ITSS SOFTWARE DEVELOPMENT Lab 09 – Programming

Lecturer: NGUYEN Thi Thu Trang, trangntt@soict.hust.edu.vn

1. SUBMISSION GUIDELINE

You are required to push all your work to the valid GitHub repository complying with the naming convention:

"<MSTeamName>-<StudentID>.<StudentName>".

For this lab, you have to turn in your work twice before the following deadlines:

- **Right after class**: Push all the work you have done during class time to Github.
- **10 PM the day after the class**: Create a branch named "***release/lab09***" in your GitHub repository and push the full submission for this lab, including in-class tasks and homework assignments, to this branch.

2. IN-CLASS ASSIGNMENT

In this section, we will get familiar with the software construction process and try ourselves with programming for the Case Study.

You will learn how to add and generate Javadoc with Eclipse. Then, you will try identifying classes/methods that need be refactored and then refactor them. You would need Eclipse IDE, Oracle JDK 11, and then import the given sample project[1].

You are asked to work individually for this section and put the codes in "Programming" directory. After that, push your commit to your individual repository before the announced deadline.

2.1. GETTING STARTED

At first, please see the class **`API.java`** in the package ***`utils`*** and try to answer the following questions.

- What is purpose of this class?
- Who might write this class?

[1] https://github.com/trangntt-for-student/AIMS

- How to use this class?

You might find overwhelmed when looking at this class. Probably, you could not understand anything about this class at all. The class not only is a closed book to you, but also has problems with reusability, maintainability, etc. We would face these difficulties in the very next parts.

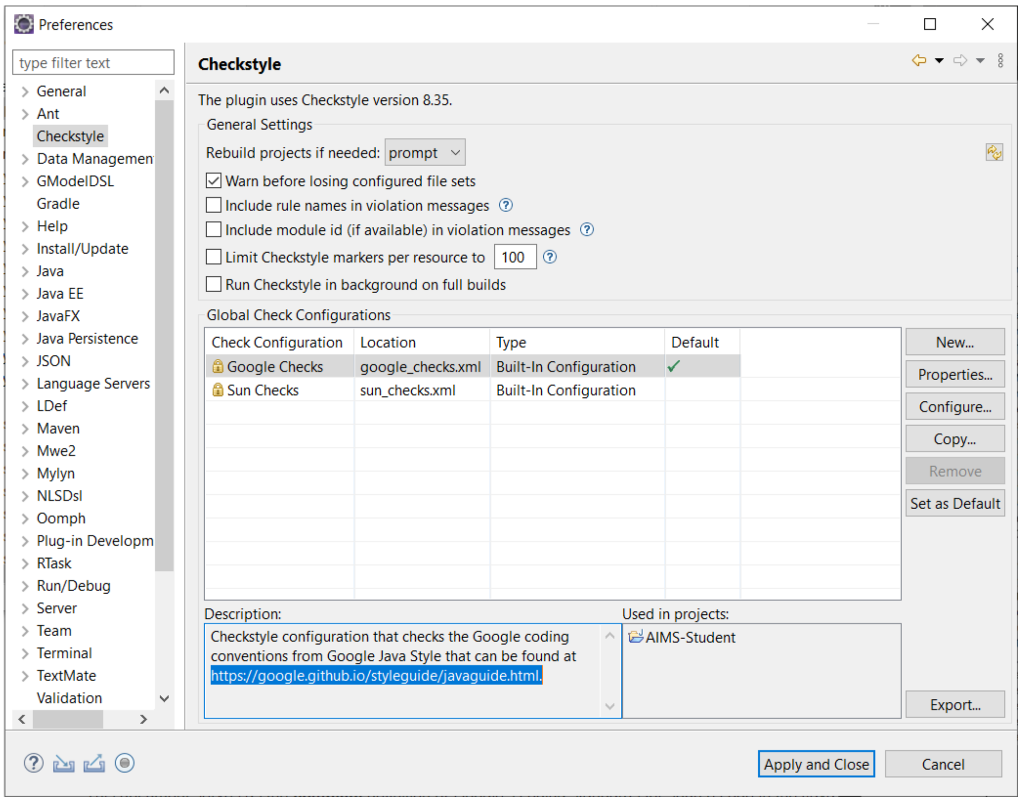
2.2. CHECKSTYLE

The programming style is one of the problems we need to deal with. Fortunately, there is a popular tool to help us with this – CheckStyle. Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard[2].

2.2.1. Installation

To install CheckStyle in Eclipse IDE, go to Eclipse -> Help -> Eclipse Marketplace..., search for CheckStyle Plugin (https://marketplace.eclipse.org/content/checkstyle-plug), and then install it. After restarting the Eclipse IDE, go to Eclipse -> Window -> Preferences -> Checkstyle. As can be seen, the default programming style is Google Java Style[3], yet you can always switch to/add here a new one that you prefer.

[2] https://checkstyle.org/
[3] https://google.github.io/styleguide/javaguide.html.



Checkstyle

The plugin uses Checkstyle version 8.35.

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

Description:

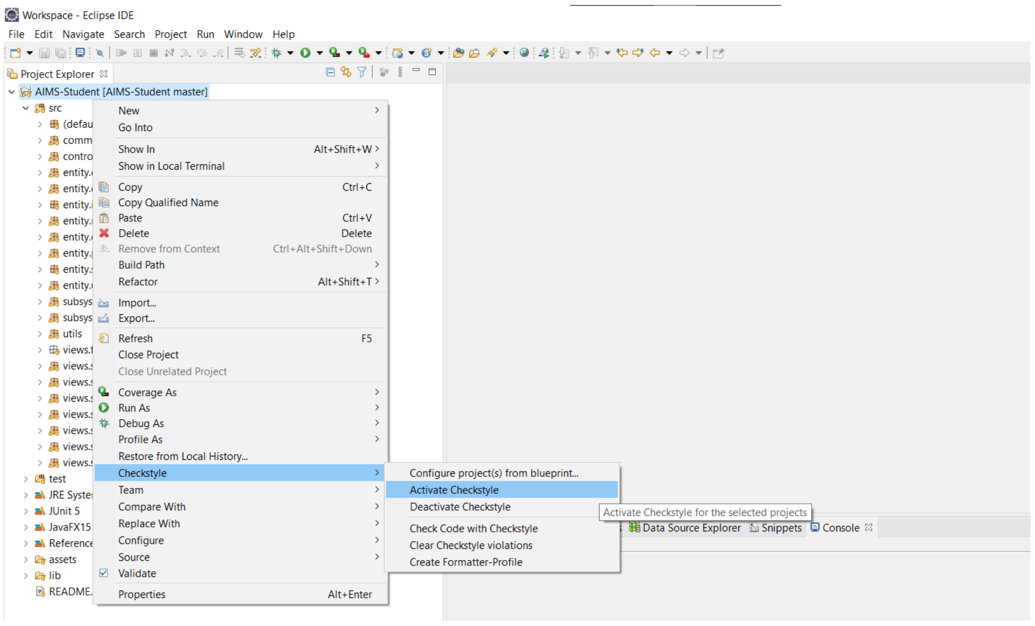
Checkstyle configuration that checks the Google coding conventions from Google Java Style that can be found at https://google.github.io/styleguide/javaguide.html.

