2023/2024(1)

EF234302 Object Oriented Programming

Lecture #2

Eclipse IDE for Java Programming

Misbakhul Munir IRFAN SUBAKTI

司馬伊凡

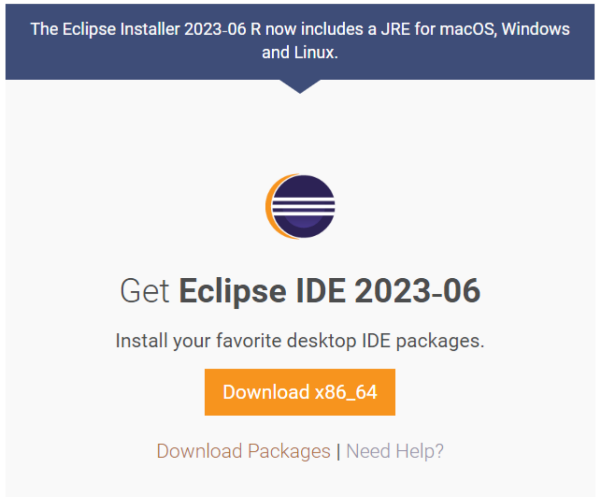
Мисбакхул Мунир Ирфан Субакти

Eclipse: Why?

- Website: www.eclipse.org
- Free, open-source IDE (Integrated Development Environment) that runs on most modern OS
- Commonly used for developing all of type Java applications including Android apps

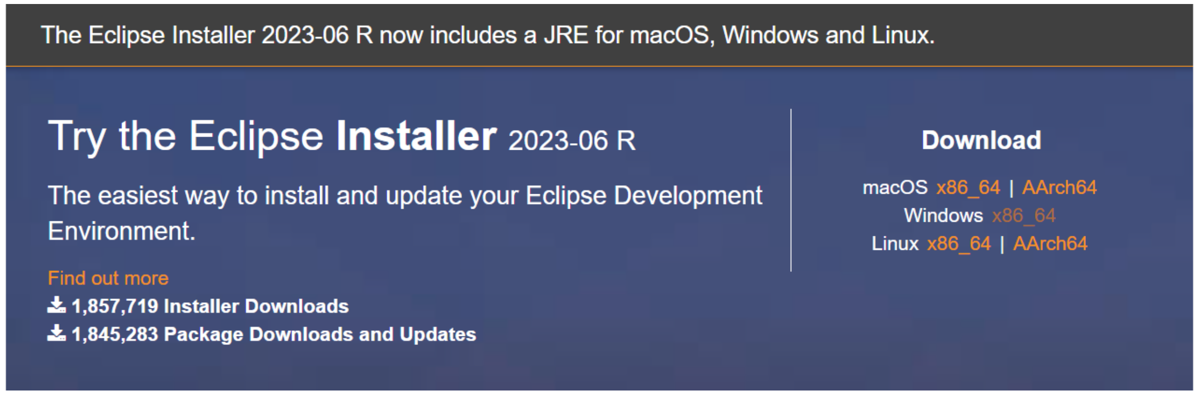
Downloading

- https://www.eclipse.org/downloads/


The Eclipse Installer 2023-06 R now includes a JRE for macOS, Windows and Linux.

Get **Eclipse IDE 2023-06**

Install your favorite desktop IDE packages.


The Eclipse Installer 2023-06 R now includes a JRE for macOS, Windows and Linux.

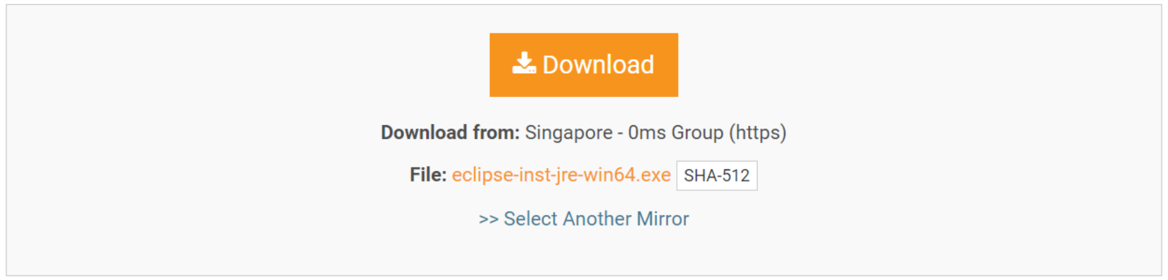
Try the Eclipse **Installer** 2023-06 R

The easiest way to install and update your Eclipse Development Environment.

Find out more

1,857,719 Installer Downloads

1,845,283 Package Downloads and Updates

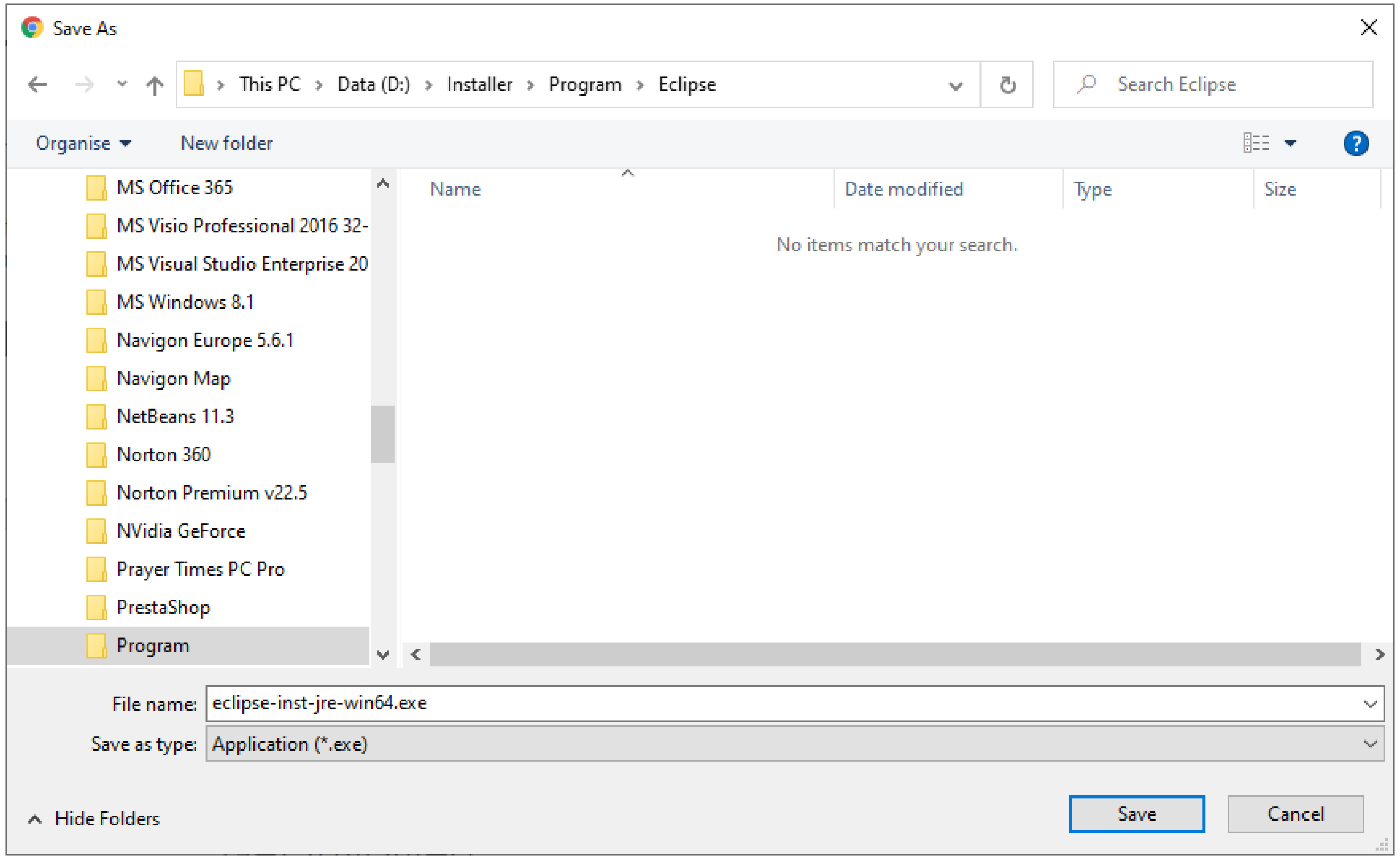


Download from: Singapore - 0ms Group (https)

File: eclipse-inst-jre-win64.exe SHA-512

>> Select Another Mirror

Downloading (continued)



No items match your search.

File name: eclipse-inst-jre-win64.exe

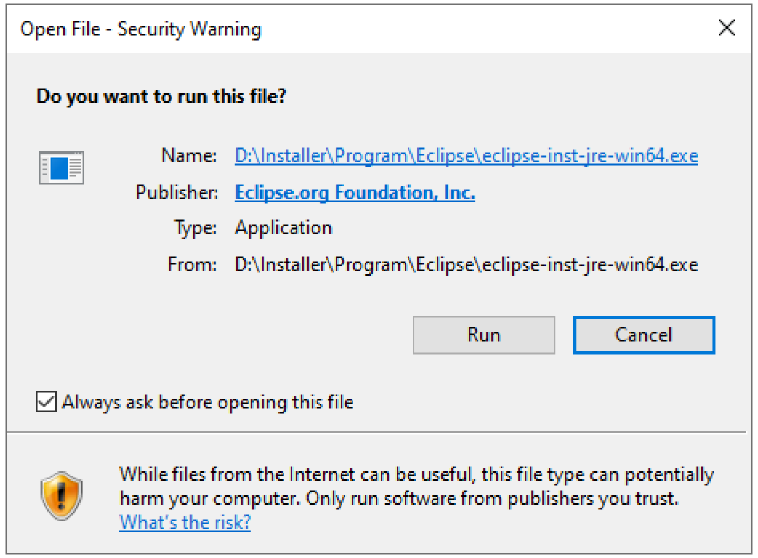
Save as type: Application (*.exe)

Downloading (continued)

- Linux → unzip it before you can start it.
- Mac → the installer is delivered as packaged application and can be installed and started regular Mac installation procedures.
- Windows and Mac → can run it directly via the delivered executable/ package application.

