

# 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* begins with all uppercase letter.
  - 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 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");
.....}
....}
}
```

**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.



It's even good to be really obvious.

```
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. */
```

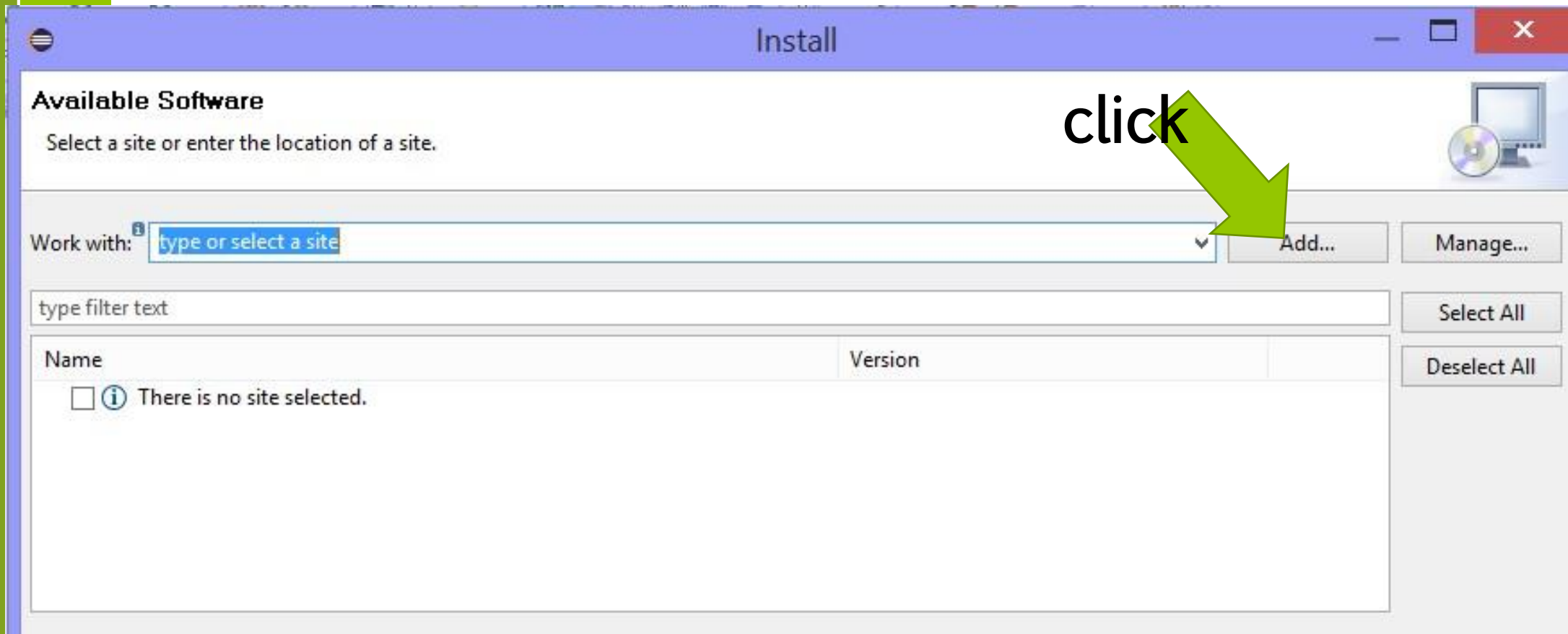
**Tip:** When writing multi-line comments, use the `/* ... */` style if you want automatic code formatters to re-wrap the lines when necessary (paragraph-style). Most formatters don't re-wrap lines in `// ...` style comment blocks.