Used in projects:

AIMS-Student

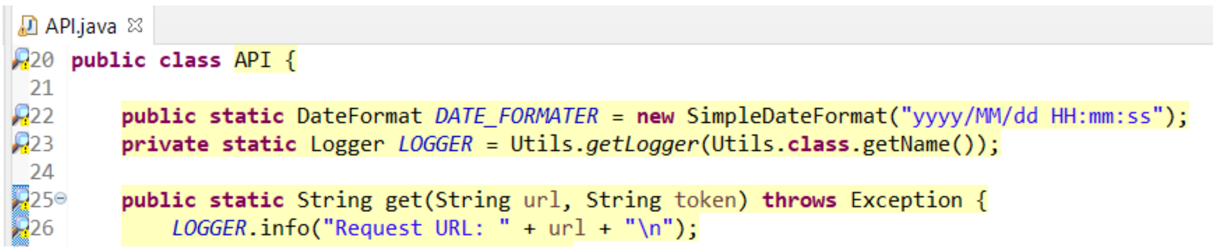
2.2.2. Usage

Activate Checkstyle



Detect convention violations

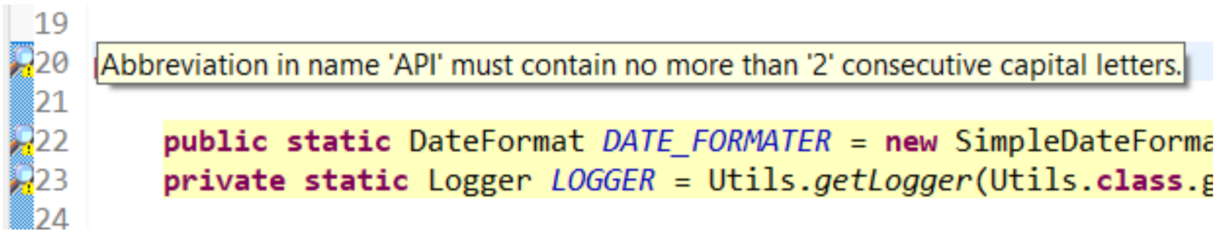
Open the class API. We can see that there are a lot of highlighted parts.



API.java

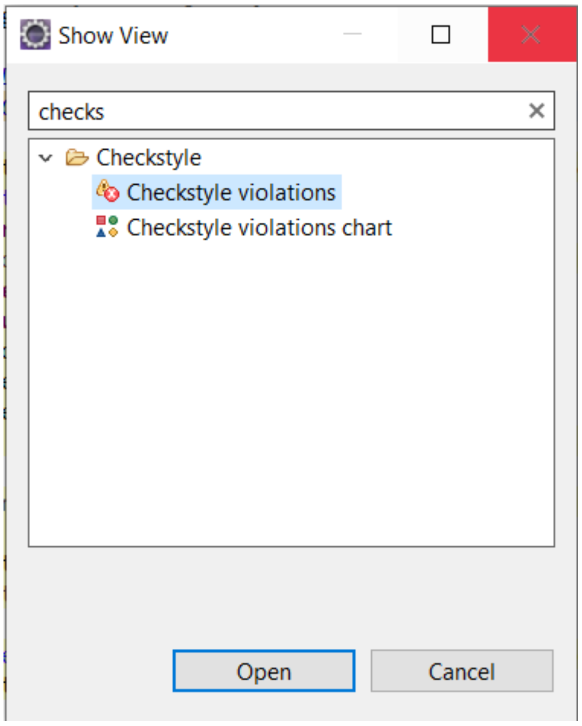
```
public class API {

    public static DateFormat DATE_FORMATER = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
    private static Logger LOGGER = Utils.getLogger(Utils.class.getName());

    public static String get(String url, String token) throws Exception {
        LOGGER.info("Request URL: " + url + "\n");
```

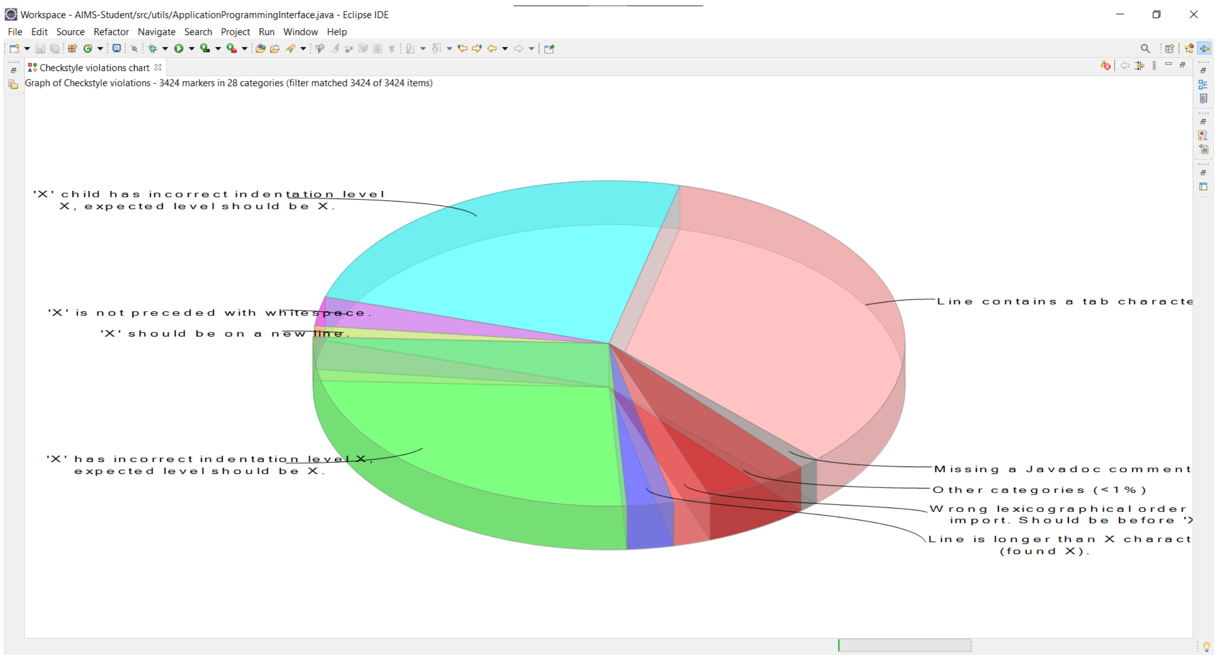
To see a violation, you can click or hover any highlighted part.



