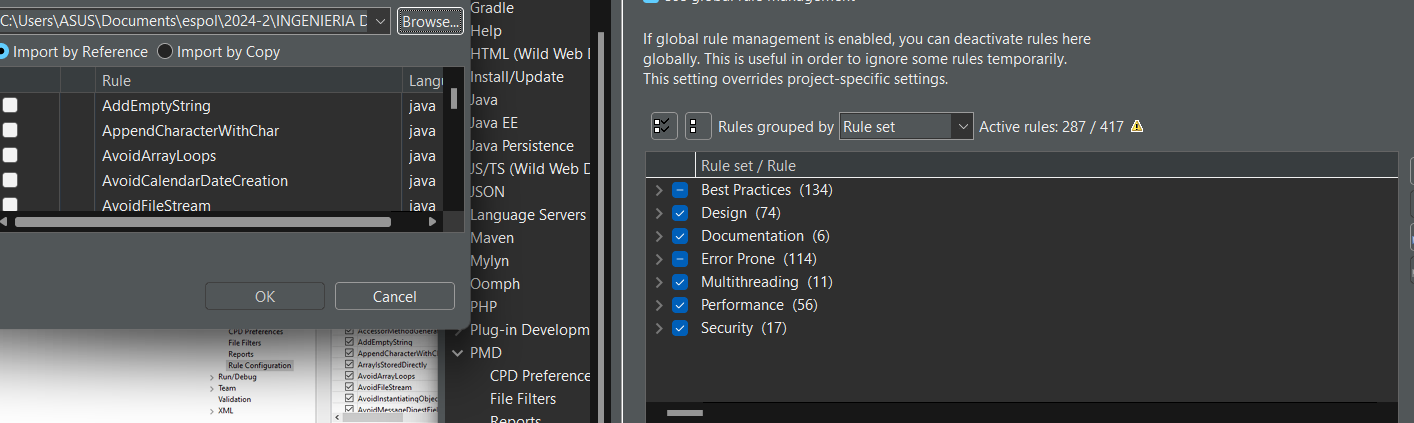
Creation xml



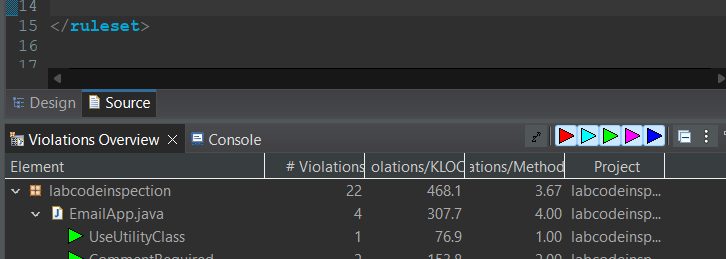
Deleting Ruleset



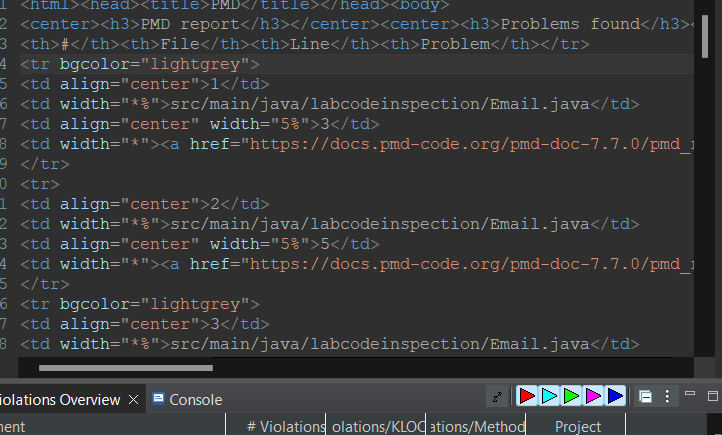
Importing my ruleset



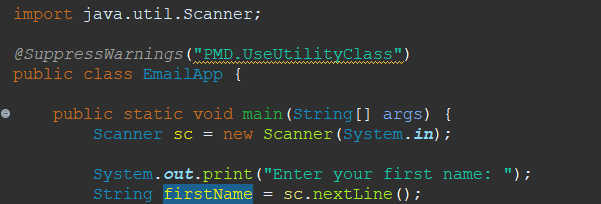
Violations after my ruleset

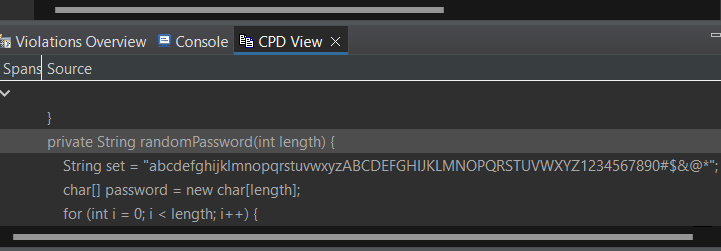


Generated Report

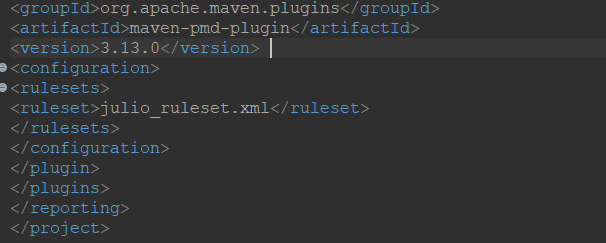


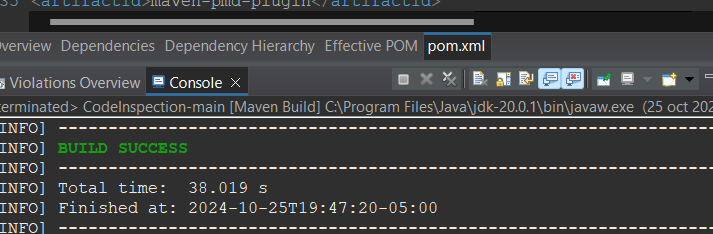
Working on supressing pmd rules



Copy and Paste CPD  


Changing pom.xml



Running project  


URL: https://github.com/jcvivas/tallerCodeIsnpections.git