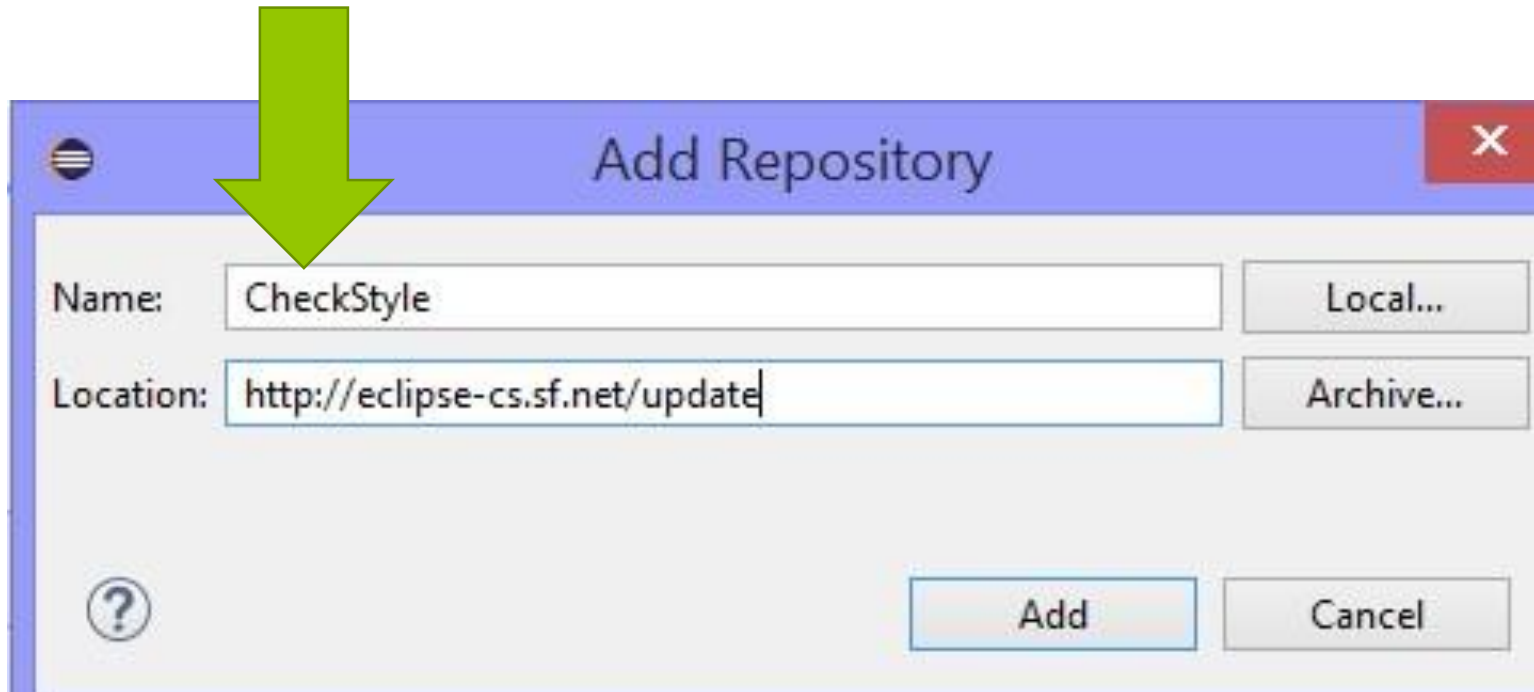
# Forcing good style with CheckStyle Plugin

› Install

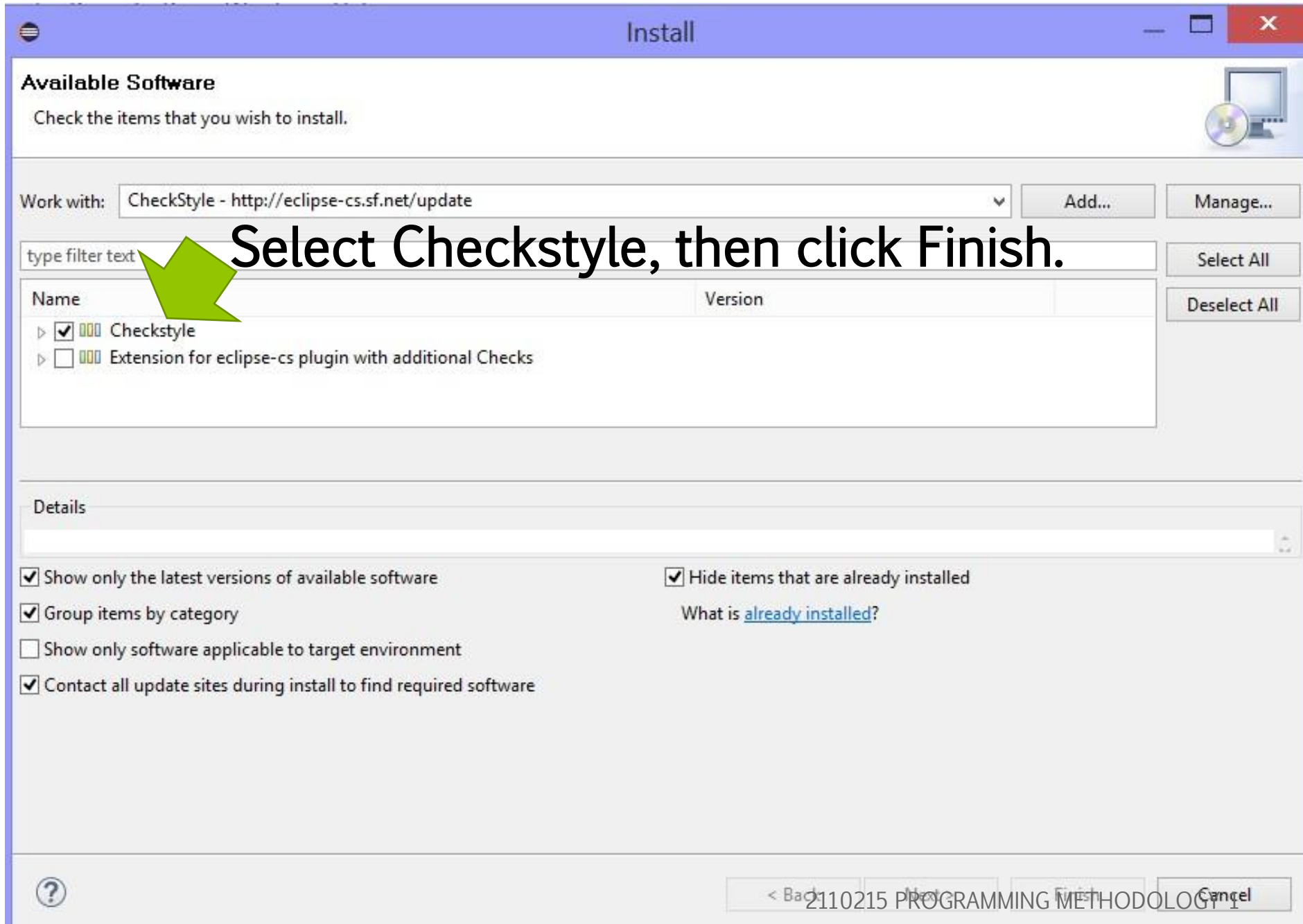




Fill in text and site, then click Add



The screenshot shows the 'Add Repository' dialog box. The 'Name' field is labeled 'Name:' and contains the text 'CheckStyle'. The 'Location' field is labeled 'Location:' and contains the URL 'http://eclipse-cs.sf.net/update'. To the right of the 'Name' field is a button labeled 'Local...'. To the right of the 'Location' field is a button labeled 'Archive...'. At the bottom right are two buttons: 'Add' and 'Cancel'. A green arrow points to the 'Name' field. A question mark icon is in the bottom left corner.





