

Google Summer of Code 2025

Proposal for Checkstyle

Project: Upgrade dependencies to use latest version of libraries and use their features



Name	Amit Kumar Deoghoria
GitHub	AmitKumarDeoghoria
LinkedIn	AmitKumarDeoghoria
Email	Kamit5741@gmail.com
Time Zone	Indian Standard Time , UTC +5:30
Project Size	Medium
Mentors	Roman Ivanov

Table Of Content

Content	Page No.
About me	3
Project Details	4
Technical Details and Step by Step Implementation	5-15
Project Timeline	16
Plans & expectations	17
Contribution In relation to this project	18
Contribution Highlights	19
Why Me?	20
Future Work	21

About Me

Amit Kumar Deoghoria

Final-Year Student, BITS Pilani

B.E. (Electronics & Communication) & M.Sc. (Chemistry)

Technical Expertise

- **Languages:** Java, C++, Python
- **Frameworks & Tools:** SpringBoot, SQL, Docker, CI/CD, Kafka, Prometheus, Grafana, Git, Bash, Groovy, Junit, Mongo, SQL, Ant
- **Achievements :** 700+ Problems Solved in *LeetCode*, 6-star Hacker Rank, Achieved **Global Rank 2** in *Codechef* Contest.

Internships & Experience

◆ **Goldman Sachs (Summer Analyst)**

- Migrated multiple flows to a strategic architecture focusing on scalability, cost efficient, reliability, performance & monitoring
- Developed scripts & unit test cases for comprehensive coverage

◆ **Hyphen SCS (Software Engineering Intern)**

- Optimized logistics operations using advanced graph algorithms

Open-Source Contributions (Checkstyle)

✓ **80+ PRs merged since October 2024** ([Link](#))

✓ **Strong understanding of:**

Checkstyle architecture, Mutation testing, Maven plugin, Code coverage
Test case development, Dependency management, PIT test suppressions

✓ **Issues Solved:** Fixed test execution failures, optimized dependencies, enabled example tests, resolved PIT test suppressions, Regression tests.

Why Checkstyle?

- ✓ Beginner-friendly community with strong mentorship.
- ✓ Well-documented codebase for seamless contributions.
- ✓ Deepened understanding of static analysis tools & clean coding.

Project Details

Description

Currently, Checkstyle is using JDK 11, while the rest of the world has moved to JDK 21. This project aims to incrementally upgrade the minimum JDK version to JDK 17 first and then to JDK 21. Alongside this upgrade, dependencies such as Apache Doxia and others will be updated to their latest versions. This will enhance the project's maintainability, performance, and access to newer features.

Deliverables

- Upgrade JDK dependency to 17 and subsequently to 21.
- Update Apache Doxia and other necessary dependencies.
- Upgrade IntelliJ IDEA inspection configurations.
- Evaluate and implement BOMs (Bill of Materials) for dependency management.
- Activate the following Checkstyle checks:
 - UnusedLambdaParameterShouldBeUnnamed
 - UnusedCatchParameterShouldBeUnnamed
 - SealedShouldHavePermitsList
 - WhenShouldBeUsed
 - MissingNullCaseInSwitch
- Integrate the Maven Wrapper for build consistency.
- Migrate the codebase to utilize the new NIO JDK API ([Issue #16155](#)).
- Implement module-info.java for modularization ([Initial Activity](#)).

This upgrade will modernize the Checkstyle project, ensuring compatibility with the latest Java features and best practices while improving long-term maintainability.

Technical Details & Step-by-Step Implementation

Step 1: Upgrade JDK to 17

- **Modify Build Configuration: Update pom.xml:**

```
<properties>
  <!-- Current -->
  <java.version>11</java.version>
  <maven.compiler.release>${java.version}</maven.compiler.release>

  <!-- Updated for JDK 17 -->
  <java.version>17</java.version>
  <maven.compiler.release>17</maven.compiler.release>
</properties>
```

- **Identify and Replace Deprecated APIs:**
 - Use `jdeprscan` tool to analyze deprecated APIs:

```
jdeprscan --release 17 --class-path target/classes
```

- Replace deprecated constructs with modern equivalents.
- **Run Tests and Refactor Failing Code:** Execute `mvn clean test` and fix any compatibility issues. Ensure proper handling of new Java features such as pattern matching and records.