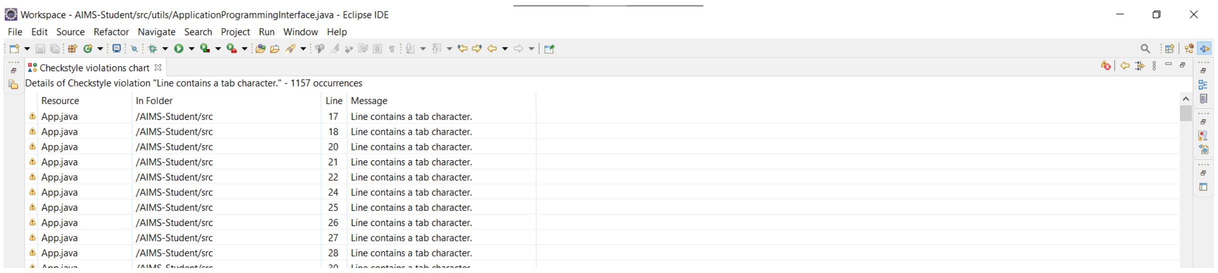
```
Abbreviation in name 'API' must contain no more than '2' consecutive capital letters.

    public static DateFormat DATE_FORMATER = new SimpleDateForma
    private static Logger LOGGER = Utils.getLogger(Utils.class.g
```

Additionally, you can see the violation report by go to Eclipse -> Window -> Show View -> Checkstyle violations





Graph of Checkstyle violations - 3424 markers in 28 categories (filter matched 3424 of 3424 items)


Workspace - AIMS-Student/src/utils/ApplicationProgrammingInterface.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Checkstyle violations chart

Details of Checkstyle violation "Line contains a tab character." - 1157 occurrences

Resource	In Folder	Line	Message
App.java	/AIMS-Student/src	17	Line contains a tab character.
App.java	/AIMS-Student/src	18	Line contains a tab character.
App.java	/AIMS-Student/src	20	Line contains a tab character.
App.java	/AIMS-Student/src	21	Line contains a tab character.
App.java	/AIMS-Student/src	22	Line contains a tab character.
App.java	/AIMS-Student/src	24	Line contains a tab character.
App.java	/AIMS-Student/src	25	Line contains a tab character.
App.java	/AIMS-Student/src	26	Line contains a tab character.
App.java	/AIMS-Student/src	27	Line contains a tab character.
App.java	/AIMS-Student/src	28	Line contains a tab character.
App.java	/AIMS-Student/src	30	Line contains a tab character.

Compile with the programming style

Here is the result after refactoring and saving the class.

```java
public class ApplicationProgrammingInterface {

  public static DateFormat DATE_FORMATER = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
  private static Logger LOGGER = Utils.getLogger(Utils.class.getName());
```

You will see the more details of refactoring with Eclipse later.

2.3. DOCUMENTATION WITH JAVADOC

As mentioned in Lab06-Class Design, you had better write documentation comments like the design in SDD.

To sum up,

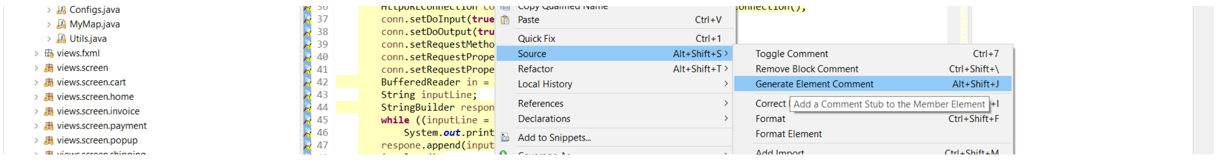
- Documentation comments (doc comments): The special comments in the Java source code that are delimited by the /** ... */ delimiters. These comments are processed by the Javadoc tool to generate the API docs.
- Javadoc: The JDK tool that generates API documentation from documentation comments.
- Include tags in the following order:
    - @author (classes and interfaces only, required)
    - @version (classes and interfaces only, required)
    - @param (methods and constructors only)
    - @return (methods only)
    - @exception (@throws is a synonym added in Javadoc 1.2)
    - @see – Ask to see another element for better understanding, usually used with {@link}
    - @since – the version which this element is added
    - @serial (or @serialField or @serialData)
    - @deprecated - let others know that this method or class would be removed

  Even though the name of the tag is self-explained, you are recommended to see one of the following links.

  [How to Write Doc Comments for the Javadoc Tool (oracle.com)](https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html) [4]

  [Java: Javadoc tags | I'd Rather Be Writing](https://idratherbewriting.com/java-javadoc-tags/)[5]

To generate an initial doc comment for **an element**, e.g., class/method/attribute, right click on it -> Source -> Generate Element Comment, or click on the element and use the hot keys Alt + Shift + J.



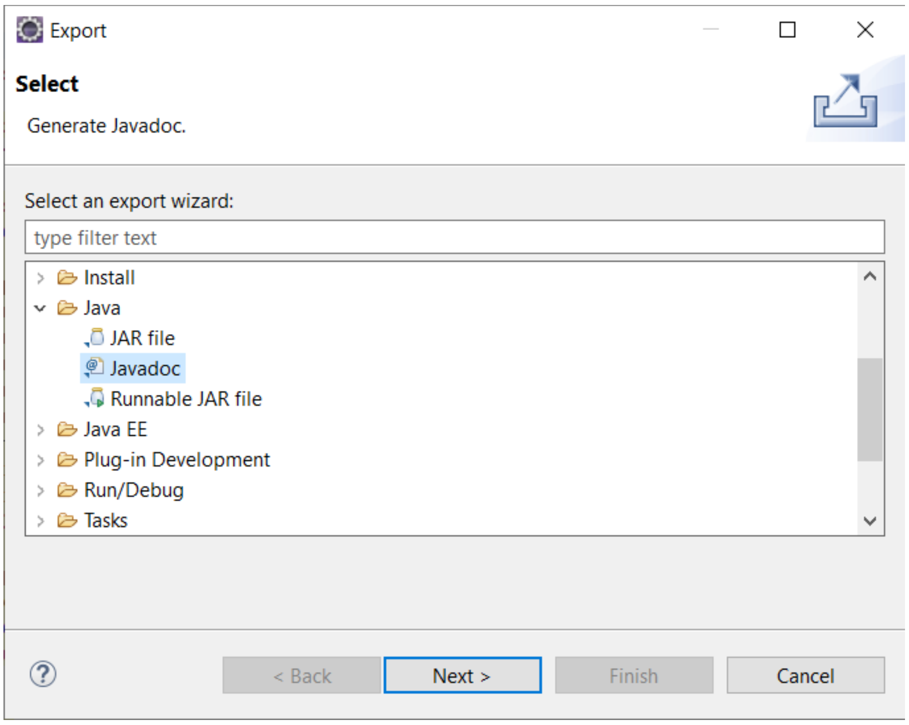
```
conn.setDoInput(true
conn.setDoOutput(tru
conn.setRequestMetho
conn.setRequestPrope
conn.setRequestPrope
BufferedReader in =
String inputLine;
StringBuilder respon
while ((inputLine =
    System.out.print
respone.append(input
```