Installing

- Run the installation file has been downloaded



Open File - Security Warning

Do you want to run this file?

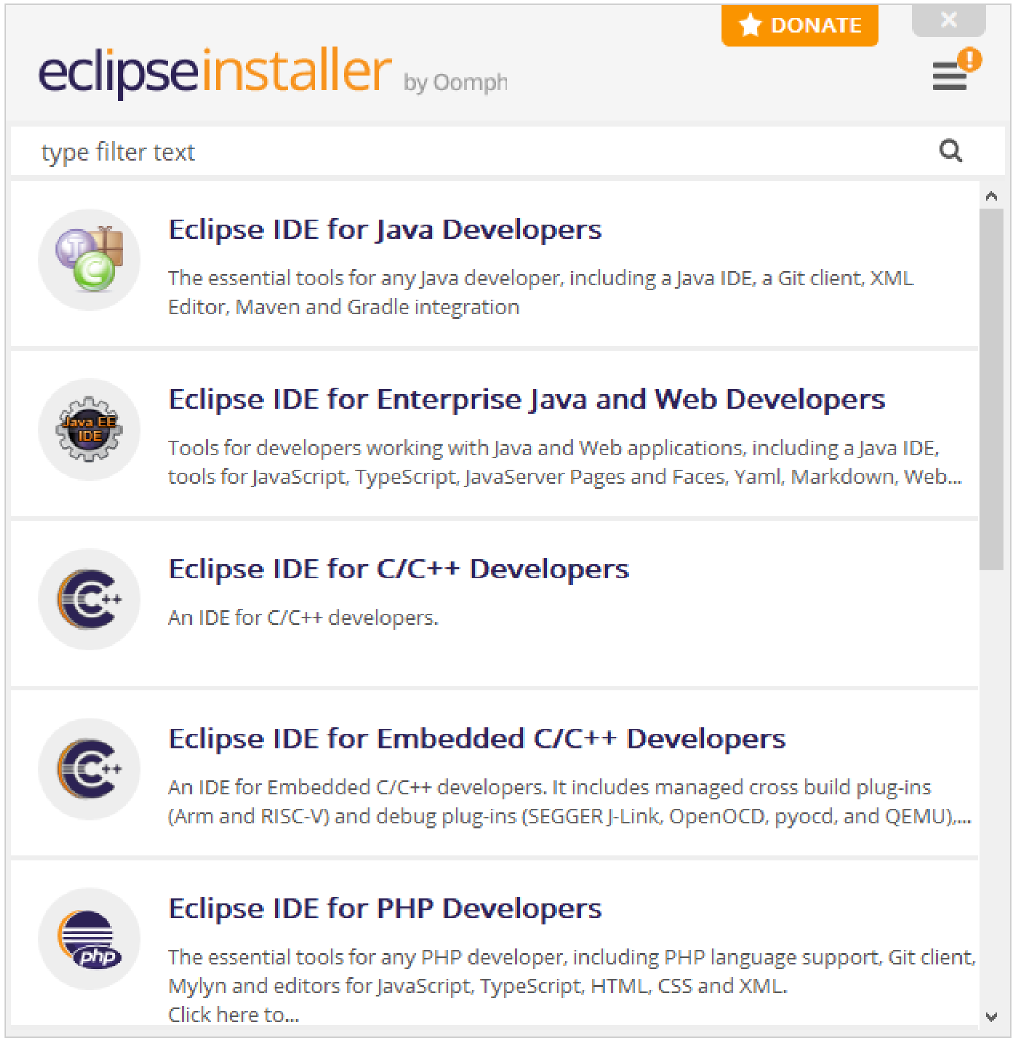
Name: D:\Installer\Program\Eclipse\eclipse-inst-jre-win64.exe
Publisher: **Eclipse.org Foundation, Inc.**
Type: Application
From: D:\Installer\Program\Eclipse\eclipse-inst-jre-win64.exe

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Installing (continued)

- Pick Eclipse IDE for Java Developers from the list and perform the installation.



Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration

Eclipse IDE for Enterprise Java and Web Developers
Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web...

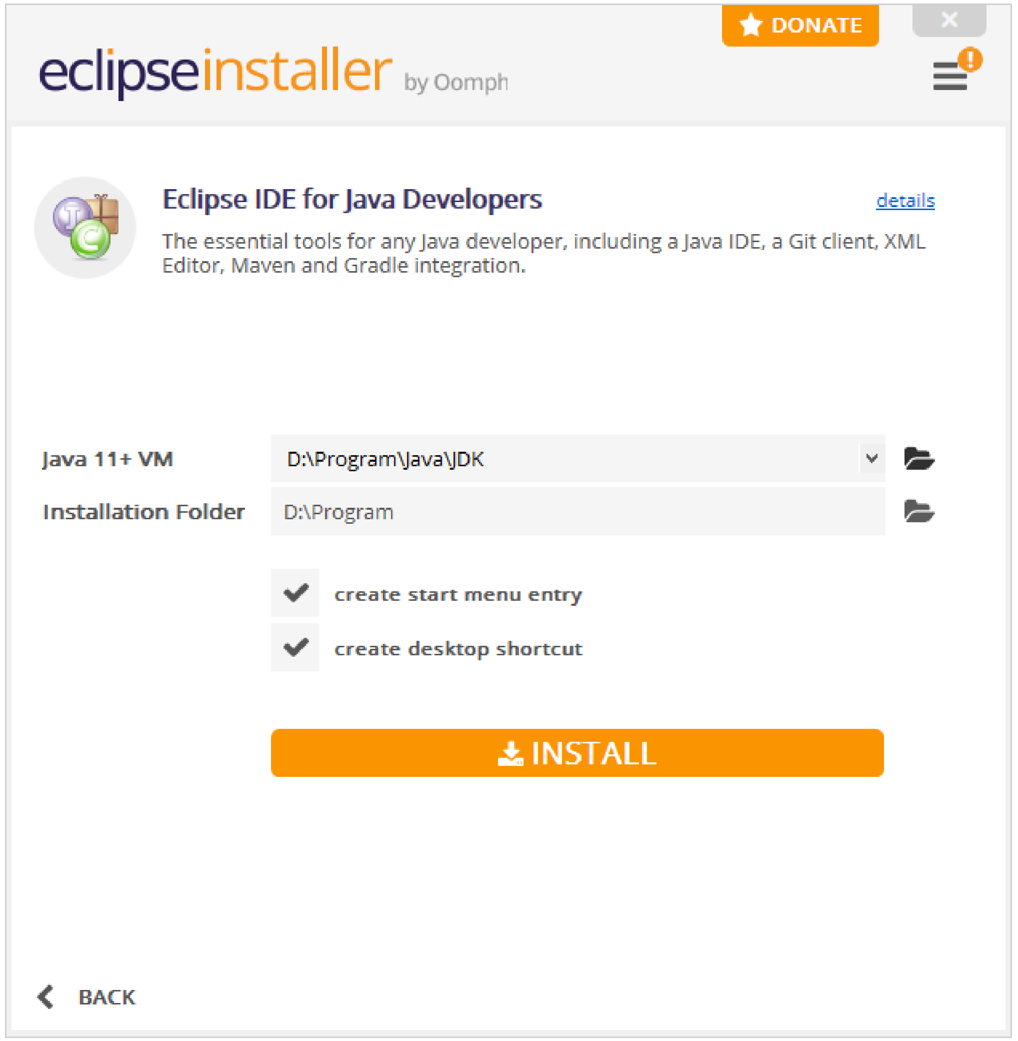
Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers.

Eclipse IDE for Embedded C/C++ Developers
An IDE for Embedded C/C++ developers. It includes managed cross build plug-ins (Arm and RISC-V) and debug plug-ins (SEGGER J-Link, OpenOCD, pyocd, and QEMU),...

Eclipse IDE for PHP Developers
The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, TypeScript, HTML, CSS and XML.
Click here to...

Installing (continued)

- Fill an intended parent folder where the eclipse folder will be created
- Click the Install button



Eclipse IDE for Java Developers

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration.

Java 11+ VM: D:\Program\Java\JDK

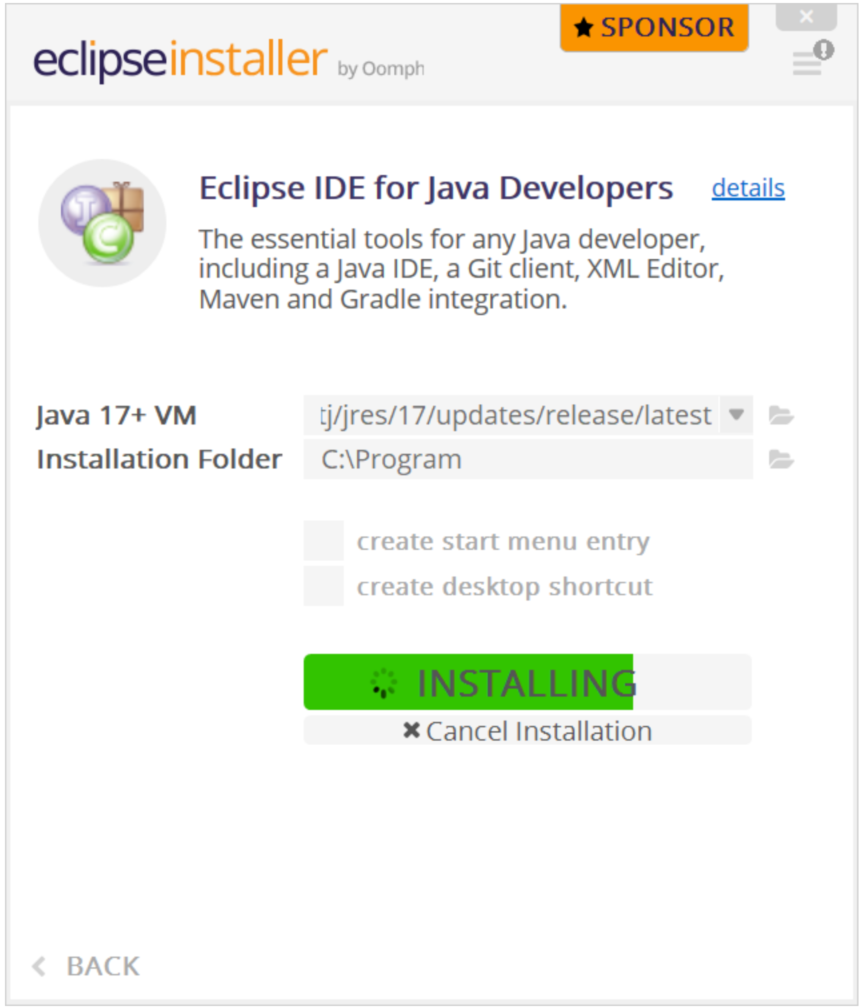
Installation Folder: D:\Program

✓ create start menu entry

✓ create desktop shortcut

Installing (continued)

- Wait until the installation finishes



Eclipse IDE for Java Developers details

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration.