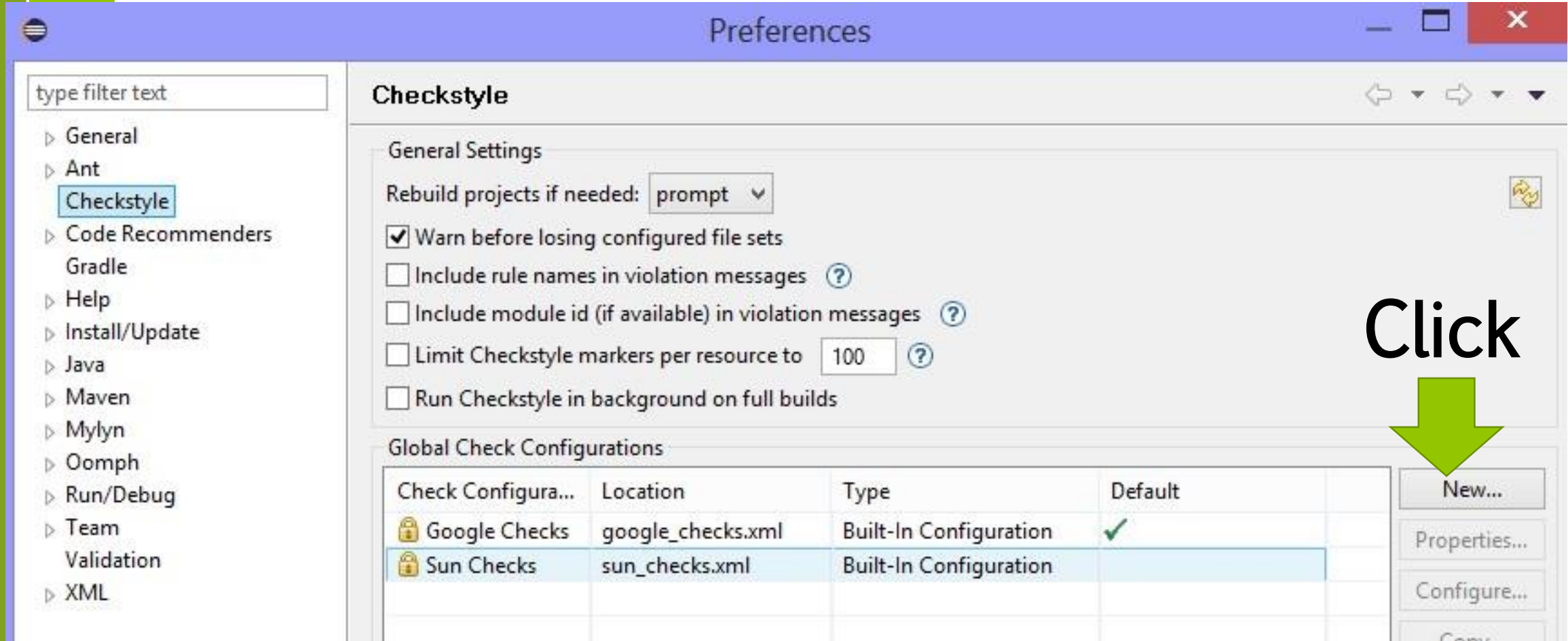