Possible Compatibility Issues and their Resolution

a. Deprecated or Removed APIs Symptoms:

NoSuchMethodError, ClassNotFoundException, or test failures due to deprecated APIs (e.g., java.util.Date, Vector).

Solution: Use `jdeprscan --release 17 --class-path target/classes` to identify deprecated APIs. Replace deprecated classes with modern equivalents (e.g., ArrayList instead of Vector, java.time.LocalDate instead of java.util.Date).

b. Module System (JPMS) Conflicts Symptoms:

IllegalAccessError or reflection failures due to stricter encapsulation in Java modules.

Solution: Add `--add-opens` or `--add-exports` JVM args in `maven-surefire-plugin.xml`

c. Incompatible Libraries Symptoms:

Dependency clashes (e.g., older versions of Apache Commons or Guava).

Solution: Update dependencies in `pom.xml` to versions compatible with Java 17

d. Deprecated Checks/Properties Symptoms:

Checkstyle warnings/errors for removed rules (e.g., Translation replaced by IllegalTokenText)

Solution: Update `checkstyle.xml` to use modern equivalents

e. New Syntax (Records/Pattern Matching) Symptoms:

False positives or missed violations for records.

Solution: Upgrade Checkstyle (supports Java 17 syntax). Configure checks like JavadocType to handle records:

Step 2: Upgrade Dependencies (Apache Doxia & Others)

```
<properties>
  <!-- Current -->
  <doxia.version>1.12.0</doxia.version>

  <!-- Updated -->
  <doxia.version>2.0.0</doxia.version>
</properties>

<!-- Remove obsolete exclusions for Doxia (if no longer needed) -->
<dependency>
  <groupId>org.apache.maven.doxia</groupId>
  <artifactId>doxia-core</artifactId>
  <version>${doxia.version}</version>
  <exclusions>
    <!-- Remove if no longer required in 2.0.0 -->
    <exclusion>
      <groupId>commons-codec</groupId>
      <artifactId>commons-codec</artifactId>
    </exclusion>
    <exclusion>
      <groupId>com.google.collections</groupId>
      <artifactId>google-collections</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

- Upgrade Apache Doxia and check for API changes.
- Run dependency analysis to resolve version conflicts:

```
mvn dependency:tree
```

- Test Application Stability After Upgrades: Run mvn verify and update tests if necessary.
- Update plugins (e.g., maven-site-plugin) to match upgraded dependencies.
- Ensure cross-module dependency versions align (e.g., parent POM <dependencyManagement>).
- Add tests for features reliant on upgraded dependencies (e.g., report generation with Doxia).

Step 3: Add BOM for Dependency Management

- Adding BOM will ensure: version consistency, Simplified Dependency Declaration, Easier Upgrades, and Compatibility Guarantees.

I have already worked on using Use bom artifact for junit version management in Checkstyle Issue: [#16396](#)

- [#16484](#) Has been merged successfully In checkstyle
- Added JUnit BOM in <dependencyManagement>

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.junit</groupId>
      <artifactId>junit-bom</artifactId>
      <version>${junit.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

- Removed Hardcoded Versions from JUnit Dependencies.

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>${junit.version}</version> <!-- Explicit version -->
</dependency>
```

- Resolved Duplicate junit.version Property
- Resolved Duplicate junit.version Property
- Retained Non-JUnit Dependency Versions

Step 4: Upgrade to JDK 21

- Modify Build Configuration:

```
<properties>
  <maven.compiler.source>21</maven.compiler.source>
  <maven.compiler.target>21</maven.compiler.target>
</properties>
```

- Refactor Code for JDK 21 Features:

- Migrate sealed classes where applicable.

```
<module name="TreeWalker">
  <!-- Allow JDK 21 features -->
  <module name="SuppressionXpathSingleFilter">
    <property name="checks" value=".*"/>
    <property name="query" value="//CLASS_DEF[./IDENT[@text='sealed' or @text='non-sealed']]"/>
  </module>

  <!-- Configure rules for modern Java -->
  <module name="JavadocMethod">
    <property name="allowMissingParamTags" value="true"/> <!-- Optional -->
  </module>

  <!-- Allow 'var' for local variables (common in virtual threads) -->
  <module name="LocalVariableType">
    <property name="validateAbstractClassNames" value="true"/>
    <property name="tokens" value="VARIABLE_DEF"/>
  </module>
</module>
```

- Replace traditional thread management with virtual threads.

```
try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
  executor.submit(() -> System.out.println("Running in a virtual thread"));
}
```

- Run Full Test Suite to Identify Issues and Fix Failures.
- Problem: Checkstyle fails to parse new syntax (e.g., sealed classes).