Java 17+ VM: tj/jres/17/updates/release/latest

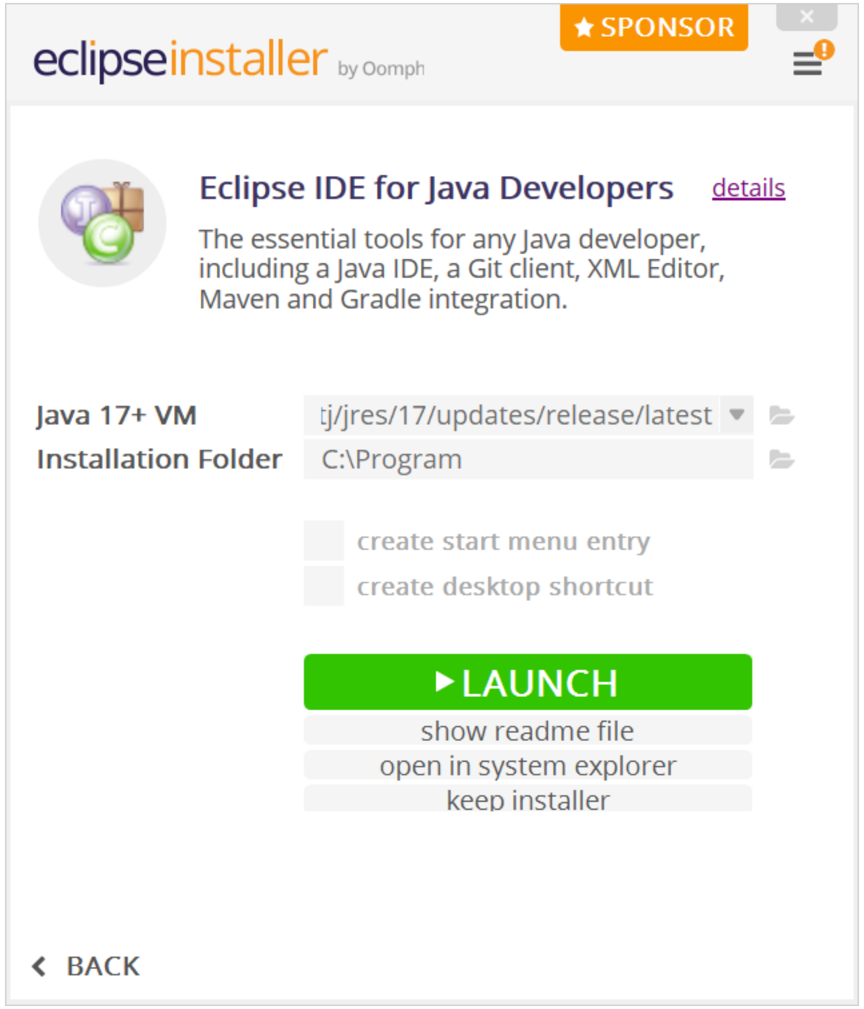
Installation Folder: C:\Program

create start menu entry

create desktop shortcut

Installing (continued)

- Once it's finished, click the Launch button



eclipseinstaller by Oomph

Eclipse IDE for Java Developers

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration.

Java 17+ VM: tj/jres/17/updates/release/latest

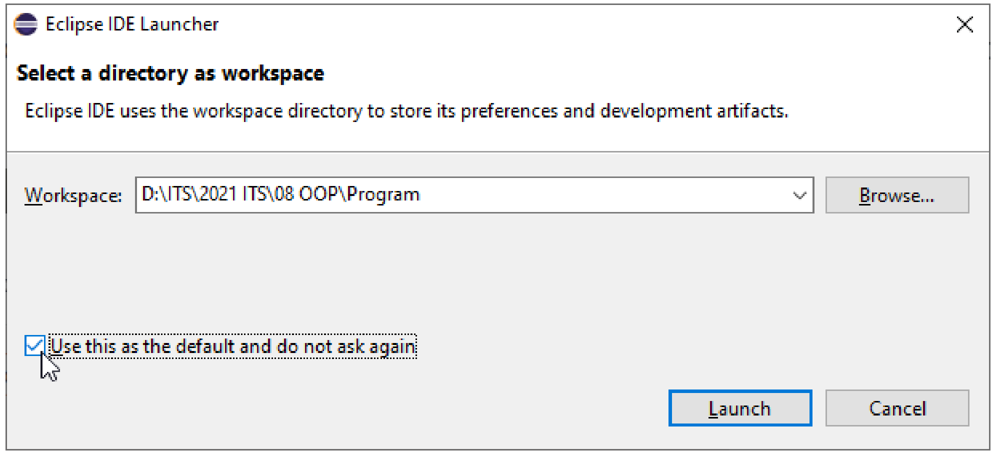
Installation Folder: C:\Program

Running

- The current version will be shown

Workspace setting

- It prompts you for a workspace to store it configuration
- Select an intended/empty folder
  - You may use this folder as the default one
- Click the Launch button


Eclipse IDE Launcher

Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: D:\ITS\2021 ITS\08 OOP\Program

[✓] Use this as the default and do not ask again

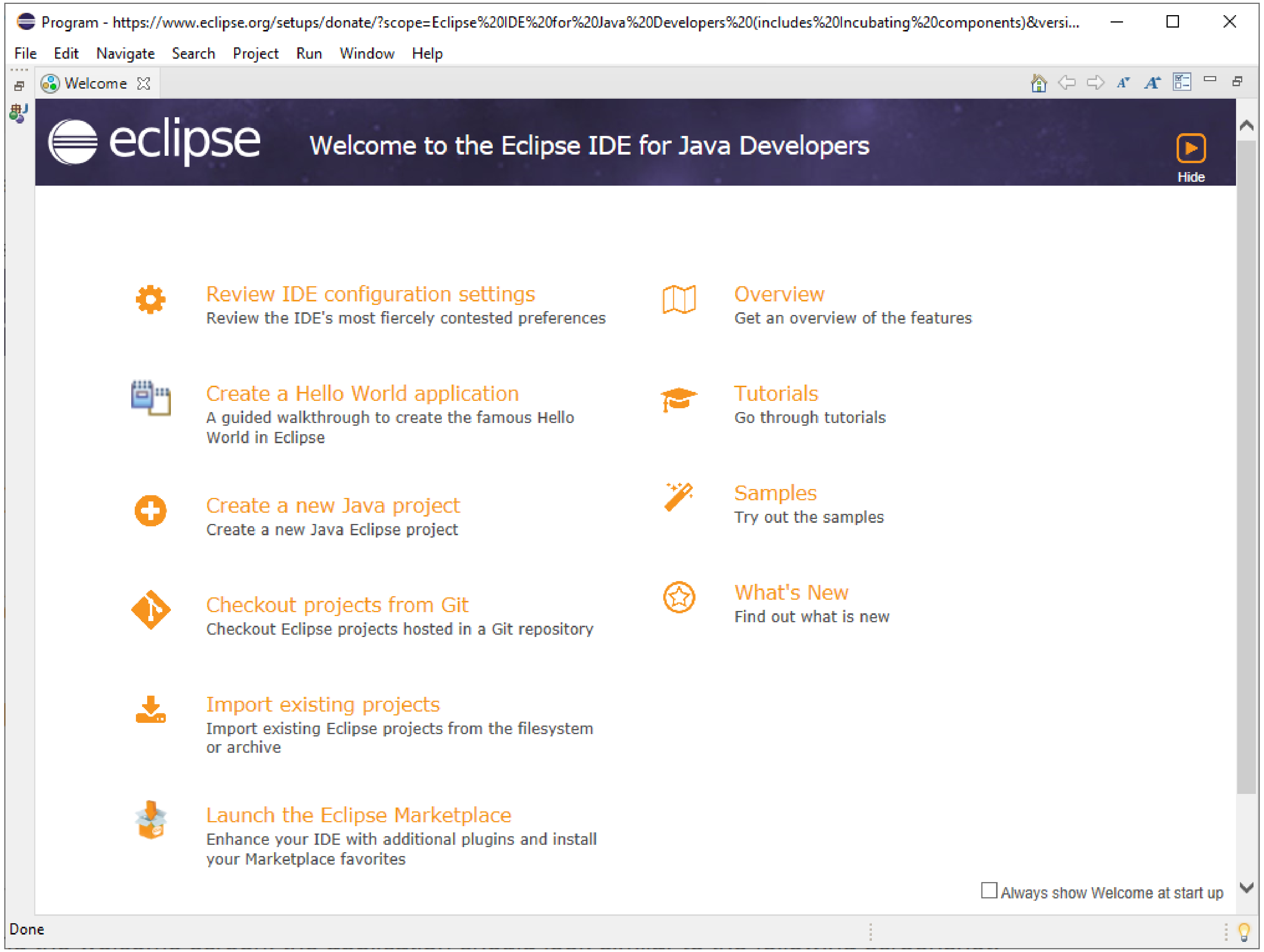
Starting

- Eclipse starts

- … and show the Welcome page.

