Coding Style (Java)

2110215 - Programming Methodology

Outline

- Format
- Naming (meaningful)
- Conditional
- Class
- Method
- Example

Format

› Printout using monospace font (`Courier`, **`Consolas`**, …)

› Comment (beginning of every file, unobvious steps, …)

› A space before and after an operator, i.e., x + y rather than x+y

› Avoid long statement in one line

Naming (meaningful)

- **class name**: *Singular noun* begins with *uppercase* letter
- **method name**: *Verb* begins with *lowercase* letter
- **variable name**: *Noun* begins with *lowercase* letter.
  - For **boolean** variables, use *isXXX* or *hasXXX*
- **constant**: *Noun* with all *uppercase* letters.
  - **double** PI, MAX_SPEED

Conditional

- `if(booleanVariable == true)` → `if(booleanVariable)`
- `if(booleanVariable == false)` → `if(!booleanVariable)`
- `if-else` can be used instead of serie of `if`'s ?
- use `.equals()` to compare string/reference not ==

Class

- use get/set for private fields
  - Do not use public field
- Prepare constructor(s)
- `Don't forget to write equals(), toString()`

Method

- small, normally should be < 20 lines
  - refactor to other private methods if it is long.
- Make your method perform only one task.
- Avoid duplicated code.

Coding

› avoid magic number, use constant instead

Example - Indent style

Code

```
// preferred.
if (x < 0) {
  negative(x);
} else {
  nonnegative(x);
}

// Not like this.
if (x < 0)
  negative(x);

// Also not like this.
if (x < 0) negative(x);
```

Example - Indent style

Code

```
public class HelloWorld
{
....public void greetUser(int curre
....{
........System.out.print("Good");
........if (currentHour < AFTERNOON)
........{
.............System.out.println(" Morning");
........}
........else if (currentHour < EVENING)
........{
.............System.out.println( "Afternoon");
........}
........else
........{
.............System.out.println("Evening");
........}
....}
}
```

Note: The period char (.) is used to show indentation

This is just another style.

Example - Indent style

Note: The period char (.) is used to show indentation

Code

```
public class HelloWorld {
....public void greetUser(int currentHour)  {
........System.out.print("Good");
........if (currentHour < AFTERNOON) {
............System.out.println("Morning");
........} else if (currentHour < EVENING) {
............System.out.println("Afternoon");
........} else {
............System.out.println("Evening");
........}
....}
}
```

Ctrl+Shift+F will do the indentation for you.

Example

Code

```
// Bad.
final String value =
    otherValue;

// Good.
final String value = otherValue;
```

Don't break up a statement unnecessarily.

Example - Field, class, and method declarations

Code

```
// Bad.
final static private String value;

// Good.
private static final String value;
```

The order does not matter but it is nice to be consistent with most people.

Example - Variable naming

Code

```
// Bad.
//   - Field names give little insight into what
fields are used for.
class User {
  private final int a;
  private final String m;

  ...
}

// Good.
class User {
  private final int ageInYears;
  private final String maidenName;

  ...
}
```

Extremely short variable names should be reserved for instances like loop indices.

Example - Include units in variable names

Code

```
// Bad.
long pollInterval;
int fileSize;

// Good.
long pollIntervalMs;
int fileSizeGb.

// Better.
//   - Unit is built in to the type.
//   - The field is easily adaptable between units,
readability is high.
Amount<Long, Time> pollInterval;
Amount<Integer, Data> fileSize;
```

But it depends on the usage too.

Example - Don't embed metadata in variable names

Code

```
// Bad.
Map<Integer, User> idToUserMap;
String valueString;

// Good.
Map<Integer, User> usersById;
String value;
```

A variable name should describe the variable's purpose. Adding extra information like scope and type is generally a sign of a bad variable name.

Avoid embedding the field type in the field name.

Example

Code

```
// Bad.
String _value;
String mValue;

// Good.
String value;
```

Also avoid embedding scope information in a variable. Hierarchy-based naming suggests that a class is too complex and should be broken apart.

Example - Space pad operators and equals.

Code

```
// Bad.
//   - This offers poor visual separation of operations.
int foo=a+b+1;

// Good.
int foo = a + b + 1;
```

Example - Be explicit about operator precedence

Code

```
// Bad.
return a << 8 * n + 1 | 0xFF;

// Good.
return (a << (8 * n) + 1) | 0xFF;
```

Don't make your reader open the spec to confirm, if you expect a specific operation ordering, make it obvious with parenthesis.

```
if ((values != null) && (10 > values.size())) {
   ...
}
```

Example - Avoid unnecessary code

Code

```
// Bad.
//   - The variable is immediately returned, and just serves to
clutter the code.
List<String> strings = fetchStrings();
return strings;

// Good.
return fetchStrings();
```

Superfluous temporary variables.