Fix: Upgrade Checkstyle to the latest version ($\geq 10.12.5$) for JDK 21 compatibility.

- Problem: Warnings about var in lambda expressions.

Fix: Configure LocalVariableType to allow var in virtual thread contexts.

- Ensure All Dependencies Use BOM Versions to Avoid Conflicts.
- Create custom Checkstyle rules (e.g., via XPath) to enforce project-specific JDK 21 patterns

```
<!-- Example: Ban synchronized blocks in virtual threads -->
<module name="Regexp">
  <property name="format" value="synchronized\s*\("/>
  <property name="message" value="Avoid 'synchronized' with virtual threads; use ReentrantLock instead."/>
  <property name="severity" value="error"/>
</module>
```

- Upgrade SpotBug/PMD accordingly to detect concurrency issues.
- Update your project's CONTRIBUTING.md to include JDK 21-specific guidelines enforced by Checkstyle

Step 5: Activate New Checkstyle Checks

- **Modify checkstyle.xml to Enable New Rules**
 - Update your Checkstyle configuration file to include the new checks. These rules target common pitfalls in modern Java code, such as unused parameters, sealed class validation, and missing null cases in switch expressions.

```
<module name="Checker">
  <module name="TreeWalker">
    <!-- New Checks for JDK 21+ Features -->
    <module name="UnusedLambdaParameterShouldBeUnnamed"/>
    <module name="UnusedCatchParameterShouldBeUnnamed"/>
    <module name="SealedShouldHavePermitsList"/>
    <module name="WhenShouldBeUsed"/>
    <module name="MissingNullCaseInSwitch"/>
  </module>
</module>
```

- **Fix Existing Code to Pass New Checks**
 - **Example 1: Unused Lambda/Catch Parameters**

```
// Replace unused 'x' with '_'
BiConsumer<String, Integer> consumer = (_, y) -> System.out.println(y);

// Rename 'e' to '_'
try { ... } catch (IOException _) { /* Ignore */ }
```

- **Example 2: Sealed Classes**

```
public sealed class Shape permits Circle, Square { ... }
```

- **Example 3: Null Handling in Switch**

```
switch (obj) {  
    case String s -> System.out.println(s);  
    case null -> System.out.println("Null encountered");  
    default -> {}  
}
```

- **Example 4: Pattern Matching with when**

```
switch (obj) {  
    case String s when !s.isEmpty() -> ... // Add guard clause  
    case String s -> ... // Handle empty strings  
}
```

- **Test the New Checks Run a full codebase scan and address violations**

- **Common Issues & Solutions:**

- **False Positives:** Suppress specific checks using `@SuppressWarnings("checkstyle:RuleName")`.
- **Legacy Code:** Gradually fix violations by enabling one check at a time.

- **Update Documentation**

Step 6: Integrate Maven Wrapper

- **Generate Maven Wrapper Scripts**

```
mvn -N io.takari:maven:wrapper
```

- **Verify the Generated Files**

```
your-project/
├── .mvn/
│   └── wrapper/
│       ├── maven-wrapper.jar
│       └── maven-wrapper.properties
├── mvnw
├── mvnw.cmd
└── pom.xml
```

- **Test the Maven Wrapper**

```
Apache Maven 3.9.8 (...)
Maven home: /home/user/.m2/wrapper/dists/...
Java version: 21.0.3, vendor: Oracle Corporation
```

- **Update your CI scripts (e.g., GitHub Actions) to use the wrapper instead of system Maven**

```
# GitHub Actions Example
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Build with Maven Wrapper
        run: ./mvnw clean verify
```

Step 7: Migrate to NIO API[\(Issue #16155\)](#)