Welcome page

- Close this page by clicking the x beside Welcome


Program - https://www.eclipse.org/setups/donate/?scope=Eclipse%20IDE%20for%20Java%20Developers%20(includes%20Incubating%20components)&versi...

Welcome to the Eclipse IDE for Java Developers

Review IDE configuration settings
Review the IDE's most fiercely contested preferences

Create a Hello World application
A guided walkthrough to create the famous Hello World in Eclipse

Create a new Java project
Create a new Java Eclipse project

Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository

Import existing projects
Import existing Eclipse projects from the filesystem or archive

Launch the Eclipse Marketplace
Enhance your IDE with additional plugins and install your Marketplace favorites

Overview
Get an overview of the features

Tutorials
Go through tutorials

Samples
Try out the samples

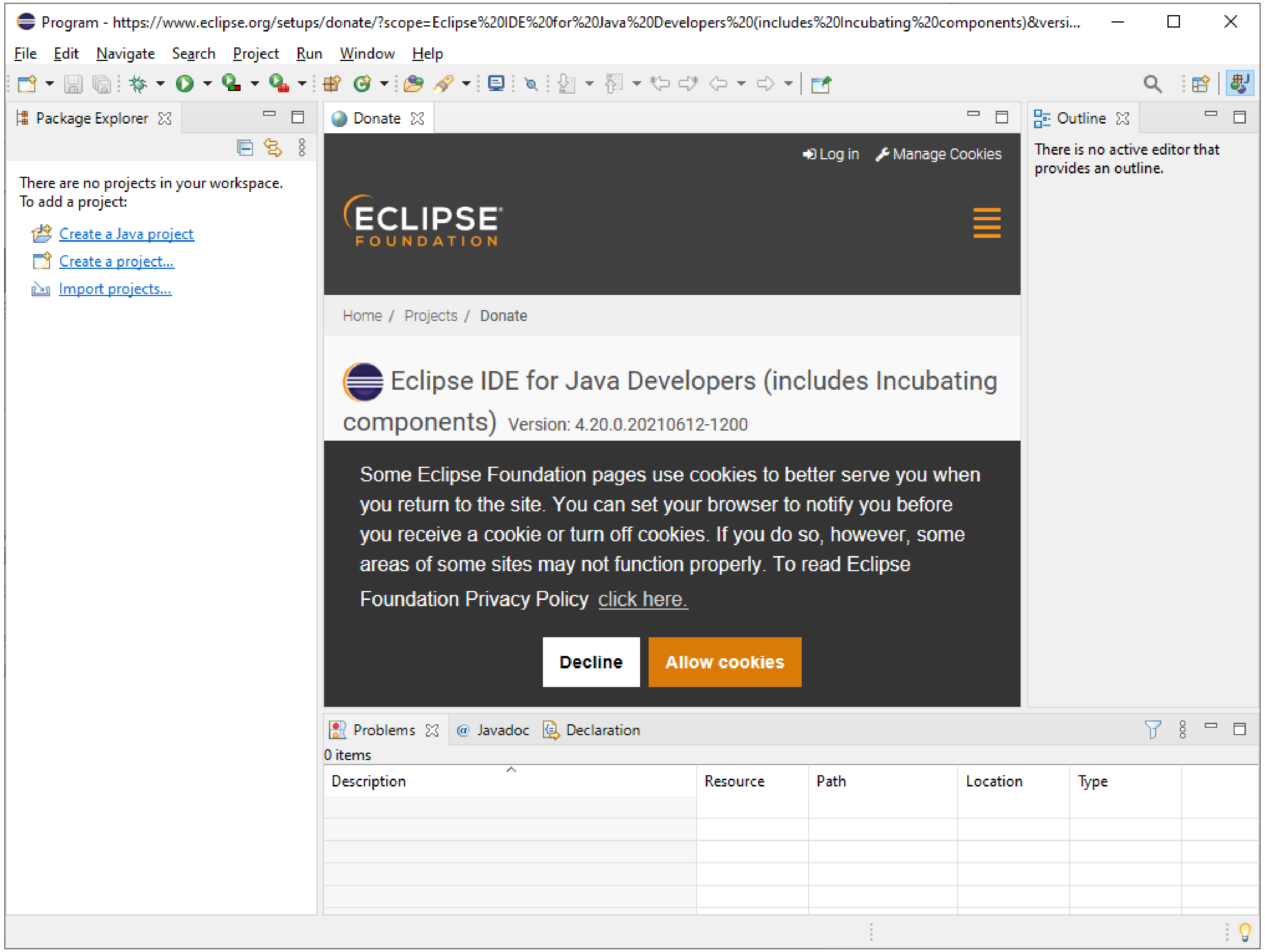
What's New
Find out what is new

☐ Always show Welcome at start up

Done

Donate page

- After closing the welcome screen, the app should look like this
- Close the tab Donate by clicking the x beside Donate



Program - https://www.eclipse.org/setups/donate/?scope=Eclipse%20IDE%20for%20Java%20Developers%20(includes%20Incubating%20components)&versi...

There are no projects in your workspace.
To add a project:

Eclipse IDE for Java Developers (includes Incubating components) Version: 4.20.0.20210612-1200

Some Eclipse Foundation pages use cookies to better serve you when you return to the site. You can set your browser to notify you before you receive a cookie or turn off cookies. If you do so, however, some areas of some sites may not function properly. To read Eclipse Foundation Privacy Policy click here.

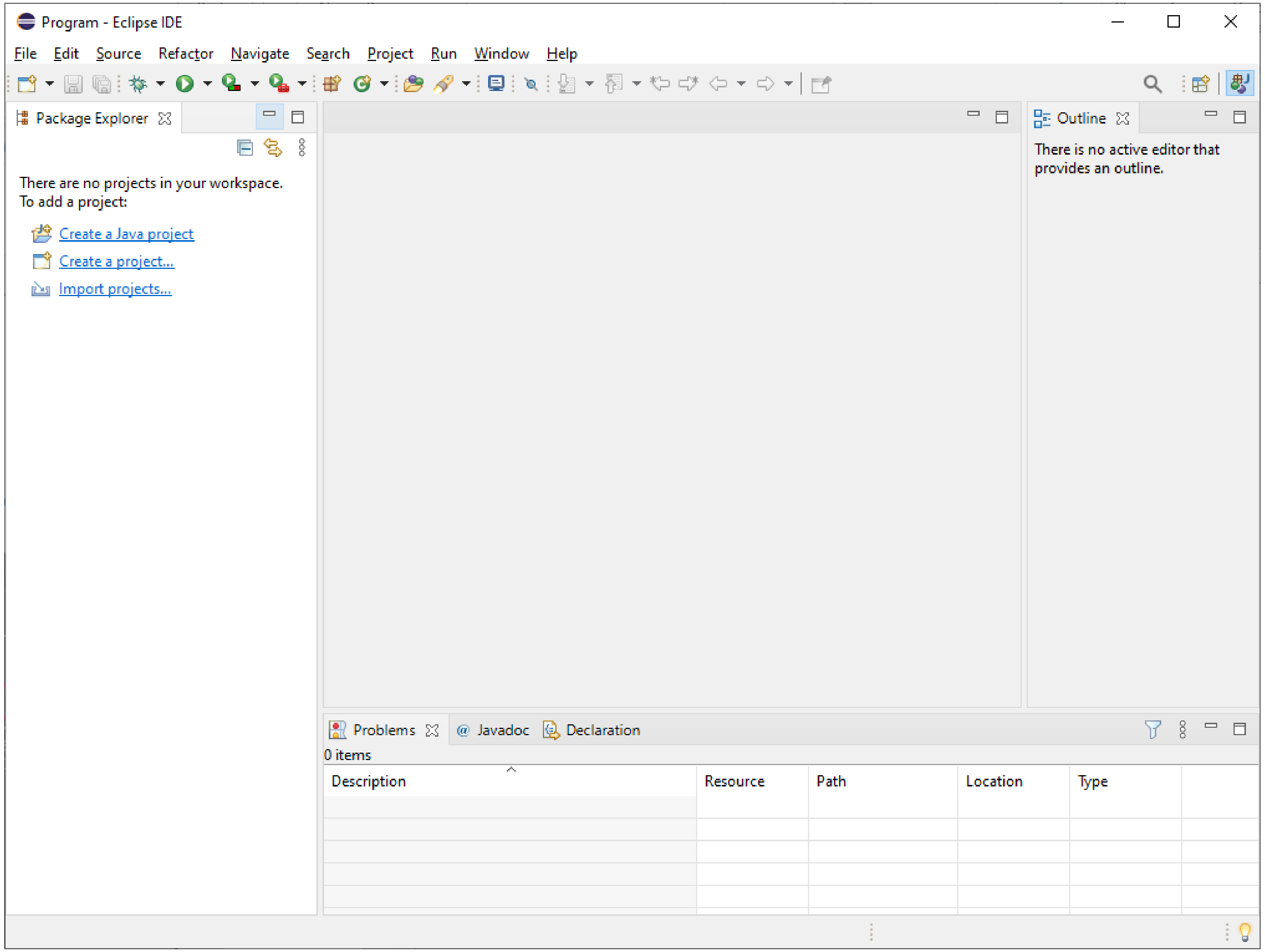
There is no active editor that provides an outline.

0 items

Description	Resource	Path	Location	Type

Ready to go!