Example - Comments

Code

```
/*
 * This is
 * okay.
 */
```

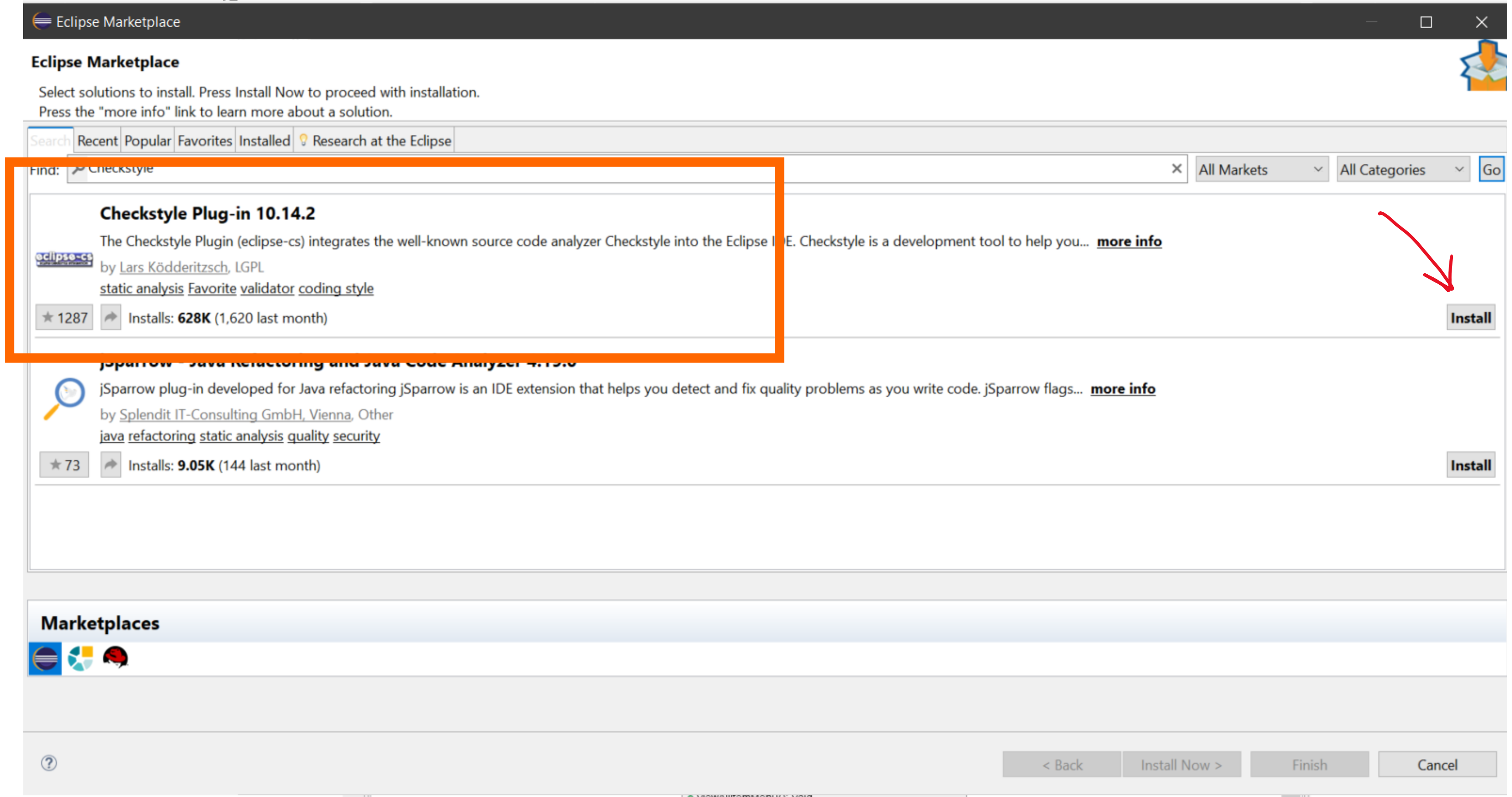
```
// And so
// is this.
```

```
/* Or you can
   even do this. */
```

Tip: When writing multi-line comments, use the /* … */ style if you want automatic code formatters to re-wrap the lines when necessary (paragraph-style). Most formatters don't re-wrap lines in // … style comment blocks.

Forcing good style with CheckStyle Plugin

› Select Eclipse Marketplace from Help Menu. Search for Checkstyle. Then click Install.



Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Checkstyle Plug-in 10.14.2

The Checkstyle Plugin (eclipse-cs) integrates the well-known source code analyzer Checkstyle into the Eclipse IDE. Checkstyle is a development tool to help you... **more info**

by Lars Ködderitzsch, LGPL

static analysis Favorite validator coding style

★ 1287 Installs: **628K** (1,620 last month)

Install

jSparrow - Java Refactoring and Java Code Analyzer 4.19.0

jSparrow plug-in developed for Java refactoring jSparrow is an IDE extension that helps you detect and fix quality problems as you write code. jSparrow flags... **more info**