# The default style is too much!

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



# Windows-> Preferences





Check Configuration Properties

**Check Configuration**  
Create a new Check Configuration

**eclipse-cs**  
ECLIPSE-CHECKSTYLE INTEGRATION

Type: Internal Configuration

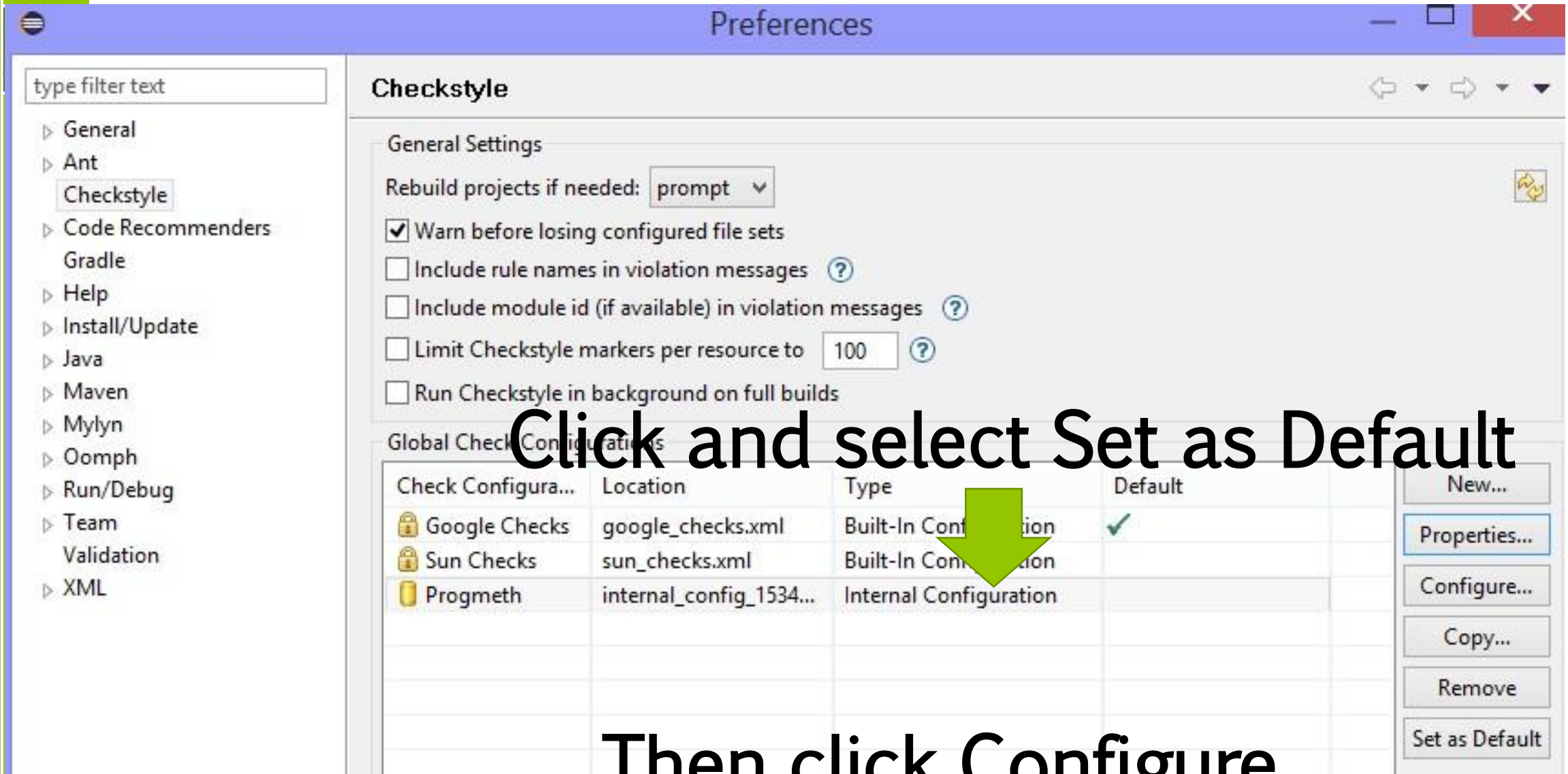
Name: Progmeth

Location:

Description:

Additional properties... ? OK Cancel Import...