- After closing the tab Donate, the app should look like this



There are no projects in your workspace.
To add a project:

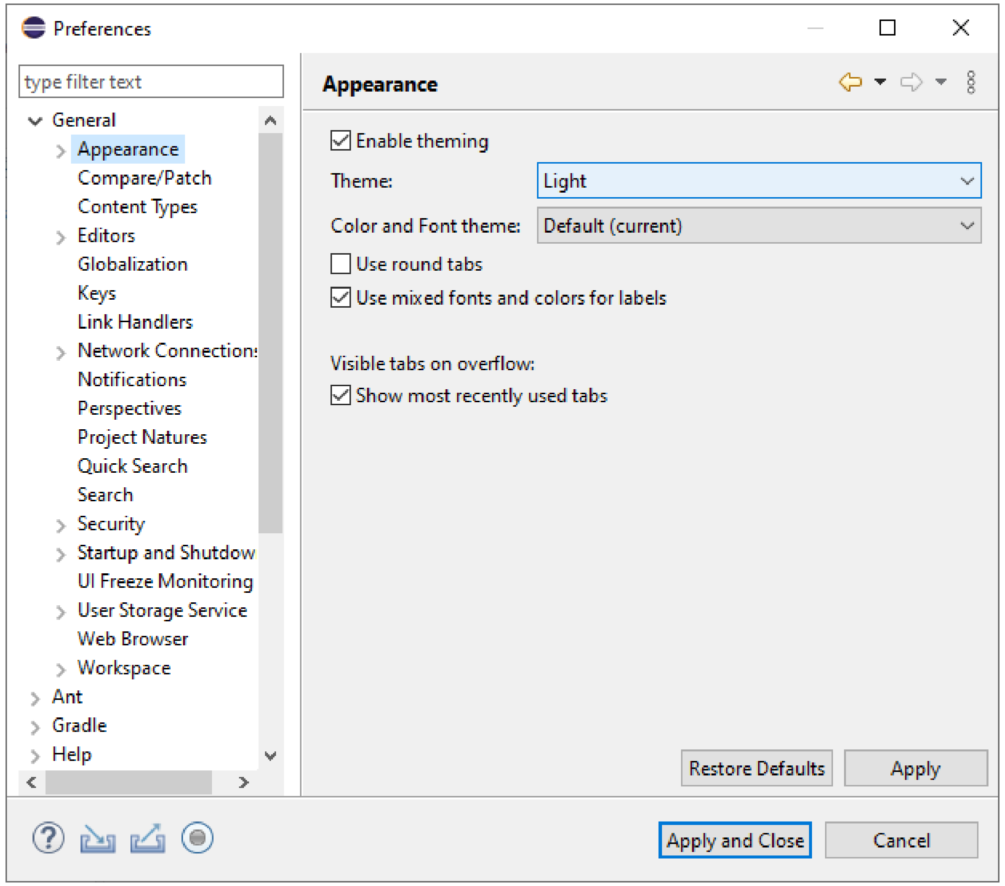
There is no active editor that provides an outline.

0 items

Description	Resource	Path	Location	Type

Appearance

- By default, Eclipse ships in a light configuration
- If you prefer to switch to a Classic/Dark/System, you can change it via **Window > Preferences > General > Appearance** menu
    - Restart your IDE afterwards, since some native OS styling functionality requires a start



Appearance

☑ Enable theming

Theme: Light

Color and Font theme: Default (current)

☐ Use round tabs

☑ Use mixed fonts and colors for labels

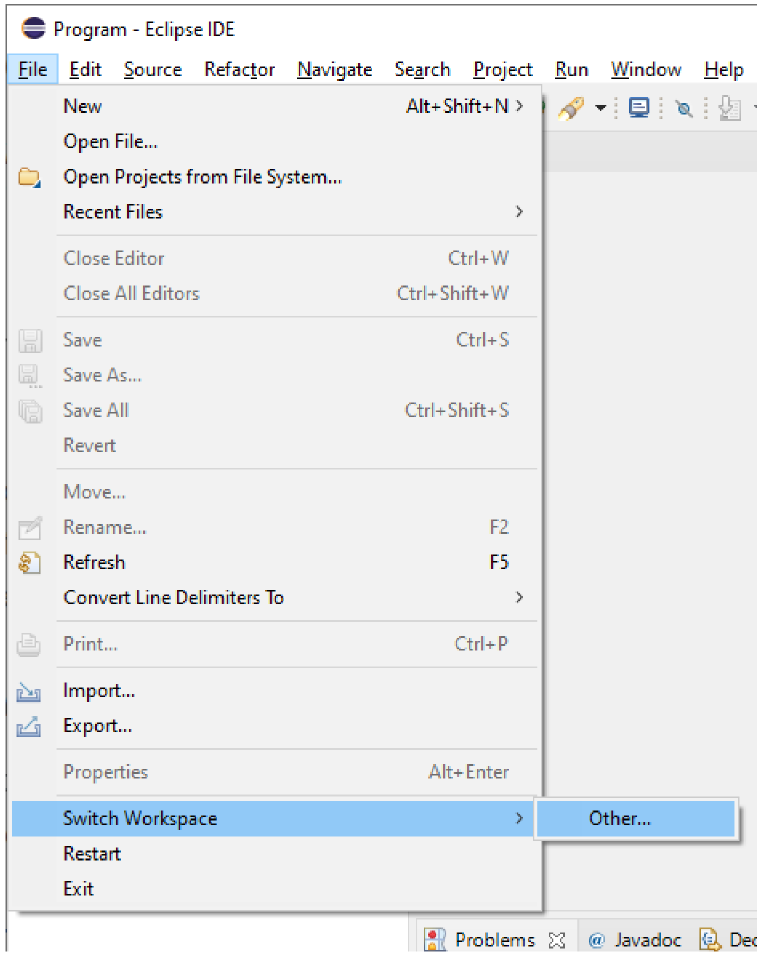
Visible tabs on overflow:

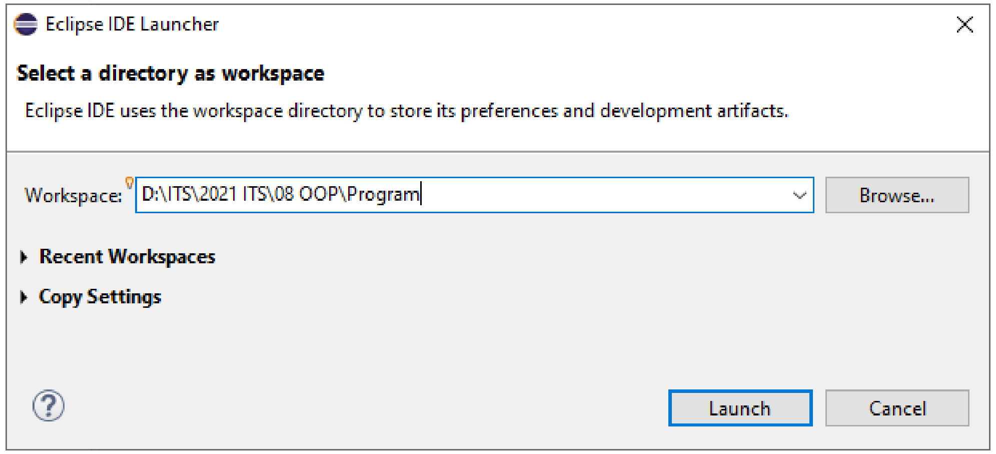
☑ Show most recently used tabs

Workspace and projects

- Workspace is the physical location (file path) for storing meta-data and (optional) your development artifacts.
  - The meta-data stored for the workspace contains preferences settings, plug-in specific meta data, logs etc.
- You can choose the workspace during startup of Eclipse or via the **File > Switch Workspace > Others** menu entry.
- Your projects, source files, images and other artifacts can be stored inside or outside your workspace.
  - For example, if you use Git as version control system, you typically would store the Git repositories outside of the workspace.

Workspace and projects (continued)





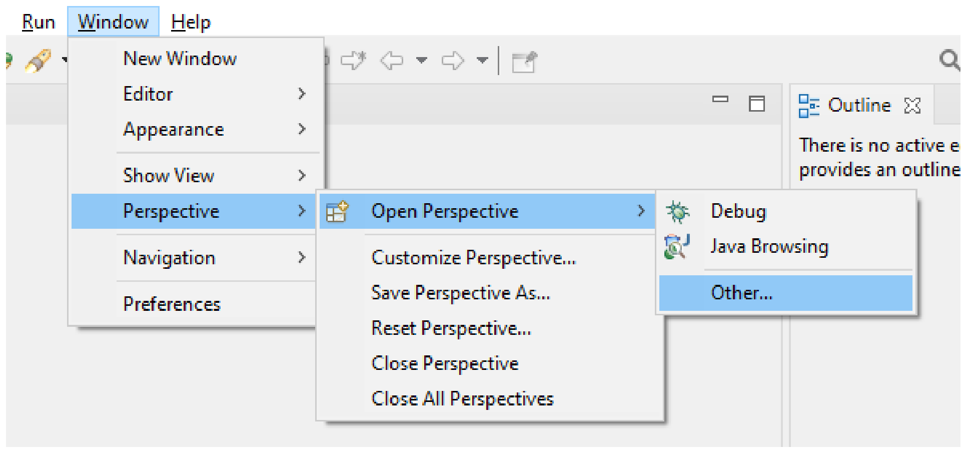
Select a directory as workspace

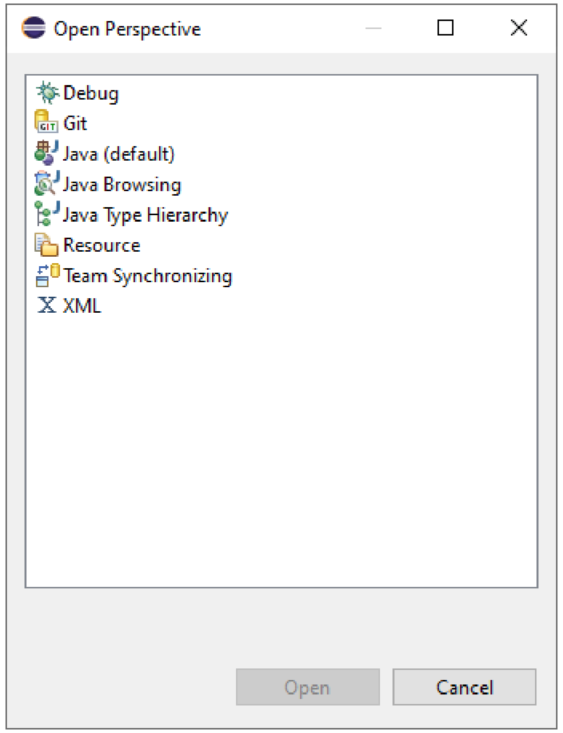
Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: D:\ITS\2021 ITS\08 OOP\Program

