Analiza statica:

1) CheckStyle

Lista de errori afisate(41):

Error	Util	Actiune
Line is longer than 80 characters (found 100). (42:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 105). (74:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 118). (62:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 125). (22:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 154). (96:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 165). (100:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 168). (106:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 266). (129:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 81). (19:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 86). (38:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 95). (102:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Line is longer than 80 characters (found 95). (108:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere

Line is longer than 80 characters (found 95). (94:0) [LineLength]	Nu	Putem ignora ca e legat doar de lungimea la linie, se asteapta sa nu fie mai lunga de 80 de charactere
Missing a Javadoc comment. (123:5) [MissingJavadocMethod]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (16:5) [JavadocVariable]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (17:5) [JavadocVariable]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (18:5) [JavadocVariable]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (19:5) [JavadocVariable]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (20:5) [JavadocVariable]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (22:5) [MissingJavadocMethod]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (34:5) [MissingJavadocMethod]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (38:5) [MissingJavadocMethod]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (42:5) [MissingJavadocMethod]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (46:5) [MissingJavadocMethod]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (50:5) [MissingJavadocMethod]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (62:5) [MissingJavadocMethod]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Missing a Javadoc comment. (70:5) [MissingJavadocMethod]	Nu	Tine de java standard ca trebuie sa documentam metodele/cimpurile care le expunem
Parameter clientSocket should be final. (70:30) [FinalParameters]	Da	In asa mod putem arata la clientul la metoda noastra ca variabila data nu va fi modificata inauntru la metoda

Parameter content should be final. (62:83) [FinalParameters]	Da	In asa mod putem arata la clientul la metoda noastra ca variabila data nu va fi modificata inauntru la metoda
Parameter contentType should be final. (62:63) [FinalParameters]	Da	In asa mod putem arata la clientul la metoda noastra ca variabila data nu va fi modificata inauntru la metoda
Parameter newPort should be final. (34:37) [FinalParameters]	Da	In asa mod putem arata la clientul la metoda noastra ca variabila data nu va fi modificata inauntru la metoda
Parameter out should be final. (62:30) [FinalParameters]	Da	In asa mod putem arata la clientul la metoda noastra ca variabila data nu va fi modificata inauntru la metoda
Parameter port should be final. (22:22) [FinalParameters]	Da	In asa mod putem arata la clientul la metoda noastra ca variabila data nu va fi modificata inauntru la metoda
Parameter status should be final. (62:48) [FinalParameters]	Da	In asa mod putem arata la clientul la metoda noastra ca variabila data nu va fi modificata inauntru la metoda
Parameter webserverMaintenanceDirectory should be final. (22:63) [FinalParameters]	Da	In asa mod putem arata la clientul la metoda noastra ca variabila data nu va fi modificata inauntru la metoda
Parameter webserverRootDirectory should be final. (22:32) [FinalParameters]	Da	In asa mod putem arata la clientul la metoda noastra ca variabila data nu va fi modificata inauntru la metoda
Parameter webserverStatus should be final. (22:101) [FinalParameters]	Da	In asa mod putem arata la clientul la metoda noastra ca variabila data nu va fi modificata inauntru la metoda
Variable 'handledRequests' must be private and have accessor methods. (16:19) [VisibilityModifier]	Da	Trebuie sa ascundem incapsulam cimpurile clasei si sa expunem prin metoda
Variable 'port' must be private and have accessor methods. (17:16) [VisibilityModifier]	Da	Trebuie sa ascundem incapsulam cimpurile clasei si sa expunem prin metoda
Variable 'webserverMaintenanceDirectory' must be private and have accessor methods. (19:19) [VisibilityModifier]	Da	Trebuie sa ascundem incapsulam cimpurile clasei si sa expunem prin metoda
Variable 'webserverRootDirectory' must be private and have accessor methods. (18:19) [VisibilityModifier]	Da	Trebuie sa ascundem incapsulam cimpurile clasei si sa expunem prin metoda
Variable 'webserverStatus' must be private and have	Da	Trebuie sa ascundem incapsulam cimpurile clasei si sa expunem prin metoda

accessor methods. (20:19) [VisibilityModifier]		

2) SonarLint - All issues(10):

Line	Error	Util	Actiune	Full Sonar description

117	SonarLint: Replace this use of System.out or System.err by a logger.	da	E buna practica sa utilizam logger nu cu sys.our ori sys.err ca e mai flexibil	Standard outputs should not be used directly to log anything
				