---

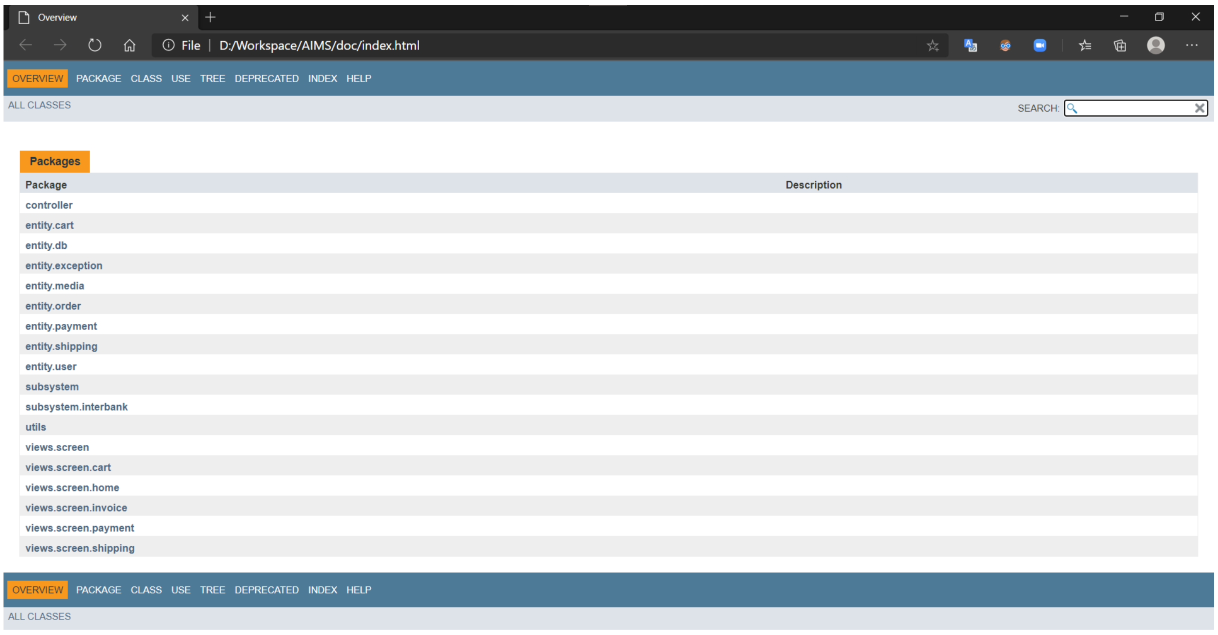
4 https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
5 https://idratherbewriting.com/java-javadoc-tags/

Besides, you could use in-line comment '//' to make things more clearly. Note that this kind of comments is not doc comment.

Finally, when you complete a version of the project, you can export the Javadoc by going to Eclipse -> File -> Export -> Java -> Javadoc and do as instructed.

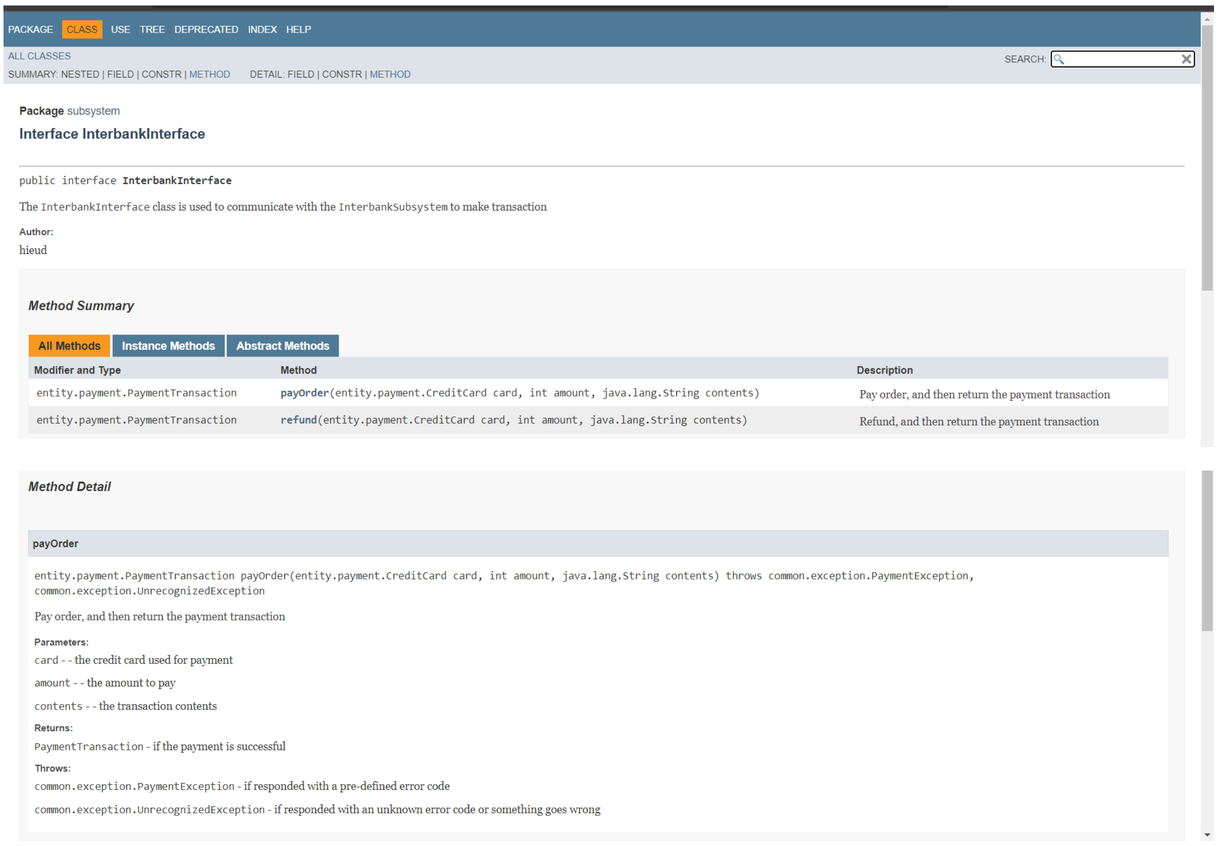


In the project directory, go to doc/index.html to see the result (if you decide to put it in the default storing location).


Overview
File | D:/Workspace/AIMS/doc/index.html
OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
ALL CLASSES
SEARCH:

Packages

Package	Description
controller	
entity.cart	
entity.db	
entity.exception	
entity.media	
entity.order	
entity.payment	
entity.shipping	
entity.user	
subsystem	
subsystem.interbank	
utils	
views.screen	
views.screen.cart	
views.screen.home	
views.screen.invoice	
views.screen.payment	
views.screen.shipping	


PACKAGE
CLASS
USE
TREE
DEPRECATED
INDEX
HELP
ALL CLASSES
SEARCH:
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

Package subsystem

Interface InterbankInterface

```
public interface InterbankInterface
```

The `InterbankInterface` class is used to communicate with the `InterbankSubsystem` to make transaction

Author:
hieud

Method Summary

Modifier and Type	Method	Description
`entity.payment.PaymentTransaction`	`payOrder(entity.payment.CreditCard card, int amount, java.lang.String contents)`	Pay order, and then return the payment transaction
`entity.payment.PaymentTransaction`	`refund(entity.payment.CreditCard card, int amount, java.lang.String contents)`	Refund, and then return the payment transaction

Method Detail

payOrder

```
entity.payment.PaymentTransaction payOrder(entity.payment.CreditCard card, int amount, java.lang.String contents) throws common.exception.PaymentException,
common.exception.UnrecognizedException
```

Pay order, and then return the payment transaction

Parameters:
`card` - - the credit card used for payment
`amount` - - the amount to pay
`contents` - - the transaction contents

Returns:
`PaymentTransaction` - if the payment is successful

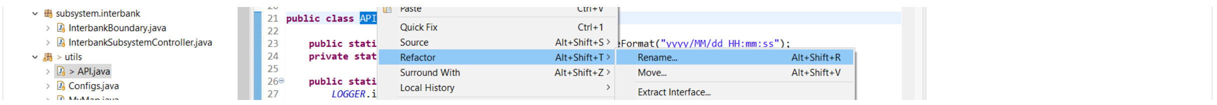
Throws:
`common.exception.PaymentException` - if responded with a pre-defined error code
`common.exception.UnrecognizedException` - if responded with an unknown error code or something goes wrong

2.4. REFACTORING WITH ECLIPSE IDE

2.4.1. Renaming an element or Moving an element to another element

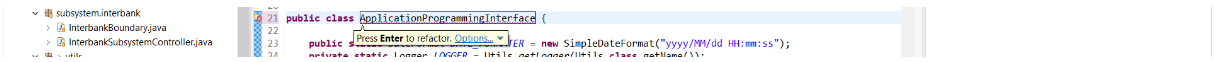
Previously, the CheckStyle detected that the class name **`API`** does not compile with the Google Java Style. Thus, we might want to change the name of the class into another one, e.g., `ApplicationProgrammingInterface`.