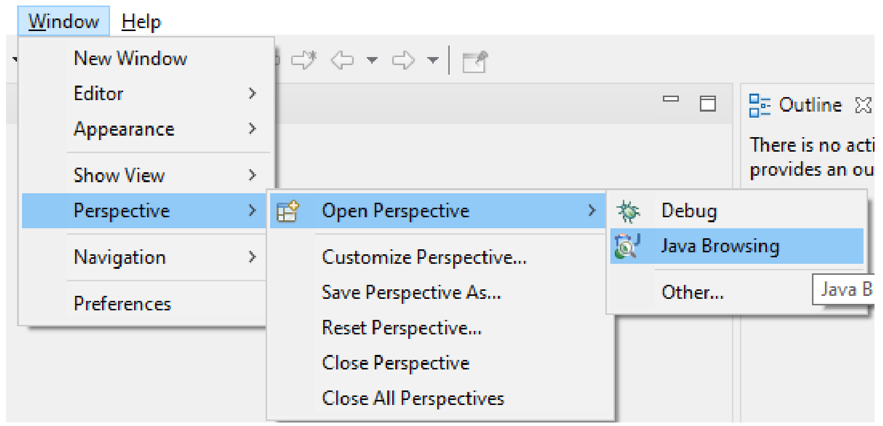
User Interface

- Eclipse provides *views* and *editors* to navigate and change content.
- View and editors can be grouped into *perspectives*.
- Eclipse provides different perspectives for different tasks. The available perspectives depend on your installation. For Java development you usually use the *Java Perspective*, but Eclipse has much more predefined perspectives, e.g., the *Debug* perspective.
- Eclipse allows you to switch to another perspective via the **Window > Perspective > Open Perspective > Other…**





User Interface (continued)



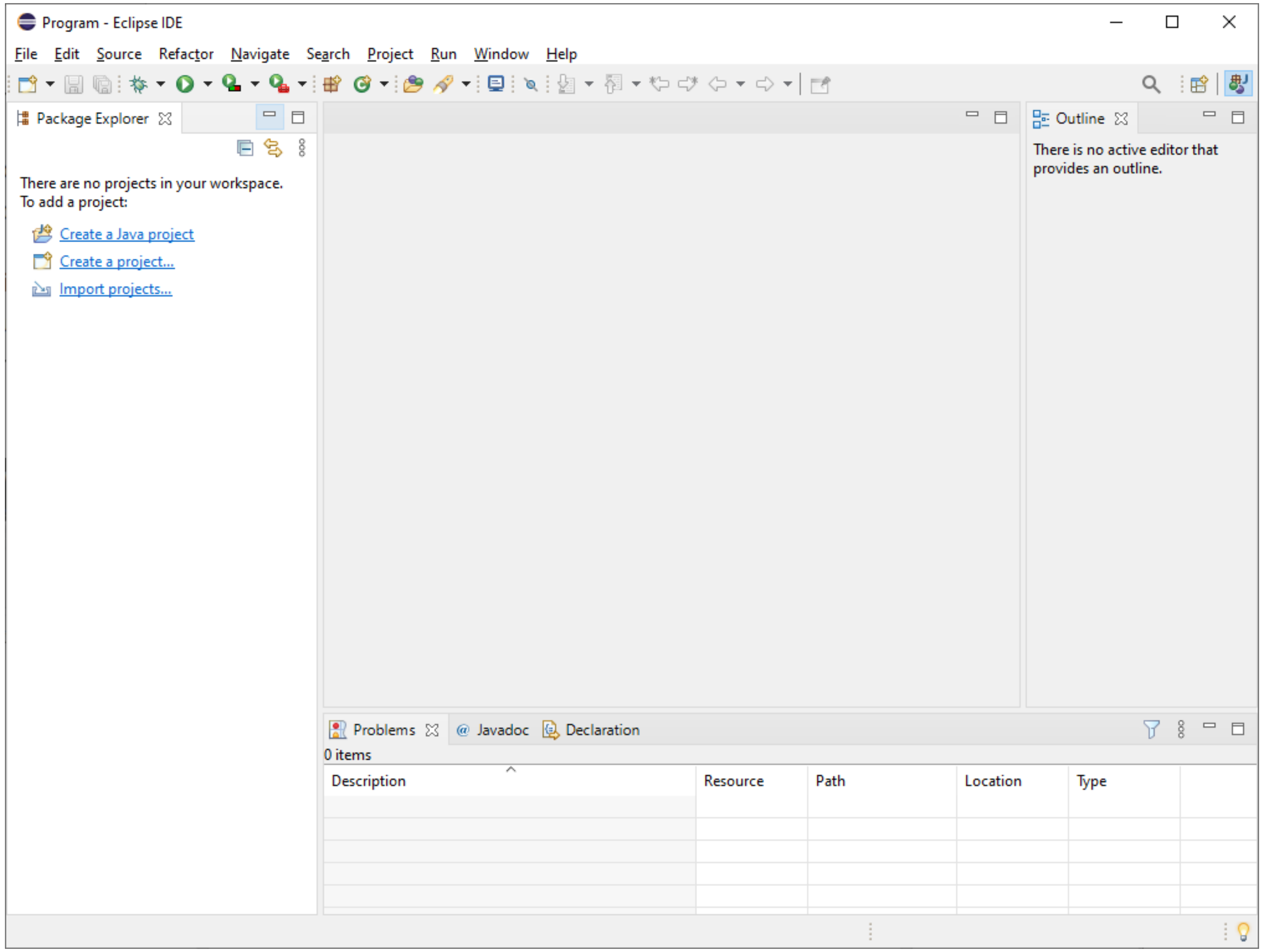
- Open editors are typically shared between perspectives, i.e., if you have an editor open in the *Java* perspective for a certain class and switch to the *Debug* perspective, this editor stays open.
- You can switch perspectives via the **Window > Perspective** > **Open Perspective > Other...**
- The main perspectives used for Java development are the *Java* perspective and the *Debug* perspective.
- The Java perspective can be opened via **Window > Perspective** > **Open Perspective** > **Java**.

User Interface (continued)

- On the left hand side, this perspective shows the *Package Explorer* view, which allows you to browse your projects and to select the components you want to open in an editor via a double-click.
- For example, to open a Java source file, open the tree under `src`, select the corresponding `.java` file and double-click it. This will open the file in the default Java editor.
- The following picture shows the default Java perspective. The *Package Explorer* view is on the left. In the middle you see the open editors. Several editors are stacked in the same container and you can switch between them by clicking on the corresponding tab. Via drag and drop you can move an editor to a new position in the Eclipse IDE.

User Interface (continued)

- To the right and below the editor area you find more views which were considered useful by the developer of the perspective. For example, the *Javadoc* view shows the Javadoc of the selected class or method.



There are no projects in your workspace.
To add a project:

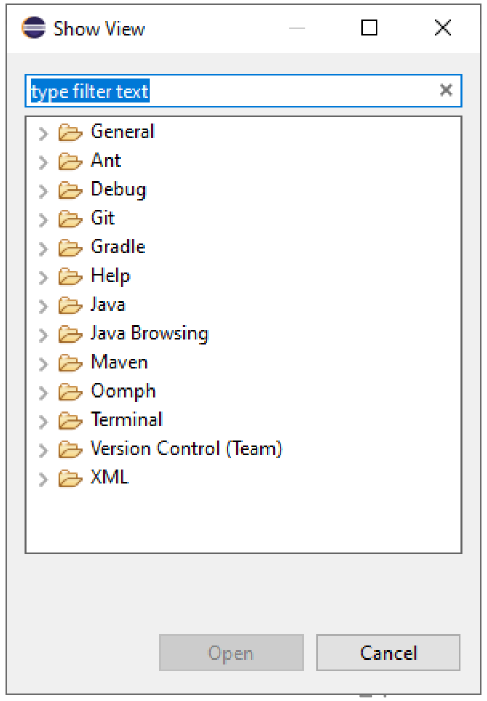
There is no active editor that provides an outline.

0 items

Description	Resource	Path	Location	Type

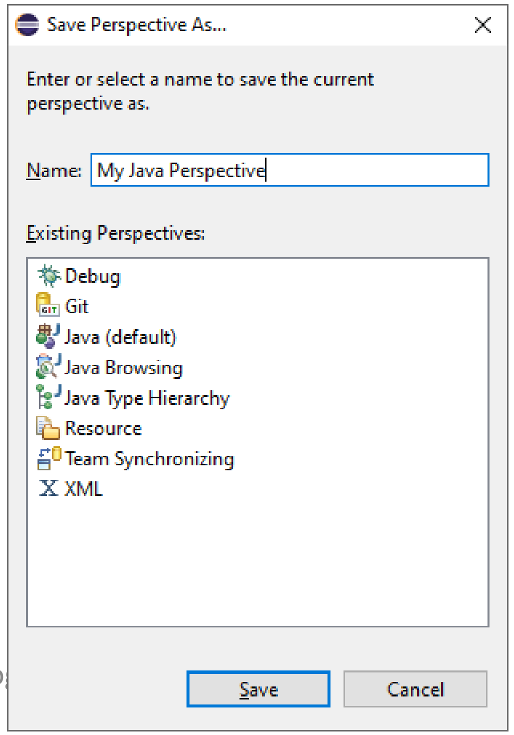
User Interface (continued)

- Resetting and customizing a perspective
  - A common problem is that you changed the arrangement of views and editors in your perspective and you want to restore its original state. For example, you might have closed a view. You can reset a perspective to its original state via the **Window** > **Perspective** > **Reset Perspective...**
  - You can change the layout and content within a perspective by opening or closing parts and by re-arranging them.
  - To open a new part in your current perspective, use the **Window** > **Show View** > **Other...** . This opens the *Show View* dialog which allows you to search for certain parts.