by Splendit IT-Consulting GmbH, Vienna, Other

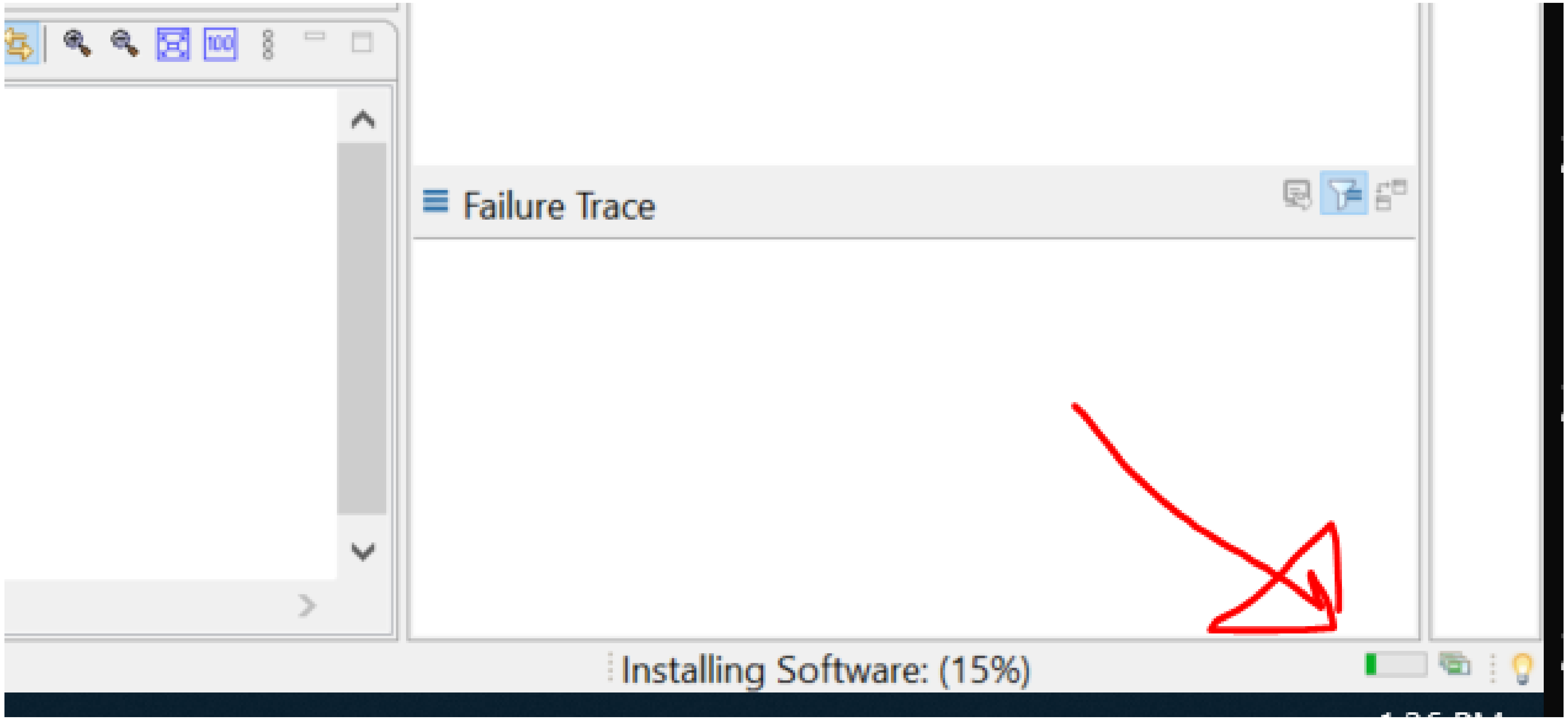
java refactoring static analysis quality security

★ 73 Installs: **9.05K** (144 last month)

Install

Marketplaces

Wait for the installation

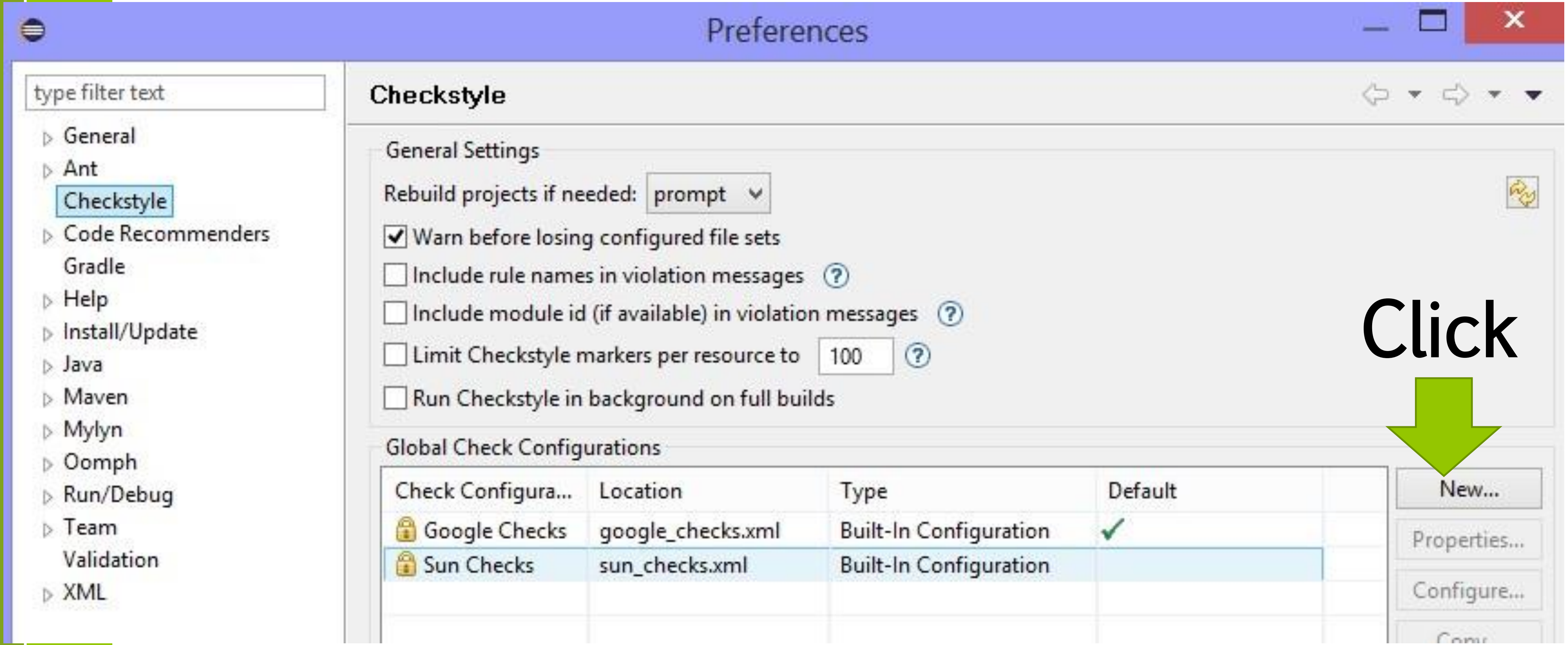

Failure Trace

Installing Software: (15%)

The default style is too much!