Then Click OK



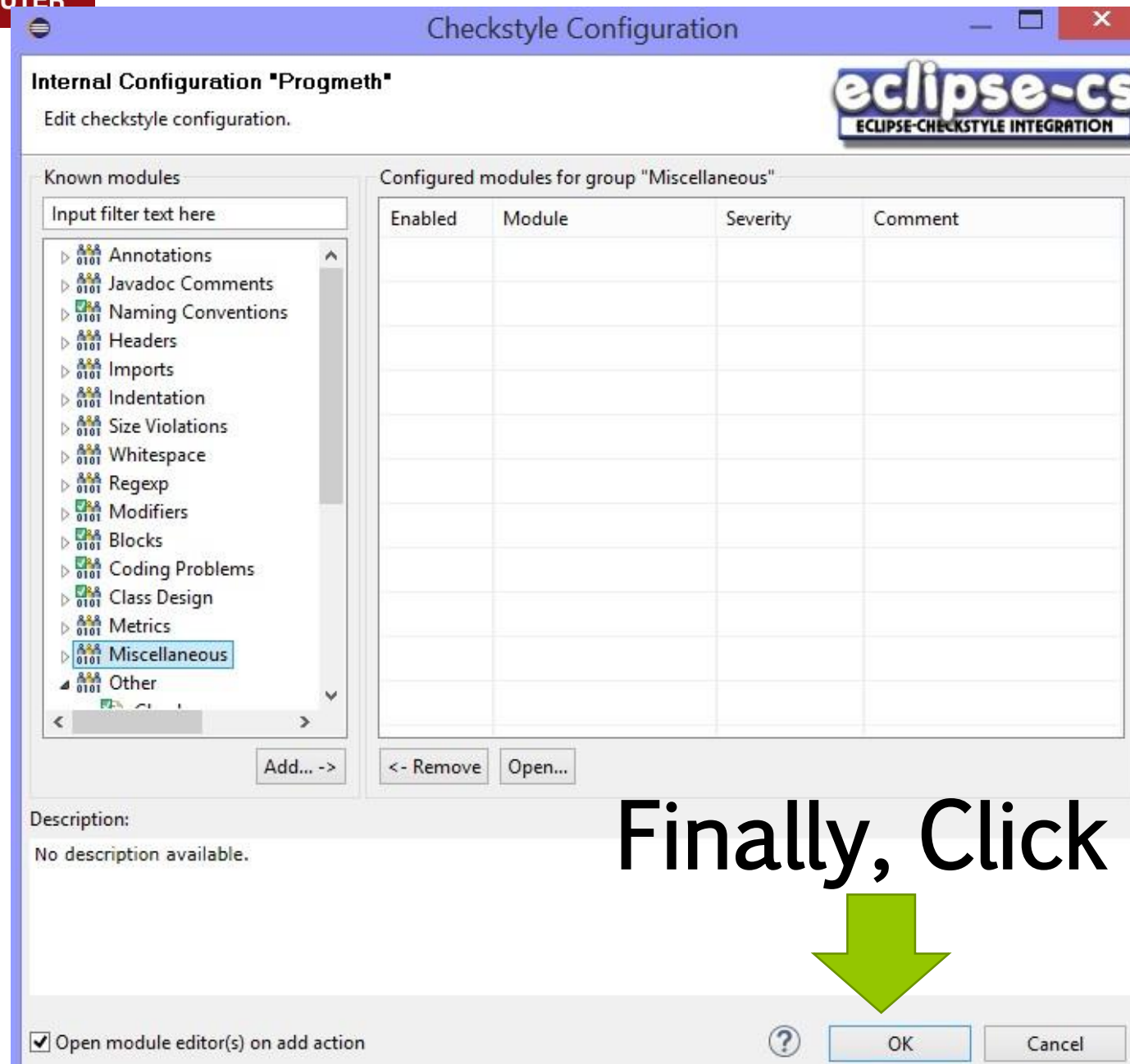
Click and select Set as Default

Then click Configure...