User Interface (continued)

- If you want to reset your current perspective to its default, use the **Window** > **Perspective** > **Reset Perspective…**
- You can save the currently selected perspective via **Window > Perspective** > **Save Perspective As…**



Enter or select a name to save the current perspective as.

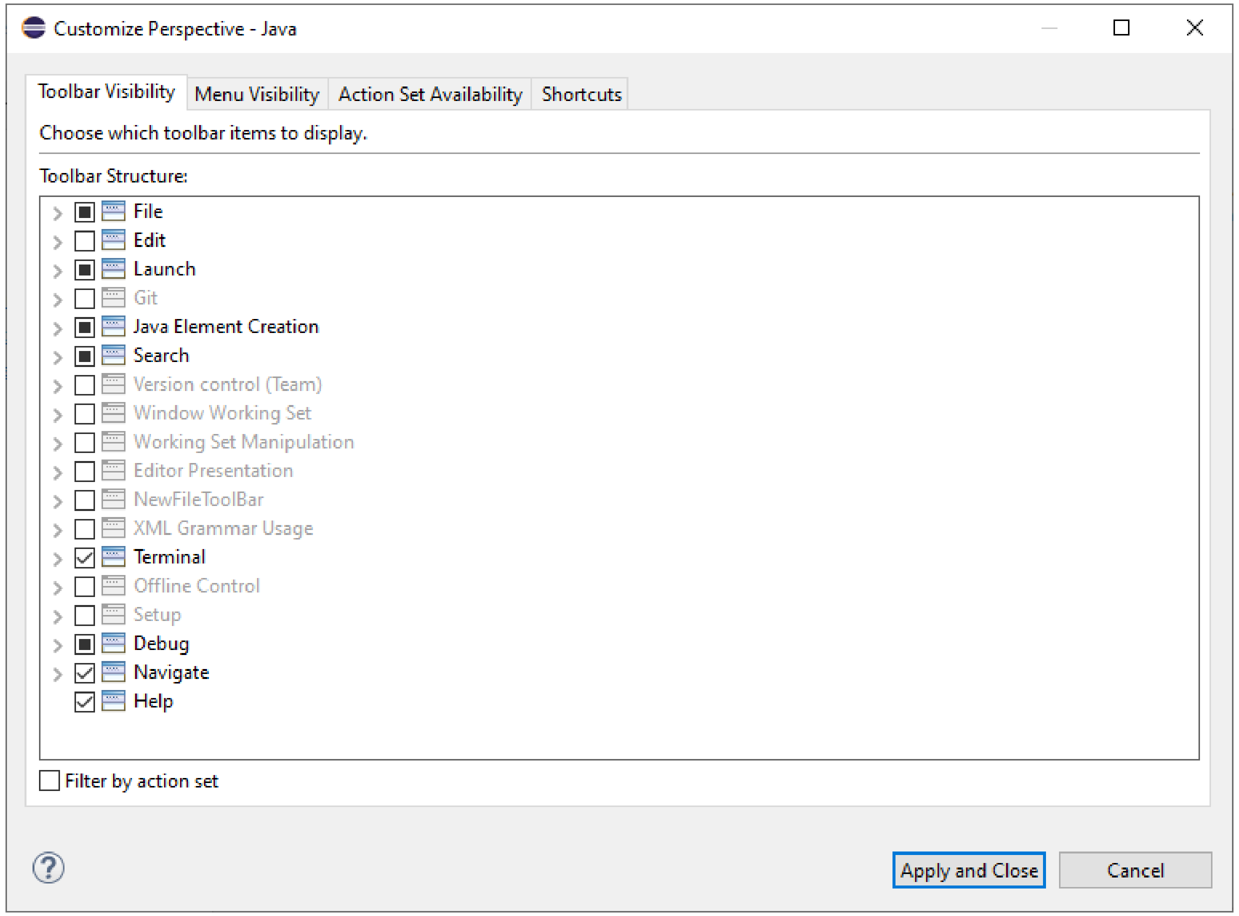
Name: My Java Perspective

Existing Perspectives:

- Debug
- Git
- Java (default)
- Java Browsing
- Java Type Hierarchy
- Resource
- Team Synchronizing
- XML

User Interface (continued)

- The **Window > Perspective > Customize Perspective…** menu entry allows you to adjust the selected perspective. For example, you can hide or show toolbar and menu entries.


Customize Perspective - Java

Toolbar Visibility | Menu Visibility | Action Set Availability | Shortcuts

Choose which toolbar items to display.

Toolbar Structure:

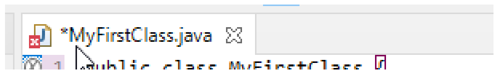
- ■ File
- ☐ Edit
- ■ Launch
- ☐ Git
- ■ Java Element Creation
- ■ Search
- ☐ Version control (Team)
- ☐ Window Working Set
- ☐ Working Set Manipulation
- ☐ Editor Presentation
- ☐ NewFileToolBar
- ☐ XML Grammar Usage
- ☑ Terminal
- ☐ Offline Control
- ☐ Setup
- ■ Debug
- ☑ Navigate
- ☑ Help

☐ Filter by action set

Apply and Close | Cancel

User Interface (continued)

- A view is typically used to display structured data and allow to modify it directly.
- For example, the *Project Explorer* view allows you to browse and modify files of Eclipse projects. If you rename a file via the *Project Explorer* the file name is directly changed without having to save.
- Editors are typically used to modify a single data element, for example a text file. To apply these changes to the underlying data mode, you need to select save from the menu or the toolbar. An editor with unsaved data (a dirty editor) is marked with an asterisk (`*`) left to the name of the modified file. E.g., `*MyFirstClass.java`



```
public class MyFirstClass
```

User Interface (continued)

- Eclipse projects
  - An Eclipse project contains source, configuration and binary files related to a certain task. It groups them into buildable and reusable units. An Eclipse project can have *natures* assigned to it which describe the purpose of this project. For example, the Java *nature* defines a project as Java project. Projects can have multiple natures combined to model different technical aspects.
  - *Natures* for a project are defined via the `.project` file in the project directory.

Java perspective

- Package Explorer view

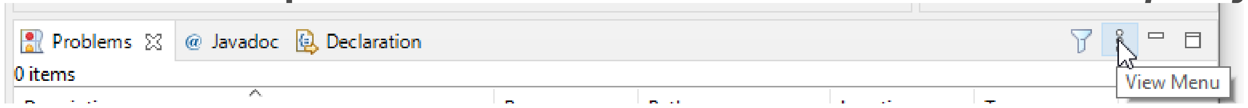
    - The *Package Explorer* view allows you to browse the structure of your projects and to open files in an *editor* via a double-click on the file.
    - It is also used to change the structure of your project. For example, you can rename files or move files and folders via drag and drop. A right-click on a file or folder shows you the available options.
- Outline view

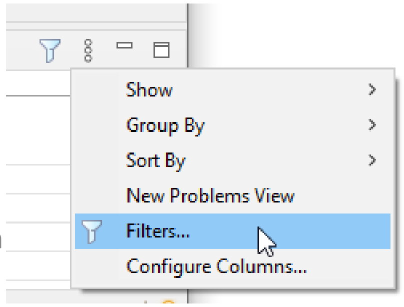
    - The *Outline* view shows the structure of the currently selected source file.

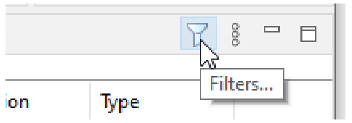
Java perspective (continued)