- Both google style and Sun style check too many things, you will be frustrated.
- So, Create the style yourself.
  - See next page!!

Windows-> Preferences



General Settings

Rebuild projects if needed: prompt

[✓] Warn before losing configured file sets

[ ] Include rule names in violation messages

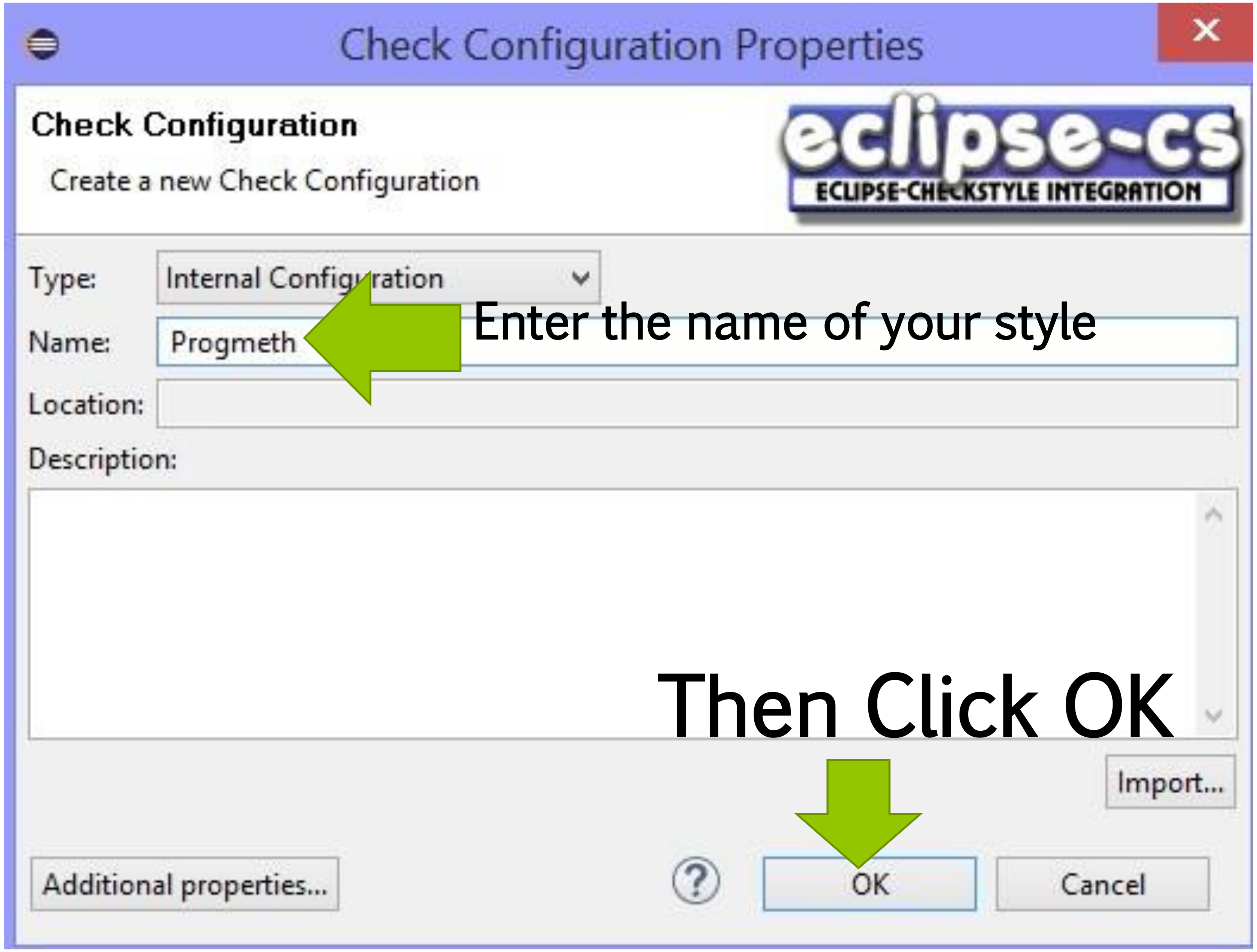
[ ] Include module id (if available) in violation messages

[ ] Limit Checkstyle markers per resource to 100

[ ] Run Checkstyle in background on full builds

Global Check Configurations

Check Configura...	Location	Type	Default
Google Checks	google_checks.xml	Built-In Configuration	✓
Sun Checks	sun_checks.xml	Built-In Configuration	