To change the name of an element, e.g., package, class, method, attribute, etc., select the element name -> right click -> Refactor -> Rename...



```
public class API
    public stati
    private stat

    public stati
        LOGGER.i
```

Then, type the new one and press Enter.

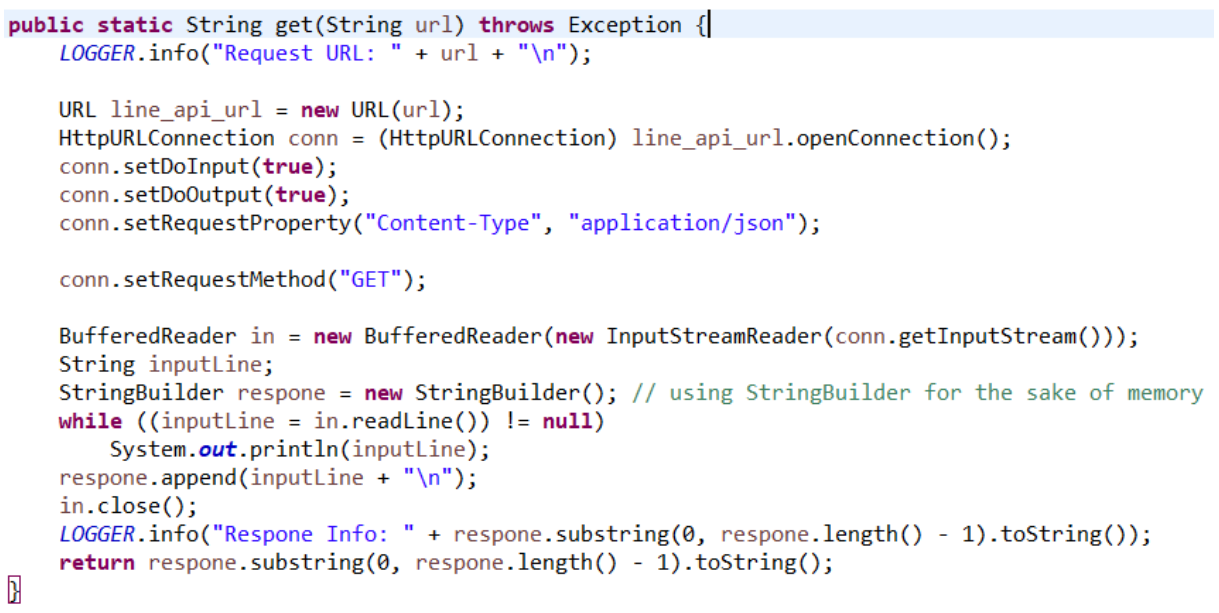

subsystem.interbank
InterbankBoundary.java
InterbankSubsystemController.java
Press Enter to refactor. Options...

```
public class ApplicationProgrammingInterface {

    public s                                TER = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
```

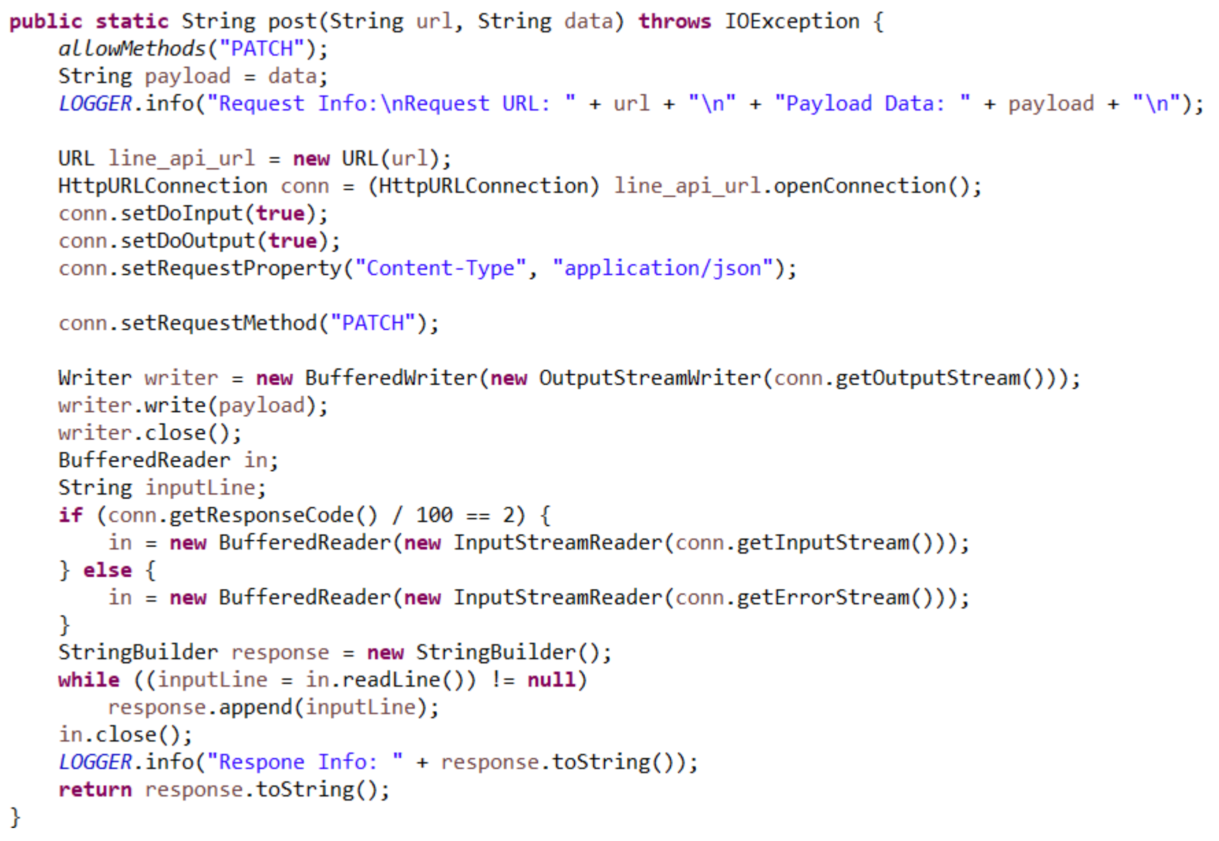
Similarly, you can move an element to another place (e.g., package or class) by select the element name -> right click -> Refactor -> Move...

2.4.2. Extract a new element

Obviously, there are lots of code smells in the class **`API`**. To illustrate, the 2 methods **`get()`** and **`post()`** share some similar lines of codes, which are setting up connection to server and reading response from server. Thus, we would extract these lines to new methods.