				Code smell
				
				Major
				java:S106
				
				When logging a message there are several important requirements which must be fulfilled:
				The user must be able to easily retrieve the logs
				The format of all logged message must be uniform to allow the user to easily read the log
				Logged data must actually be recorded
				Sensitive data must only be logged securely
				If a program directly writes to the standard outputs, there is absolutely no way to comply with those requirements. That's why defining and using a dedicated logger is highly recommended.
				Noncompliant Code Example
				System.out.println("My Message"); // Noncompliant
				Compliant Solution
				logger.log("My Message");
				See
				CERT, ERR02-J. - Prevent exceptions while logging data

119	SonarLint: Replace this use of System.out or System.err by a logger.	da	E buna practica sa utilizam logger nu cu sys.our ori sys.err ca e mai flexibil	Standard outputs should not be used directly to log anything
				
				Code smell
				
				Major
				java:S106
				
				When logging a message there are several important requirements which must be fulfilled:
				The user must be able to easily retrieve the logs
				The format of all logged message must be uniform to allow the user to easily read the log
				Logged data must actually be recorded
				Sensitive data must only be logged securely
				If a program directly writes to the standard outputs, there is absolutely no way to comply with those requirements. That's why defining and using a dedicated logger is highly recommended.
				Noncompliant Code Example
				 System.out.println("My Message"); // Noncompliant
				Compliant Solution
				 logger.log("My Message");
				See
				CERT, ERR02-J. - Prevent exceptions while logging data

130	SonarLint: Replace this use of System.out or System.err by a logger.	da	E buna practica sa utilizam logger nu cu sys.our ori sys.err ca e mai flexibil	Standard outputs should not be used directly to log anything
				
				Code smell
				
				Major
				java:S106
				
				When logging a message there are several important requirements which must be fulfilled:
				The user must be able to easily retrieve the logs
				The format of all logged message must be uniform to allow the user to easily read the log
				Logged data must actually be recorded
				Sensitive data must only be logged securely
				If a program directly writes to the standard outputs, there is absolutely no way to comply with those requirements. That's why defining and using a dedicated logger is highly recommended.
				Noncompliant Code Example
				 System.out.println("My Message"); // Noncompliant
				Compliant Solution
				 logger.log("My Message");
				See
				CERT, ERR02-J. - Prevent exceptions while logging data

16	SonarLint: Make handledRequests a static final constant or non-public and provide accessors if needed.	da	Incapsularea este o practica buna	Class variable fields should not have public accessibility
				
				Code smell
				
				Minor
				java:S1104
				
				Public class variable fields do not respect the encapsulation principle and has three main disadvantages:
				Additional behavior such as validation cannot be added.
				The internal representation is exposed, and cannot be changed afterwards.
				Member values are subject to change from anywhere in the code and may not meet the programmer's assumptions.
				By using private attributes and accessor methods (set and get), unauthorized modifications are prevented.
				Noncompliant Code Example
				 public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked public String firstName; // Noncompliant }
				Compliant Solution
				 public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked private String firstName; // Compliant public String getFirstName() { return firstName; } public void setFirstName(String firstName) { this.firstName = firstName; } }
				Exceptions
				Because they are not modifiable, this rule ignores public final fields. Also, annotated fields, whatever the annotation(s) will be ignored, as annotations are often used by injection frameworks, which in exchange require having public fields.
				See
				MITRE, CWE-493 - Critical Public Variable Without Final Modifier

17	SonarLint: Make handledRequests a static final constant or non-public and provide accessors if needed.	da	Incapsularea este o practica buna	Class variable fields should not have public accessibility
				
				Code smell
				
				Minor
				java:S1104
				
				Public class variable fields do not respect the encapsulation principle and has three main disadvantages:
				Additional behavior such as validation cannot be added.
				The internal representation is exposed, and cannot be changed afterwards.
				Member values are subject to change from anywhere in the code and may not meet the programmer's assumptions.
				By using private attributes and accessor methods (set and get), unauthorized modifications are prevented.
				Noncompliant Code Example
				 public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked public String firstName; // Noncompliant }
				Compliant Solution
				 public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked private String firstName; // Compliant public String getFirstName() { return firstName; } public void setFirstName(String firstName) { this.firstName = firstName; } }
				Exceptions
				Because they are not modifiable, this rule ignores public final fields. Also, annotated fields, whatever the annotation(s) will be ignored, as annotations are often used by injection frameworks, which in exchange require having public fields.
				See
				MITRE, CWE-493 - Critical Public Variable Without Final Modifier

18	SonarLint: Make handledRequests a static final constant or non-public and provide accessors if needed.	da	Incapsularea este o practica buna	Class variable fields should not have public accessibility
				