Check Configuration Properties

Check Configuration

Create a new Check Configuration

Type: Internal Configuration

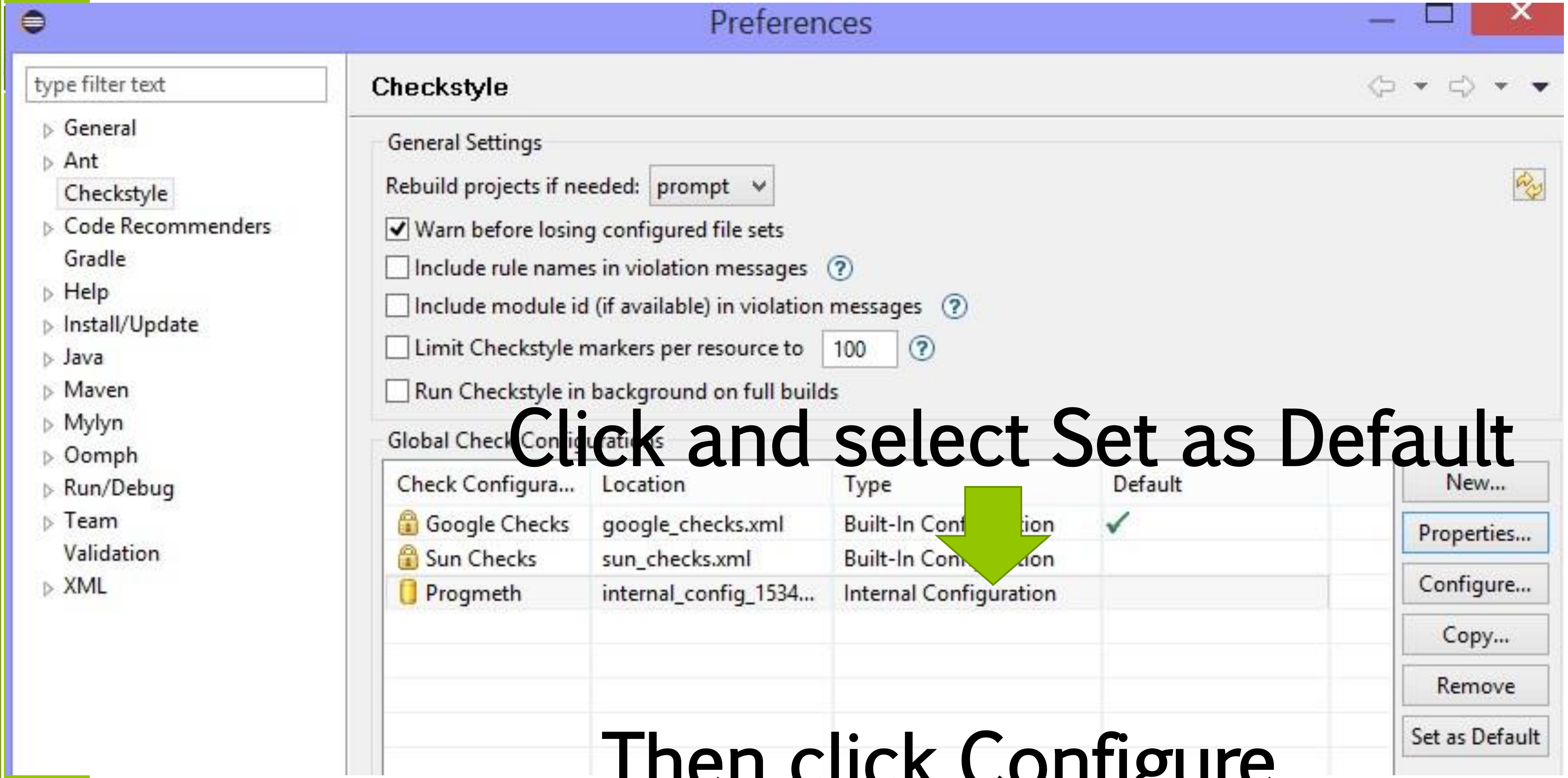
Name: Progmeth

Location:

Description:

Enter the name of your style

Then Click OK


Preferences

type filter text

- General
- Ant
- Checkstyle
- Code Recommenders
- Gradle
- Help
- Install/Update
- Java
- Maven
- Mylyn
- Oomph
- Run/Debug
- Team
- Validation
- XML