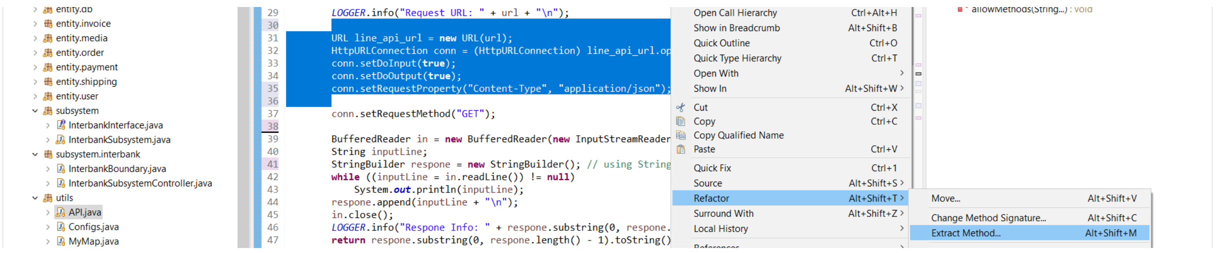


```java
public static String get(String url) throws Exception {
    LOGGER.info("Request URL: " + url + "\n");

    URL line_api_url = new URL(url);
    HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestProperty("Content-Type", "application/json");

    conn.setRequestMethod("GET");

    BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    String inputLine;
    StringBuilder respone = new StringBuilder(); // using StringBuilder for the sake of memory
    while ((inputLine = in.readLine()) != null)
        System.out.println(inputLine);
    respone.append(inputLine + "\n");
    in.close();
    LOGGER.info("Respone Info: " + respone.substring(0, respone.length() - 1).toString());
    return respone.substring(0, respone.length() - 1).toString();
}
```



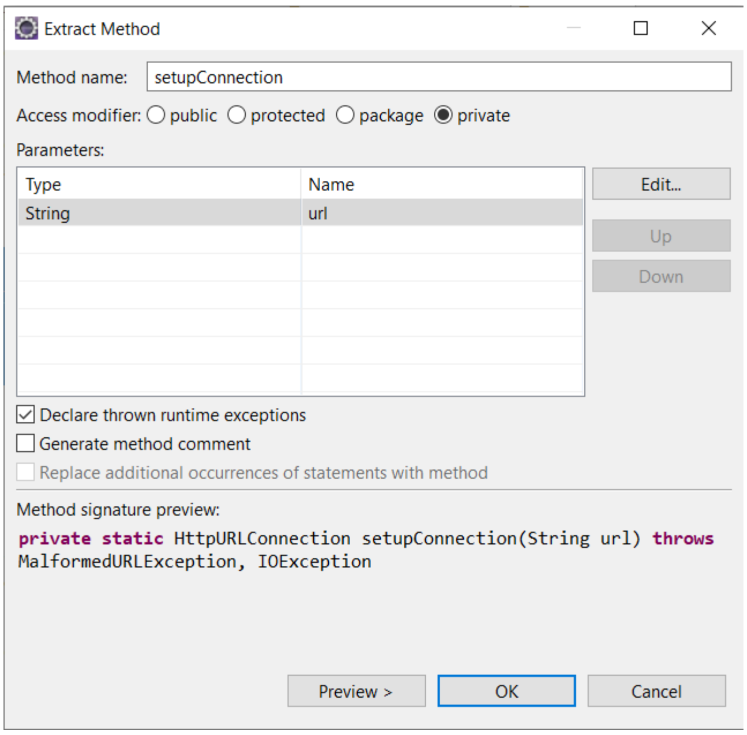
```java
public static String post(String url, String data) throws IOException {
    allowMethods("PATCH");
    String payload = data;
    LOGGER.info("Request Info:\nRequest URL: " + url + "\n" + "Payload Data: " + payload + "\n");

    URL line_api_url = new URL(url);
    HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestProperty("Content-Type", "application/json");

    conn.setRequestMethod("PATCH");

    Writer writer = new BufferedWriter(new OutputStreamWriter(conn.getOutputStream()));
    writer.write(payload);
    writer.close();
    BufferedReader in;
    String inputLine;
    if (conn.getResponseCode() / 100 == 2) {
        in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    } else {
        in = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
    }
    StringBuilder response = new StringBuilder();
    while ((inputLine = in.readLine()) != null)
        response.append(inputLine);
    in.close();
    LOGGER.info("Respone Info: " + response.toString());
    return response.toString();
}
```

To extract, select the lines -> right click -> Refactor -> Extract Method...



```
LOGGER.info("Request URL: " + url + "\n");

URL line_api_url = new URL(url);
HttpURLConnection conn = (HttpURLConnection) line_api_url.op
conn.setDoInput(true);
conn.setDoOutput(true);
conn.setRequestProperty("Content-Type", "application/json");

conn.setRequestMethod("GET");

BufferedReader in = new BufferedReader(new InputStreamReader
String inputLine;
StringBuilder respone = new StringBuilder(); // using String
while ((inputLine = in.readLine()) != null)
    System.out.println(inputLine);
respone.append(inputLine + "\n");
in.close();
LOGGER.info("Respone Info: " + respone.substring(0, respone.
return respone.substring(0, respone.length() - 1).toString()
```

Change the method name and others if need be. Then, click OK.



Extract Method

Method name: setupConnection

Access modifier: public protected package [✓ private]

Parameters:

Type	Name
String	url

[✓] Declare thrown runtime exceptions

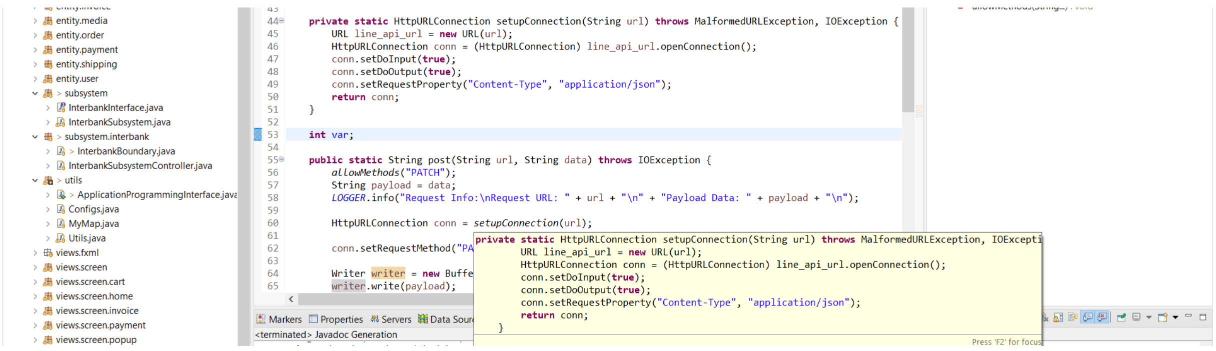
[ ] Generate method comment

[ ] Replace additional occurrences of statements with method

Method signature preview:

```
private static HttpURLConnection setupConnection(String url) throws
MalformedURLException, IOException
```