- **Revert Modernizer Exclusions:** Modernizer will now report violations

```
<!-- Before: Commented exclusions -->
<!-- <exclusionPattern>java/io/File.*</exclusionPattern> -->
<!-- <exclusionPattern>java/nio/file/Paths.get:.*</exclusionPattern> -->

<!-- After: Remove exclusions to detect violations -->
<!-- Exclusion for Paths.get is removed to allow enforcement -->
```

- **Suppress Violations in Public Apis:** For methods where File usage cannot be immediately removed
- **Fix Detected Violations :**

```
allFiles.add(Paths.get(fileName).toFile()); // Interim fix (preserve File usage)
// OR (preferred if API allows):
allFiles.add(Paths.get(fileName)); // Requires changing variable type to Path
```

- **Enforce Modernizer in CI/CD**
- **Address Specific Violations:**
 - **Case1: CheckstyleAntTask.java**

```
allFiles.add(Paths.get(fileName).toFile()); // Interim solution
```

- **Case 2: SiteUtil.java**

```
Paths.get(...); // Align with Checkstyle's existing conventions
```

Step 8: Implement module-info.java for Modularization

- **Export Only Public API Packages**

```
module com.puppycrawl.tools.checkstyle {  
    // Public API  
    exports com.puppycrawl.tools.checkstyle.api;  
    exports com.puppycrawl.tools.checkstyle.checks;  
  
    // Internal packages (not exported)  
    // com.puppycrawl.tools.checkstyle.internal is hidden  
}
```

- **Restrict Reflective Access**







```
module com.puppycrawl.tools.checkstyle {  
    exports com.puppycrawl.tools.checkstyle.api;  
  
    // Selective reflective access for testing  
    opens com.puppycrawl.tools.checkstyle.internal to junit;  
}
```

- **Address Non-Modular Dependencies:** Third-party dependencies are not modularized, causing warnings.

```
requires transitive java.xml;  
requires static lombok;           // Optional dependency  
requires automatic.module.name;  // Non-modular dependency
```

- **Fix Test Configuration**

Project Timeline

Weeks	Tasks	Tasks to be completed
-		Community Bonding Period
1-2	 Upgrade to JDK 17	<ul style="list-style-type: none"> - Update build scripts (Maven/Gradle) to JDK 17. - Resolve compiler warnings and initial compatibility issues. - Verify CI/CD pipeline stability.
3-4	 Upgrade Dependencies	<ul style="list-style-type: none"> - Modernize critical dependencies (Apache Doxia, PMD, Checker Framework). - Test backward compatibility. - Address deprecated API warnings.
5- 6	 Migrate to JDK 21	<ul style="list-style-type: none"> - Update JDK version in build configs. - Refactor code for JDK 21 features (sealed classes, virtual threads). - Fix test failures post-upgrade.
7- 8	 Implement BOM & Enable Checks	<ul style="list-style-type: none"> - Centralize dependencies via BOM. - Activate new Checkstyle rule, Fix codebase violations.
9 - 10	 Integrate Maven Wrapper & NIO API	<ul style="list-style-type: none"> - Generate mvnw scripts. - Replace java.io.File with java.nio.Path. - Optimize I/O performance.
11-12	 Modularize & Finalize	<ul style="list-style-type: none"> - Design module-info with restricted exports. - Test encapsulation and reflection. - Update docs and release notes.

Plans and Expected Collisions

Different sections of the project require varying amounts of time, so I have allocated different numbers of days to each based on their complexity and requirements.

I plan to dedicate **35-40 hours per week** to ensure steady progress and complete the project on time. However, I might have **comprehensive Exams** during the GSoC period, during which I will be able to contribute **only 2-3 hours per day**. I will plan my work accordingly to prevent delays. Since the exam dates are not announced yet, I will adjust my schedule as needed.

During the **community bonding period**, I will clarify any uncertainties with the maintainers regarding project requirements, new functionalities, and the best approach to implement them. I will also discuss strategies for transforming style guide tests to a chapter-wise testing approach. My goal is to start sending PRs as early as possible to compensate for the time lost during exams.

Expectations from Mentors

Since this project involves upgradation and involve lot of core modules, I will need the maintainers' guidance and insights, especially if I encounter challenges.

I would appreciate timely reviews and constructive feedback on new implementations, ensuring that the work progresses efficiently and aligns with the project's overall goals.