Checkstyle

General Settings

Rebuild projects if needed: prompt

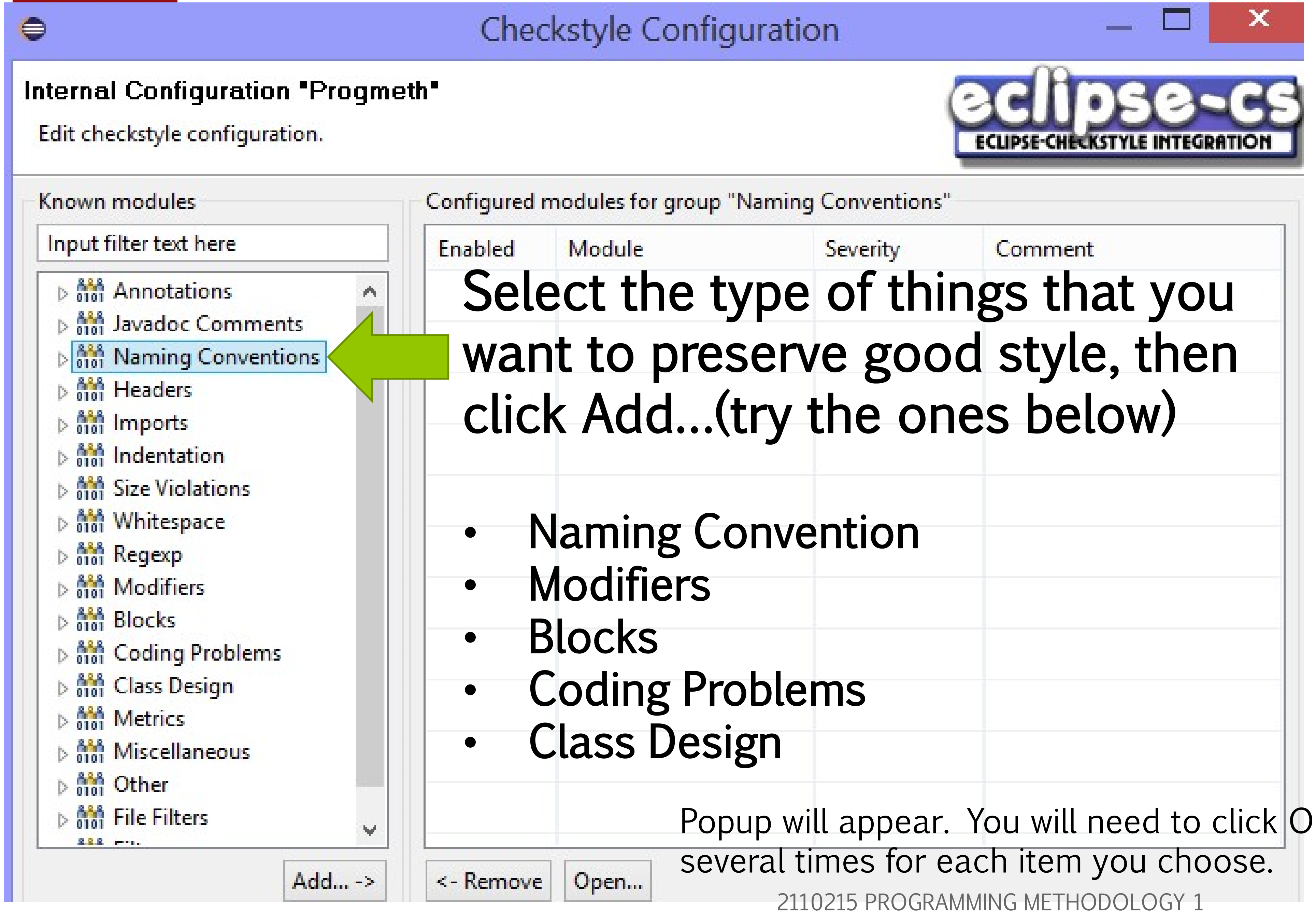
- [✓] Warn before losing configured file sets
- [ ] Include rule names in violation messages
- [ ] Include module id (if available) in violation messages
- [ ] Limit Checkstyle markers per resource to 100
- [ ] Run Checkstyle in background on full builds

Global Check Configurations

Check Configura...	Location	Type	Default
Google Checks	google_checks.xml	Built-In Configuration	✓
Sun Checks	sun_checks.xml	Built-In Configuration	
Progmeth	internal_config_1534...	Internal Configuration	

Click and select Set as Default

Then click Configure...



Checkstyle Configuration

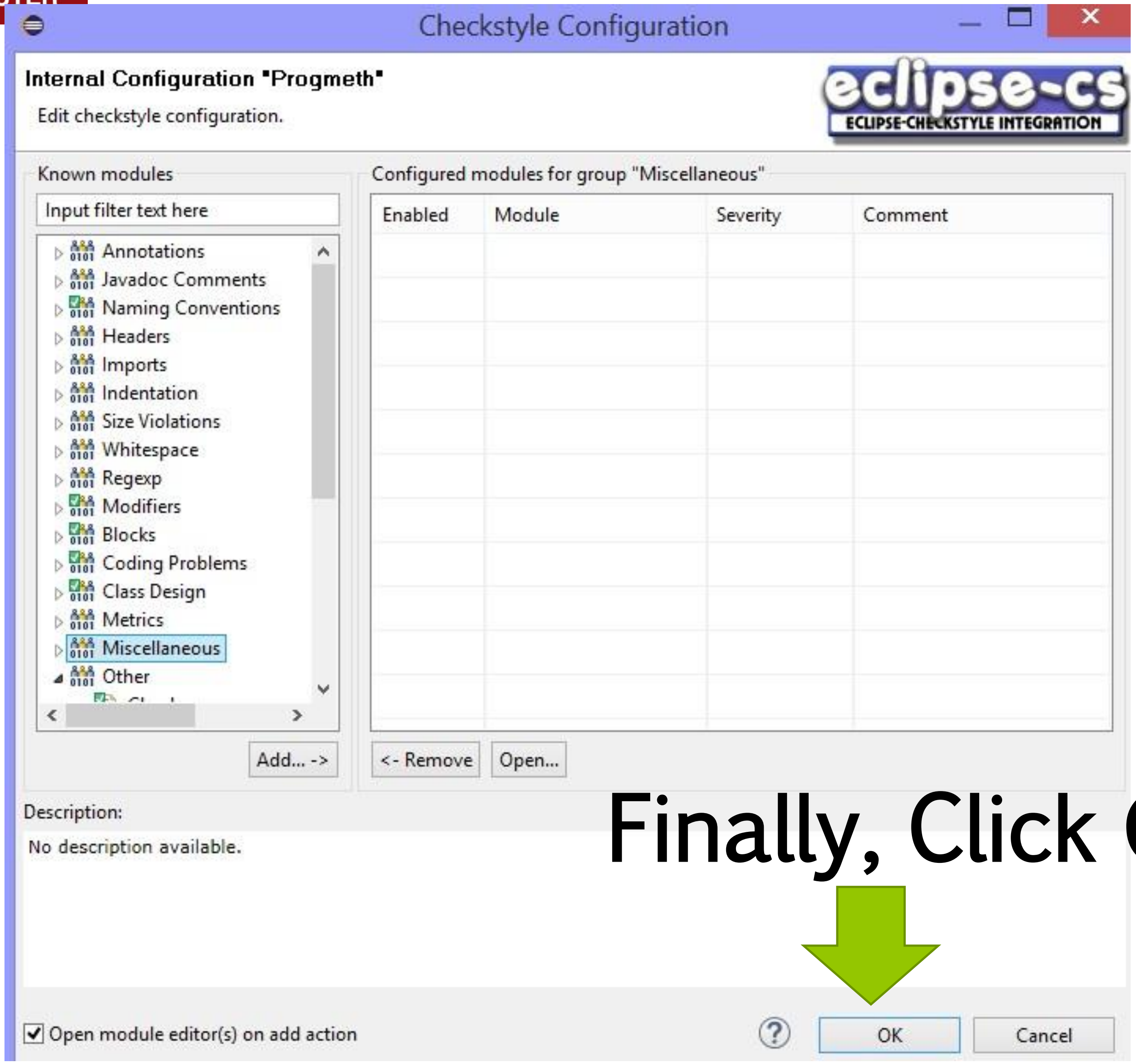
Internal Configuration "Progmeth"