Then, we can change the other method to use the extracted method.



```
private static HttpURLConnection setupConnection(String url) throws MalformedURLException, IOException {
    URL line_api_url = new URL(url);
    HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestProperty("Content-Type", "application/json");
    return conn;
}

int var;

public static String post(String url, String data) throws IOException {
    allowMethods("PATCH");
    String payload = data;
    LOGGER.info("Request Info:\nRequest URL: " + url + "\n" + "Payload Data: " + payload + "\n");

    HttpURLConnection conn = setupConnection(url);

    conn.setRequestMethod("PA

    Writer writer = new Buffe
    writer.write(payload);
```

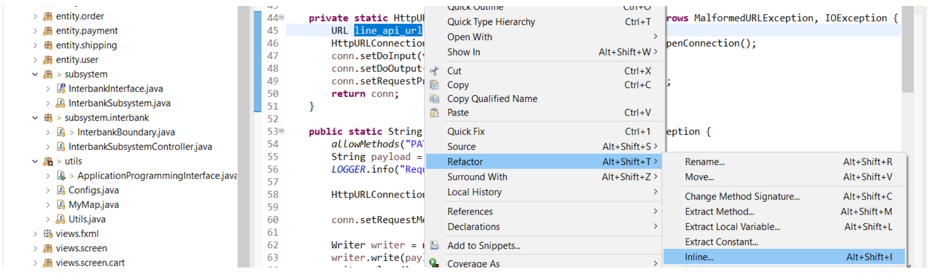
```
private static HttpURLConnection setupConnection(String url) throws MalformedURLException, IOExcepti
        URL line_api_url = new URL(url);
        HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
        conn.setDoInput(true);
        conn.setDoOutput(true);
        conn.setRequestProperty("Content-Type", "application/json");
        return conn;
    }
```

Similarly, we can extract codes into interface, class, etc.

2.4.3. Inlining

In the newly created method **`setupConnection()`**, we see that the local variable **`line_api_url`** is used only once, so we might want to inline this.

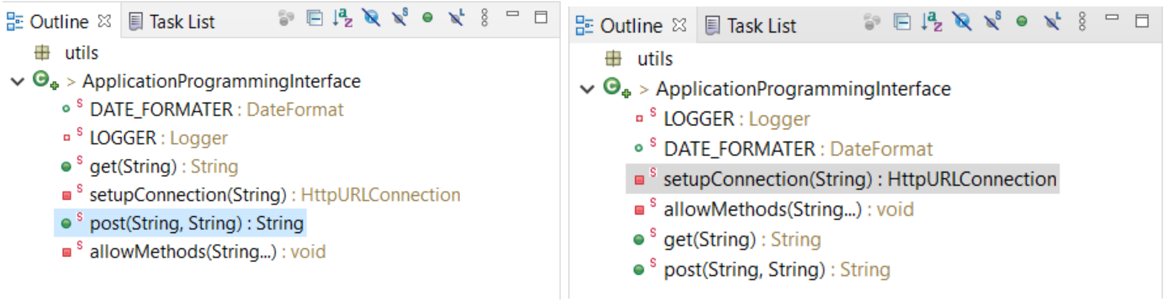
To inline, select the element name -> right click -> Refractor -> Inline...



```
    private static HttpUR                              rows MalformedURLException, IOException {
        URL line_api_url
        HttpURLConnectio                               penConnection();
        conn.setDoInput(
        conn.setDoOutput
        conn.setRequestPr                              ;
        return conn;
    }

    public static String                               eption {
        allowMethods("PA
        String payload =
        LOGGER.info("Requ

        HttpURLConnectio

        conn.setRequestMe

        Writer writer =
        writer.write(pay
```

2.4.4. Moving inside a class

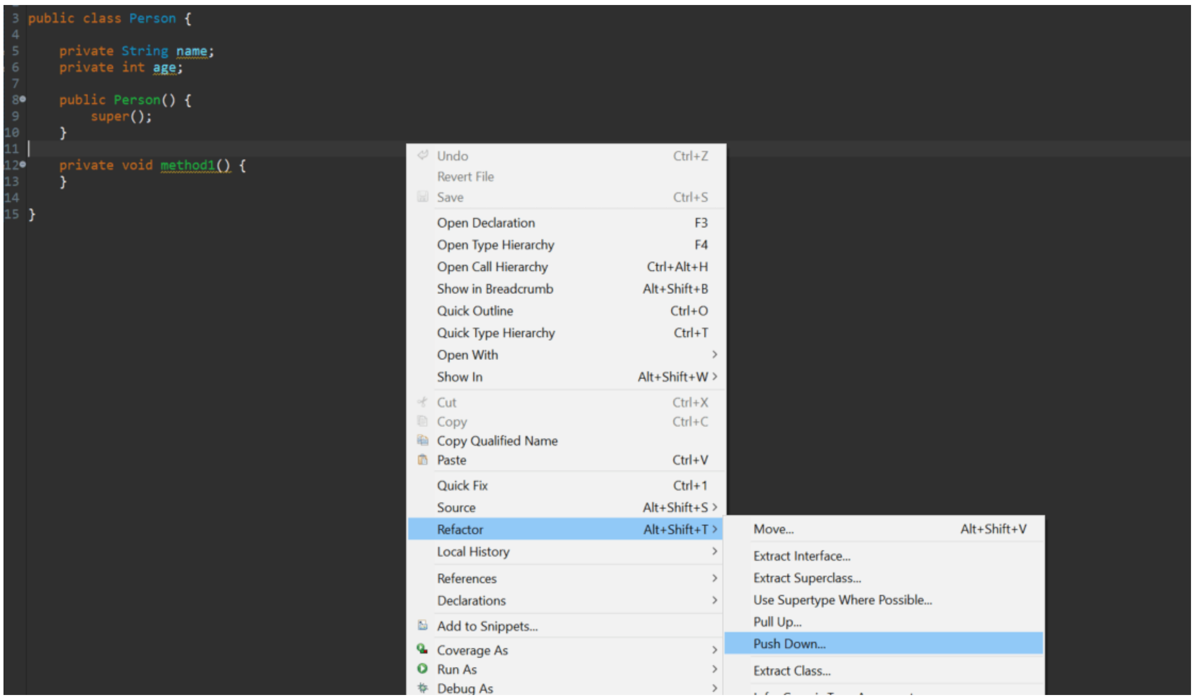
To change the order of elements in a class (e.g., method, attribute), on the Outline in the right of default Java EE perspective, you can drag any element. This would make their codes switch places, too.


Outline
Task List
utils
ApplicationProgrammingInterface
DATE_FORMATER : DateFormat
LOGGER : Logger
get(String) : String
setupConnection(String) : HttpURLConnection
post(String, String) : String
allowMethods(String...) : void
Outline
Task List
utils
ApplicationProgrammingInterface
LOGGER : Logger
DATE_FORMATER : DateFormat
setupConnection(String) : HttpURLConnection
allowMethods(String...) : void
get(String) : String
post(String, String) : String