Contributions In Relation to this Project

1. Dependency Upgrades & Compatibility Fixes

- **Bumped Checker Framework from 3.48.4 to 3.49.1**
 - **PR:** [#16481](#)
 - **Impact:** Ensured compatibility, enhanced type-checking and static analysis.
 - **Status:** Merged and verified in Checkstyle's CI.
- **Upgraded PMD from 7.10.0 to 7.11.0**
 - **PR:** [#16457](#)
 - **Impact:** Improved static code analysis
 - **Status:** Merged with successful test execution.
- **JUnit Version Management via BOM**
 - **Issue:** [#16396](#)
 - **PR:** [#16484](#)
 - **Impact:** Centralized JUnit dependencies to avoid version conflicts during test suite execution.
 - **Status:** Merged

2. Critical Issue Resolutions

- **Fixed Test Failures on Non-EN Locales**
 - **Issue:** [#16233](#)
 - **PR:** [#16500](#)
 - **Impact:** Ensured test reliability across global locales, critical for CI/CD pipelines.
 - **Status:** Merged with 67/67 checks approved.
- **CI Integration Tests for Breaking Changes**
 - **Issue:** [#6784](#)
 - **PR:** [#16510](#)
 - **Impact:** Added automated checks to detect compatibility regressions.
 - **Status:** Merged with 68/68 checks approved

Contribution Highlights([Total 80+ PRs:\(Link\)](#))

Sr	Pull Request	Link
1.	Issue #6784: Create integration tests in CI to alarm on breaking compatibility	Link
2.	Issue #16233: test execution is failing on non EN locales	Link
3.	dependency: bump checkerframework.version from 3.48.4 to 3.49.1	Link
4.	dependency: bump pmd.version from 7.10.0 to 7.11.0	Link
5.	dependency: bump checkerframework.version from 3.48.4 to 3.49.1	Link
6.	Issue #13999: Resolve pitest suppression for skipHtmlComment()	Link
7.	Issue #11163: Enforce file size on CovariantEqualsCheck	Link
8.	Issue #13345: Enable examples tests for DescendantTokenCheck	Link
9.	Issue #16157: Populated Empty Cells in Active tool	Link
10.	Issue #13345: Enable examples tests for MatchXPathCheck	Link

Why Me?

- ✓ **Experienced Contributor** – I have been contributing to Checkstyle since **October 2024** and have merged **80+ PRs**, showcasing my deep engagement with the project.
- ✓ **Strong Codebase Understanding** – With **6+ months** of active contributions, I have gained an in-depth understanding of **Checkstyle's architecture, coding standards, and workflows**.
- ✓ **Direct Contributions to Upgradation Project** – I have already made **significant contributions** to the ongoing upgradation project, making me well-versed in the necessary improvements and challenges.
- ✓ **Robust Technical Skills** – My **expertise in Java, static code analysis, and software development** ensures that I can efficiently implement changes and enhance Checkstyle's functionality.
- ✓ **Proven Problem-Solving Ability** – My contributions demonstrate my capability to **write clean, efficient code** and debug complex issues, making me a reliable candidate for this project.
- ✓ **Commitment to the Project** – My **consistent involvement** and collaboration with the Checkstyle community reflect my dedication to improving the tool and ensuring high-quality contributions.

With my experience, technical expertise, and deep involvement in Checkstyle, I am confident that I can make a meaningful impact through this GSoC project! 🚀

Future Work

✓ **Continued Contributions** – After GSoC, I will remain an **active contributor** to Checkstyle, making high-quality PRs and enhancing the project.

✓ **Helping New Contributors** – I will **assist newcomers**, guide them through the contribution process, and help resolve their doubts to grow the community.

✓ **Expanding My Knowledge** – My journey with Checkstyle has strengthened my understanding of **Java, CI/CD, Mutation Testing, Code Coverage, and Open-Source development**. I aim to **further deepen** my expertise in these areas.

✓ **Innovating with New Features** – I have **thought of new feature ideas** to enhance Checkstyle's functionality and user experience. I look forward to **discussing and implementing** them with the maintainers.

✓ **Long-Term Commitment** – My experience with Checkstyle has been invaluable, and I am excited to **keep contributing, improving the tool, and collaborating** with the community in the long run. 🚀

Thank You 😊