Java / Checkstyle

For Java, Checkstyle is used, which is a static code analysis tool that helps programmers write Java code that adheres to a coding standard. For this class, *the coding standard is provided* in the form of a Checkstyle configuration file. It can be downloaded directly from github, and is based off of the one used in CS 1332, Data Structures & Algorithms. More details on each check can be found in the official Checkstyle documentation.

In addition, there are some examples of compliant/noncompliant example code provided:

- `CalculatorWindow.java` (non-compiliant) - example Swing application with poor formatting
- `CalculatorWindowFormatted.java` (compliant) - same as above, plus it passes all Checkstyle checks

As with pylint, there are a few different ways of running Checkstyle, each of which are detailed below.

Running Checkstyle with Script *(Recommended)*

A utility script was developed by the CS 2340 TAs to make running Checkstyle on your projects easier.

Prerequisites

- Java installed and on the `PATH` (tutorial)
- Python **3** installed and on the `PATH` (tutorial on Canvas... complete the "Installing Python" section)
- Checkstyle script downloaded

Running

The `run_checkstyle.py` script supports running checkstyle over every java file in a directory. For example, if my project structure is as follows:

```
project/
├── src/
│   ├── Application.java
│   ├── SpringUtilities.java
│   └── CalculatorWindow.java
└── run_checkstyle.py
```

then I can run checkstyle on every `.java` file in the project directory:

```
python run_checkstyle.py
```

This will output something like the following:

```
~/project/ $ python run_checkstyle.py

Downloading cs2340_checks.xml
100.0% 4873 / 4873

Downloading checkstyle-8.24-all.jar
100.0% 11625142 / 11625142

> Note: The checkstyle jar has been downloaded and a gitignore has automatically been created
        for you at the project root. This file should be checked into version control.

Running Checkstyle on 3 files:

Starting audit...
[WARN] ~/project/src/CalculatorWindow.java:29:43: ',' is preceded with whitespace.
    [NoWhitespaceBefore]
[WARN] ~/project/src/CalculatorWindow.java:41:44: '{' is not preceded with whitespace.
    [WhitespaceAround]
[WARN] ~/project/src/CalculatorWindow.java:52:22: Name 'CONSTRUCT_LAYOUT' must match pattern
    '^[a-z][a-zA-Z0-9]*$'. [MethodName]
[WARN] ~/project/src/CalculatorWindow.java:55: Line is longer than 100 characters (found 119).
    [LineLength]
... other errors ...
Audit done.

-----------------------------------------------------------
Your code has been rated at 6.73/10 [raw score: 6.73/10]
```

As is shown in the example output, the script will automatically download the checkstyle jar and the code style XML config file and place them in the same folder as the `run_checkstyle.py` script.

> **Note** After this is run, the script will also modify or create a new `.gitignore` file that includes a rule for git to ignore the checkstyle jar file. This means that the jar will not appear to git in the working tree and cannot be staged/committed. **You should commit this file**, as it is considered bad practice to include compiled binaries/jars in git repositories.

The score given at the bottom is a metric of overall code quality, and is computed using the following formula where `e` is the number of Checkstyle errors/warnings and `n` is the number of Java statements in the scanned code:

$$\mathcal{S}(e,n) = 10\left(1 - \frac{5e}{n}\right)$$

Running over another directory

To run the script on a directory other than the current working directory, specify a relative or absolute path using `--root path/to/folder`:

```
python run_checkstyle.py --root path/to/folder
```

Running Checkstyle Directly

Prerequisites

- Java installed and on the `PATH` (tutorial)
- checkstyle jar downloaded and in the same directory as your code
- configuration file downloaded and in the same directory as your code

> **Note:** the given configuration file has only been tested with the latest version of Checkstyle as of writing this (8.24), so it might break with older versions

Running

Once the required files are present, run the following command, adding each file to the end as necessary:

```
java -jar checkstyle-8.24-all.jar -c cs2340_checks.xm file1.java file2.java
```

The program should output something similar to the following, where each violation of Checkstyle is listed, along with its filename, line number, and column number (where applicable):

```
$ java -jar checkstyle-8.24-all.jar -c cs2340_checks.xml Application.java CalculatorWindow.java
Starting audit...
[WARN] ~/project/src/CalculatorWindow.java:29:43: ',' is preceded with whitespace.
    [NoWhitespaceBefore]
[WARN] ~/project/src/CalculatorWindow.java:41:44: '{' is not preceded with whitespace.
    [WhitespaceAround]
[WARN] ~/project/src/CalculatorWindow.java:52:22: Name 'CONSTRUCT_LAYOUT' must match pattern
    '^[a-z][a-zA-Z0-9]*$'. [MethodName]
[WARN] ~/project/src/CalculatorWindow.java:55: Line is longer than 100 characters (found 119).
    [LineLength]
... other errors ...
Audit done.
```

Overall, this method is more complex and requires using platform-specific ways of finding every java file in a directory to run Checkstyle on. For those reasons, we recommend using the `run_checkstyle.py` script as detailed above.

Running via IDE/Editor Plugins

Plugins are available for Checkstyle for a variety of different editors/IDEs. Some of the more popular ones are listed below:

- Visual Studio Code - vscode plugin runs checkstyle automatically
  - Make sure to follow the instructions at the link to set the `cs2340_checks.xml` as the Checkstyle configuration file
- Eclipse - Checkstyle integration into Eclipse
- IntelliJ IDEA - JetBrains plugin that adds Checkstyle side pane for realtime and on-demand Checkstyle running

> **Note**: it is recommended to run Checkstyle via the script/directly at least once before submitting each milestone to make sure all errors are caught.