				Code smell
				
				Minor
				java:S1104
				
				Public class variable fields do not respect the encapsulation principle and has three main disadvantages:
				Additional behavior such as validation cannot be added.
				The internal representation is exposed, and cannot be changed afterwards.
				Member values are subject to change from anywhere in the code and may not meet the programmer's assumptions.
				By using private attributes and accessor methods (set and get), unauthorized modifications are prevented.
				Noncompliant Code Example
				 public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked public String firstName; // Noncompliant }
				Compliant Solution
				 public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked private String firstName; // Compliant public String getFirstName() { return firstName; } public void setFirstName(String firstName) { this.firstName = firstName; } }
				Exceptions
				Because they are not modifiable, this rule ignores public final fields. Also, annotated fields, whatever the annotation(s) will be ignored, as annotations are often used by injection frameworks, which in exchange require having public fields.
				See
				MITRE, CWE-493 - Critical Public Variable Without Final Modifier

19	SonarLint: Make handledRequests a static final constant or non-public and provide accessors if needed.	da	Incapsularea este o practica buna	Class variable fields should not have public accessibility
				
				Code smell
				
				Minor
				java:S1104
				
				Public class variable fields do not respect the encapsulation principle and has three main disadvantages:
				Additional behavior such as validation cannot be added.
				The internal representation is exposed, and cannot be changed afterwards.
				Member values are subject to change from anywhere in the code and may not meet the programmer's assumptions.
				By using private attributes and accessor methods (set and get), unauthorized modifications are prevented.
				Noncompliant Code Example
				 public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked public String firstName; // Noncompliant }
				Compliant Solution
				 public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked private String firstName; // Compliant public String getFirstName() { return firstName; } public void setFirstName(String firstName) { this.firstName = firstName; } }
				Exceptions
				Because they are not modifiable, this rule ignores public final fields. Also, annotated fields, whatever the annotation(s) will be ignored, as annotations are often used by injection frameworks, which in exchange require having public fields.
				See
				MITRE, CWE-493 - Critical Public Variable Without Final Modifier

20	SonarLint: Make handledRequests a static final constant or non-public and provide accessors if needed.	da	Incapsularea este o practica buna	Class variable fields should not have public accessibility
				
				Code smell
				
				Minor
				java:S1104
				
				Public class variable fields do not respect the encapsulation principle and has three main disadvantages:
				Additional behavior such as validation cannot be added.
				The internal representation is exposed, and cannot be changed afterwards.
				Member values are subject to change from anywhere in the code and may not meet the programmer's assumptions.
				By using private attributes and accessor methods (set and get), unauthorized modifications are prevented.
				Noncompliant Code Example
				 public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked public String firstName; // Noncompliant }
				Compliant Solution
				 public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked private String firstName; // Compliant public String getFirstName() { return firstName; } public void setFirstName(String firstName) { this.firstName = firstName; } }
				Exceptions
				Because they are not modifiable, this rule ignores public final fields. Also, annotated fields, whatever the annotation(s) will be ignored, as annotations are often used by injection frameworks, which in exchange require having public fields.
				See
				MITRE, CWE-493 - Critical Public Variable Without Final Modifier

70	SonarLint: Refactor this method to reduce its Cognitive Complexity from 22 to the 15 allowed.	da	Este greu de procesat cu ochii o metoda care este lunga si are multe inner ifs	Cognitive Complexity of methods should not be too high Code smell Critical java:S3776 Cognitive Complexity is a measure of how hard the control flow of a method is to understand. Methods with high Cognitive Complexity will be difficult to maintain. See Cognitive Complexity Parameters Following parameter values can be set in Rule Settings. In connected mode, server side configuration overrides local settings. Threshold The maximum authorized complexity.

				Current value: 15 Default value: 15

94	SonarLint: Define a constant instead of duplicating this literal "200 OK" 3 times.	da	Nu e bine de avut tare multe duplicate literale in cod	String literals should not be duplicated
				
				Code smell
				
				Critical
				java:S1192
				
				Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences.
				On the other hand, constants can be referenced from many places, but only need to be updated in a single place.
				Noncompliant Code Example
				With the default threshold of 3:
				 public void run() { prepare("action1"); // Noncompliant - "action1" is duplicated 3 times execute("action1"); release("action1"); } @SuppressWarning("all") // Compliant - annotations are excluded private void method1() { /* ... */ } @SuppressWarning("all") private void method2() { /* ... */ } public String method3(String a) { System.out.println("'" + a + "'"); // Compliant - literal "'" has less than 5 characters and is excluded return ""; // Compliant - literal "" has less than 5 characters and is excluded }
				Compliant Solution
				 private static final String ACTION_1 = "action1"; // Compliant public void run() { prepare(ACTION_1); // Compliant execute(ACTION_1); release(ACTION_1); }
				Exceptions
				To prevent generating some false-positives, literals having less than 5 characters are excluded.
				Parameters
				Following parameter values can be set in Rule Settings. In connected mode, server side configuration overrides local settings.
				threshold
				Number of times a literal must be duplicated to trigger an issue

				Current value: 3 Default value: 3