[illegible]

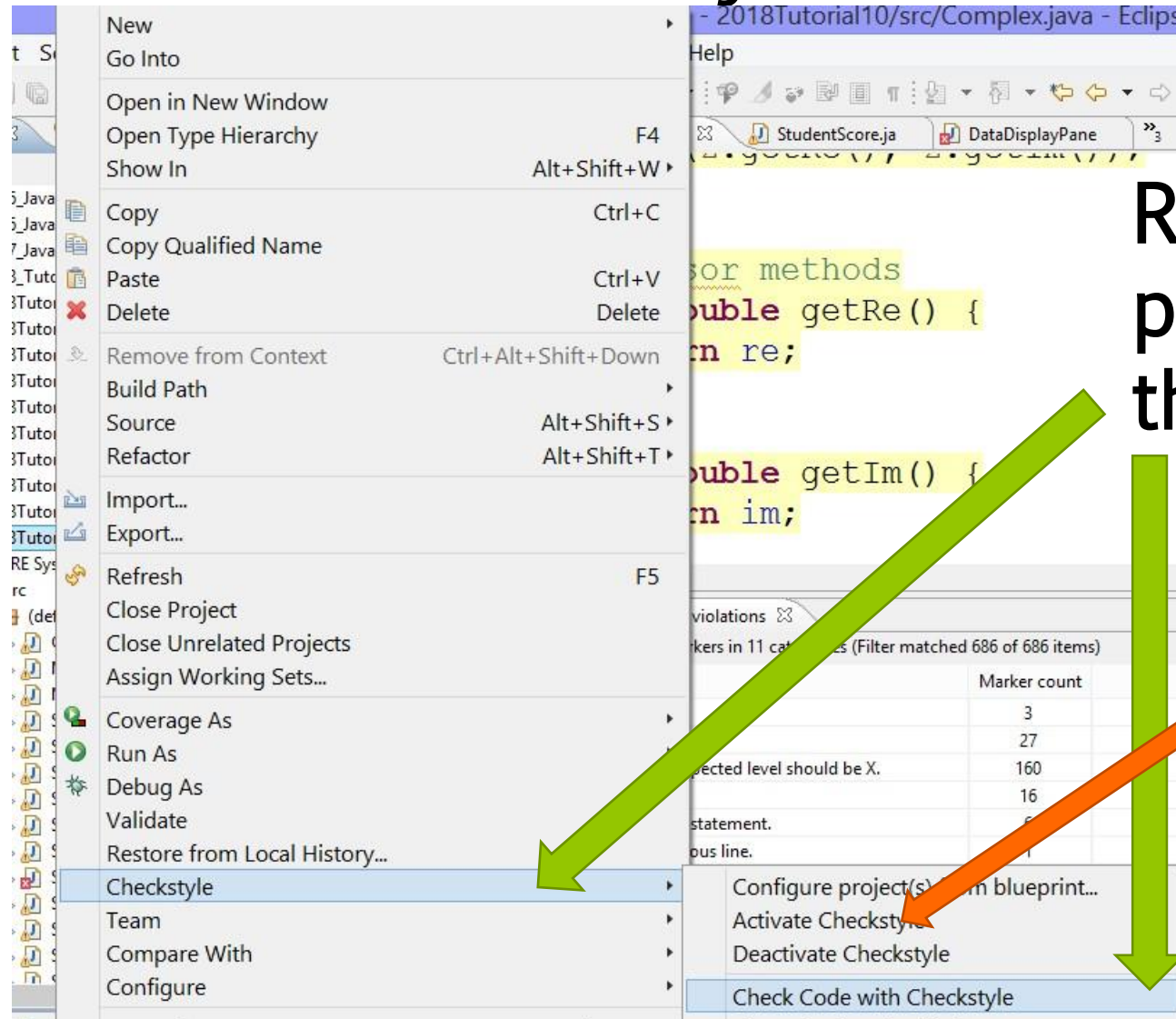
- Naming Convention
- Modifiers
- Blocks
- Coding Problems
- Class Design







# How to run checkstyle



This is for auto  
check at every  
save

# Run result

```

13 public MyPoint(double x, double y) {
14     this.x = x;
15     this.y = y;
16 }
17 public MyPoint(MyPoint p) {
18     this(p.getX(), p.getY());
19 }
20
21 // accessor methods
22 public double getX() {
23     return x;
24 }
25 public double getY() {
26     return y;
27 }

```