2.4.5. Push Down and Pull Up[6]

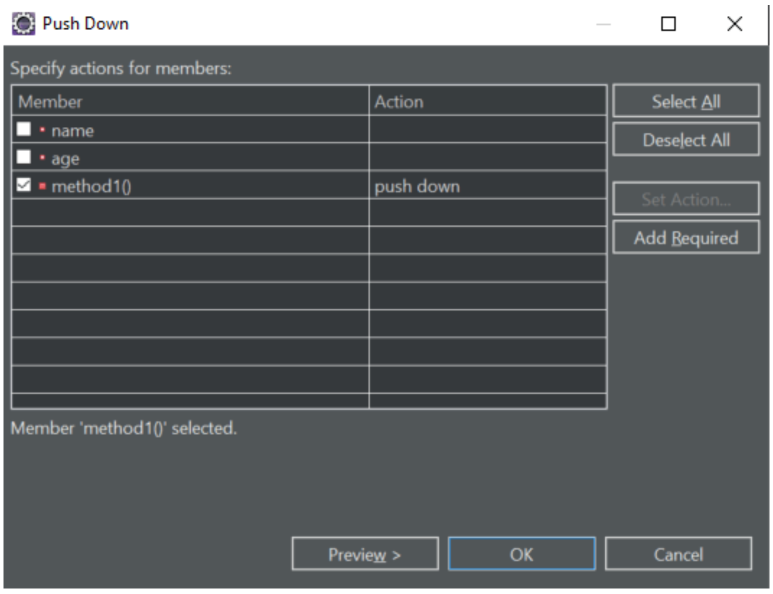
If we have a parent-child relationship (like *Employee* and *Person*) between our classes, and we want to move certain methods or variables among them, we can use the push/pull options provided by Eclipse. As the name suggests, the *Push Down* option moves methods and fields from a parent class to all child classes, while *Pull Up* moves methods and fields from a particular child class to parent, thus making that method available to all the child classes. For moving methods down to child classes, we need to **right-click anywhere in the class and choose the *Refactor > Push Down* option**:

[6] https://www.baeldung.com/eclipse-refactoring#push-up-down



```
public class Person {

    private String name;
    private int age;

    public Person() {
        super();
    }

    private void method1() {
    }
}
```

This will open a wizard where we can select items to push down:



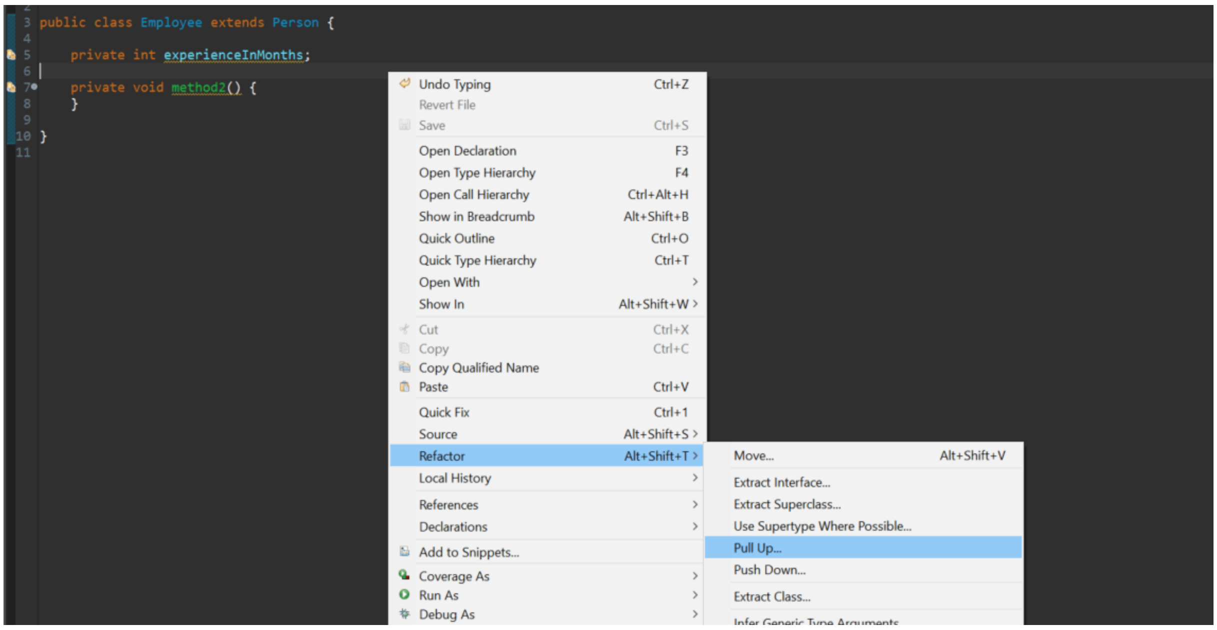
Push Down

Specify actions for members:

Member	Action
☐ name	
☐ age	
☑ method1()	push down

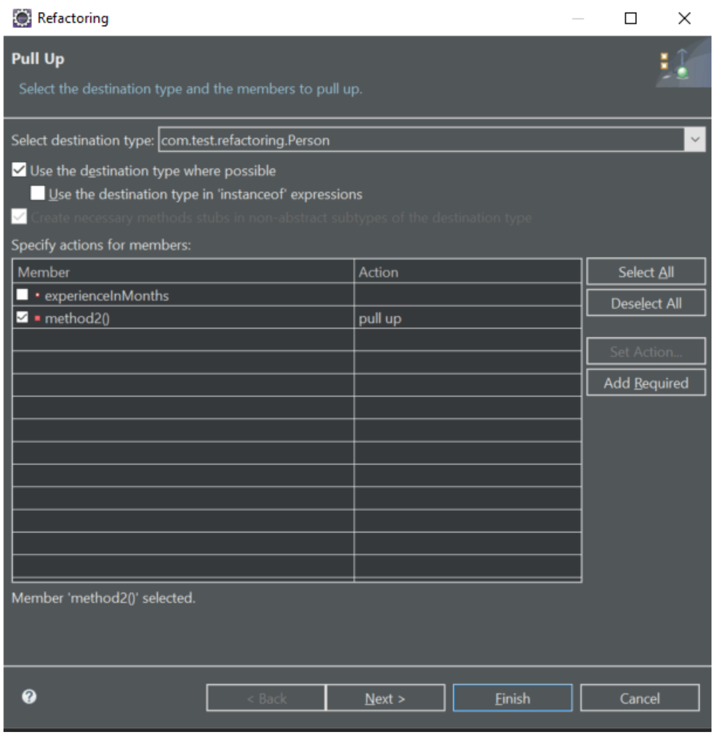
Member 'method1()' selected.

Similarly, for moving methods from a child class to parent class, we need to **right-click anywhere in the class and choose *Refactor > Pull Up***:



```
public class Employee extends Person {

    private int experienceInMonths;

    private void method2() {
    }

}
```

This will open a similar wizard where we can select items to pull up:



Refactoring

Pull Up

Select the destination type and the members to pull up.

Select destination type: com.test.refactoring.Person

☑ Use the destination type where possible
☐ Use the destination type in 'instanceof' expressions
☑ Create necessary methods stubs in non-abstract subtypes of the destination type

Specify actions for members:

Member	Action
☐ experienceInMonths	
☑ method2()	pull up

Member 'method2()' selected.

For better understanding, please see:

https://www.baeldung.com/eclipse-refactoring https://help.eclipse.org/2020-09/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Freference%2Fref-menu-refactor.htm

https://www.oreilly.com/library/view/learning-eclipse-java/9781771372824/

2.5. PROGRAMMING FOR UC "PLACE RUSH ORDER"

You are asked to:

- Implement the design of UC "Place Rush Order" by applying TDD and using related techniques which are learnt in this lab.

- Choose a programming style and apply it to the codes of UC "Place Rush Order" by using CheckStyle.

- Export the Javadoc into the default location in your project directory, namely doc.