Edit checkstyle configuration.

Select the type of things that you want to preserve good style, then click Add...(try the ones below)

- Naming Convention
- Modifiers
- Blocks
- Coding Problems
- Class Design

Popup will appear. You will need to click OK several times for each item you choose.



Checkstyle Configuration

Internal Configuration "Progmeth"

Edit checkstyle configuration.

Known modules

Input filter text here

- Annotations
- Javadoc Comments
- Naming Conventions
- Headers
- Imports
- Indentation
- Size Violations
- Whitespace
- Regexp
- Modifiers
- Blocks
- Coding Problems
- Class Design
- Metrics
- Miscellaneous
- Other

Configured modules for group "Miscellaneous"

Enabled	Module	Severity	Comment

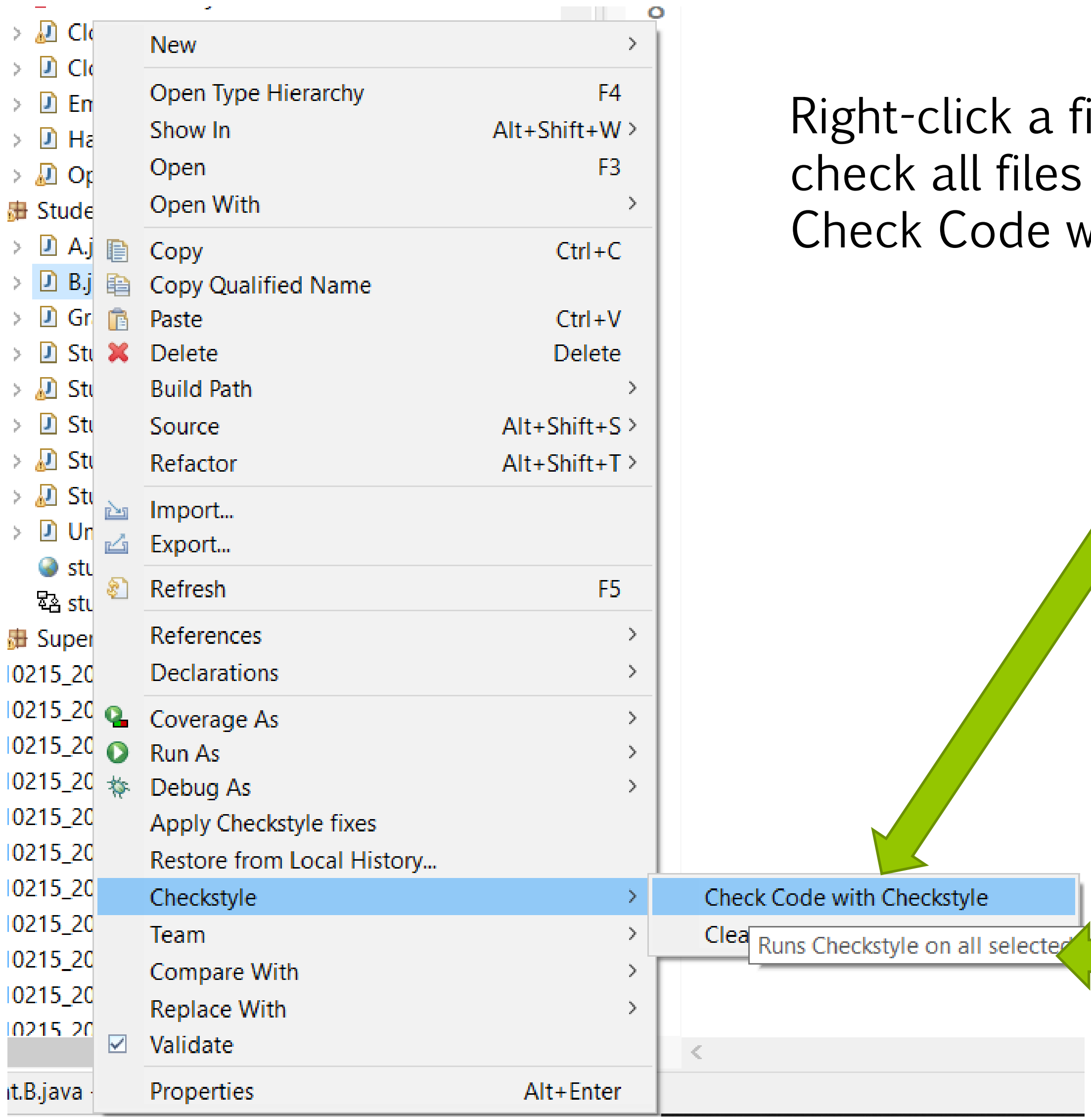
Description:

No description available.

☑ Open module editor(s) on add action

Finally, Click OK.

How to run Checkstyle

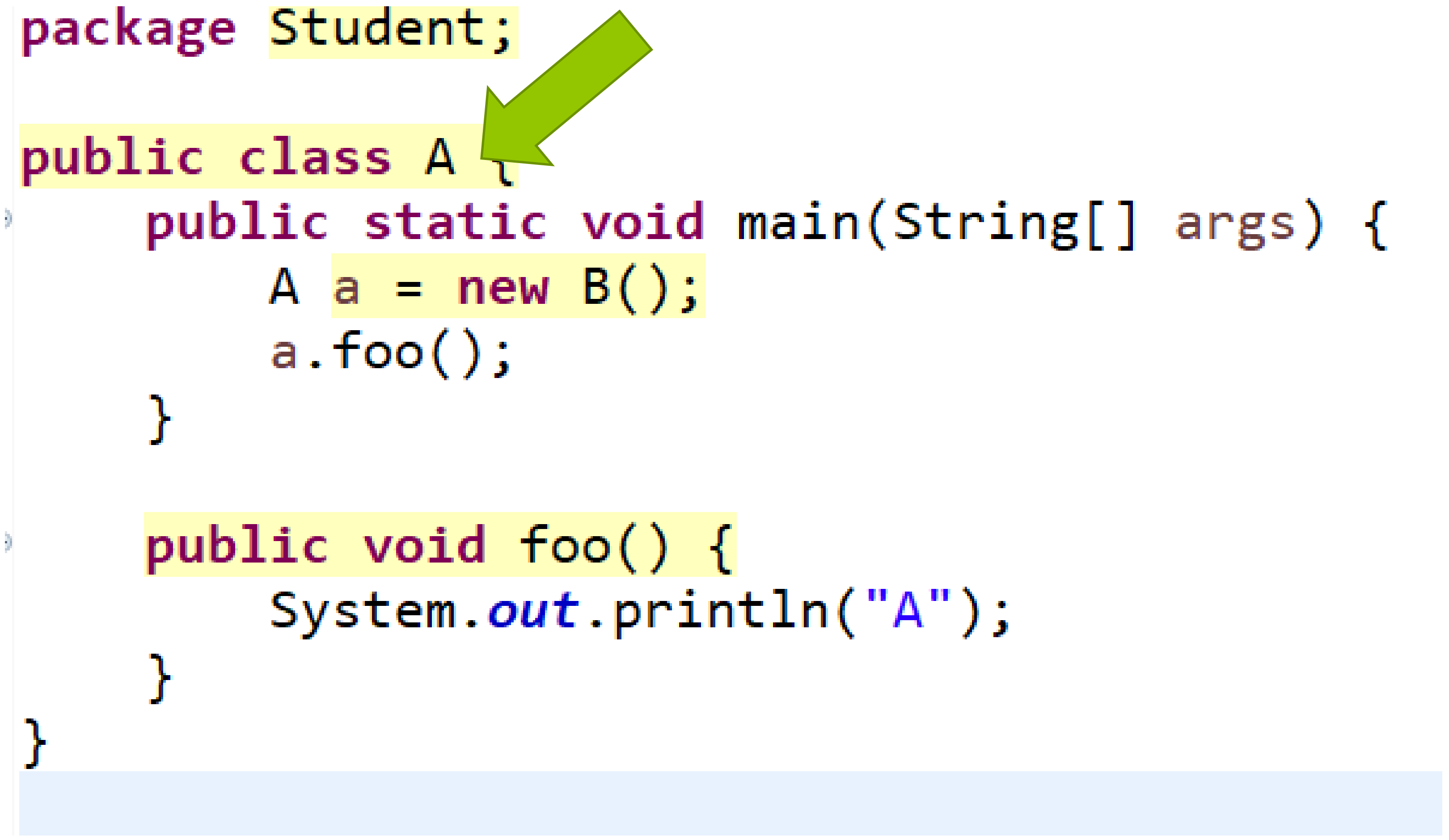


Right-click a file (or a folder if you want to check all files inside the folder), then choose Check Code with Checkstyle.

You can clear the Checkstyle with the Clear option.

Run result

Checkstyle will highlight the problems.
Hover your mouse over each highlight to see its suggestion.



```
package Student;

public class A {
    public static void main(String[] args) {
        A a = new B();
        a.foo();
    }

    public void foo() {
        System.out.println("A");
    }
}
```