- Problems view

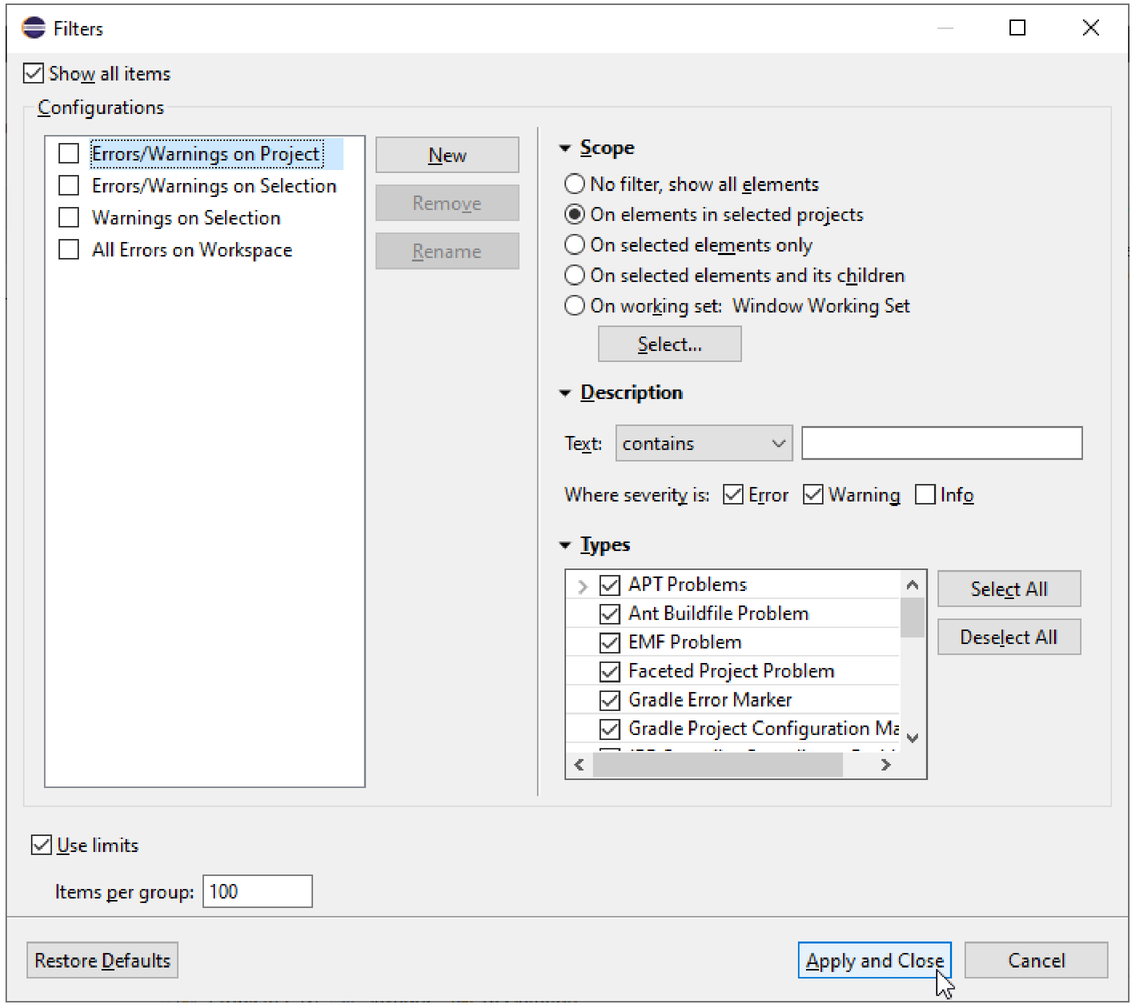
  - The *Problems* view shows errors and warning messages. Sooner or later you will run into problems with your code or your project setup. To view the problems in your project, you can use the *Problems* view which is part of the standard Java *perspective*. If this view is closed, you can open it via **Window** > **Show View** > **Problems**.
  - The messages which are displayed in the *Problems* view can be configured via the drop-down menu of the view. For example, to display the problems from the currently selected project, select *Filters…* and set the Scope to *On elements in selected projects*.







Java perspective (continued)



Filters

[✓] Show all items

Configurations

- [ ] Errors/Warnings on Project
- [ ] Errors/Warnings on Selection
- [ ] Warnings on Selection
- [ ] All Errors on Workspace

Scope

- ○ No filter, show all elements
- ● On elements in selected projects
- ○ On selected elements only
- ○ On selected elements and its children
- ○ On working set: Window Working Set

Description

Text: contains

Where severity is: [✓] Error [✓] Warning [ ] Info

Types

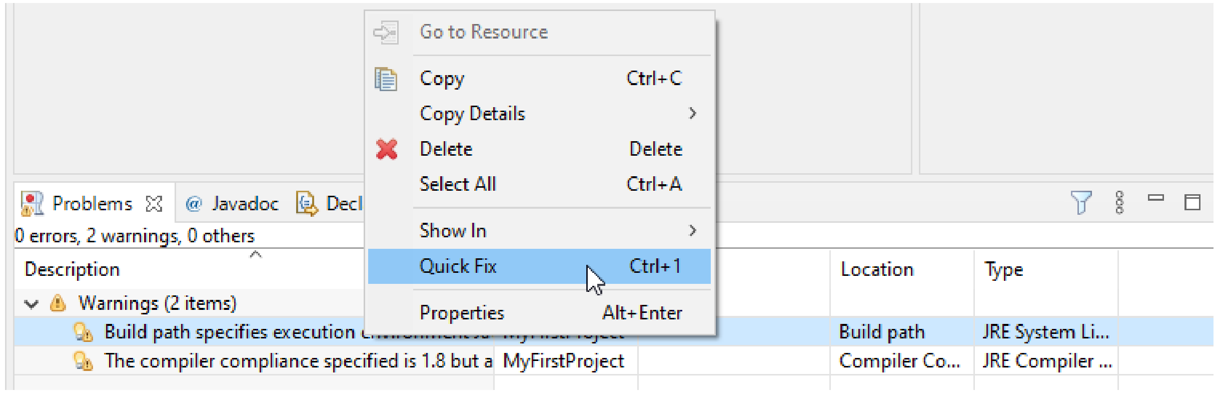
- [✓] APT Problems
- [✓] Ant Buildfile Problem
- [✓] EMF Problem
- [✓] Faceted Project Problem
- [✓] Gradle Error Marker
- [✓] Gradle Project Configuration Ma

[✓] Use limits

Items per group: 100

Java perspective (continued)

- The *Problems* view also allows you to trigger a *Quick fix* via a right mouse-click on several selected messages.

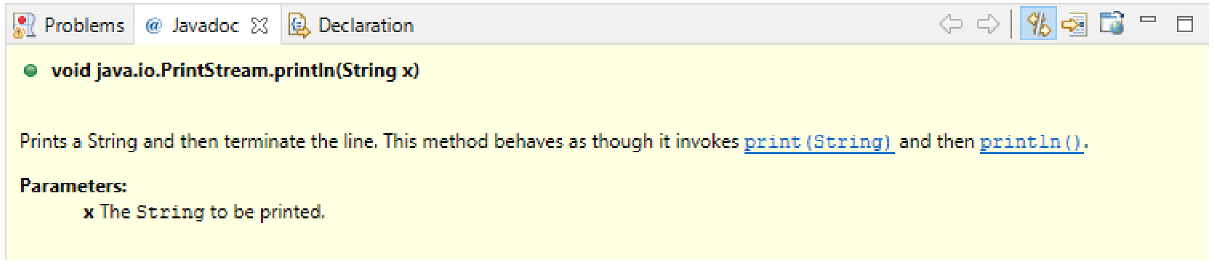


0 errors, 2 warnings, 0 others

Description			Location	Type
Warnings (2 items)				
Build path specifies execution environment...	MyFirstProject		Build path	JRE System Li...
The compiler compliance specified is 1.8 but a	MyFirstProject		Compiler Co...	JRE Compiler ...

Java perspective (continued)

- Javadoc view
  - The *Javadoc* view shows the documentation of the selected element in the Java *editor*.

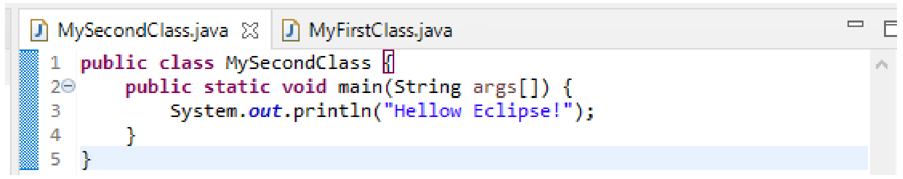


void java.io.PrintStream.println(String x)

Prints a String and then terminate the line. This method behaves as though it invokes `print(String)` and then `println()`.

Parameters:
x The `String` to be printed.

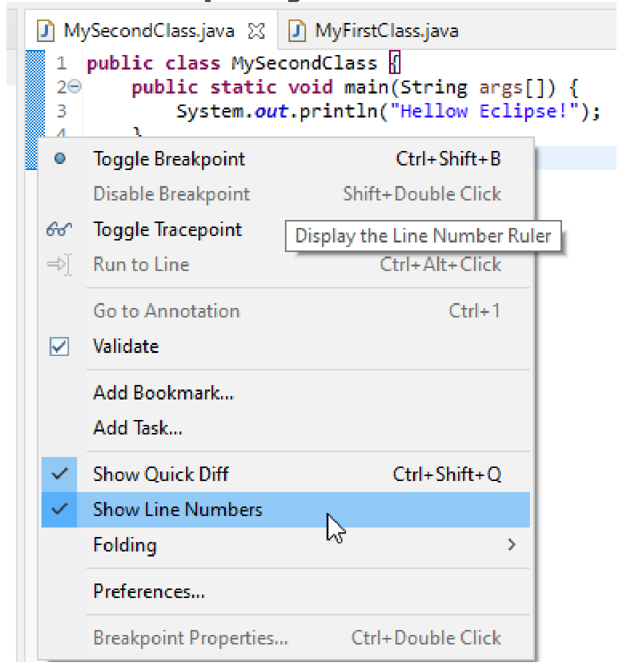
Java perspective (continued)



MySecondClass.java | MyFirstClass.java

```java
public class MySecondClass {
    public static void main(String args[]) {
        System.out.println("Hellow Eclipse!");
    }
}
```

- Java editor
  - The Java *editor* is used to modify the Java source code. Each Java source file is opened in a separate *editor*.
  - If you right click in the left column of the editor, you can configure its properties, for example, that line number should be displayed.



MySecondClass.java | MyFirstClass.java

```java
public class MySecondClass {
    public static void main(String args[]) {
        System.out.println("Hellow Eclipse!");
```

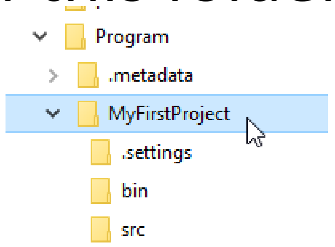
Toggle Breakpoint Ctrl+Shift+B
Disable Breakpoint Shift+Double Click
Toggle Tracepoint
Display the Line Number Ruler
Run to Line Ctrl+Alt+Click
Go to Annotation Ctrl+1
Validate
Add Bookmark...
Add Task...
Show Quick Diff Ctrl+Shift+Q
Show Line Numbers
Folding
Preferences...
Breakpoint Properties... Ctrl+Double Click

1st Java program

- Create project
  - Select **File > New > Java Project** from the menu. Enter `MyFirstProject` as the project name and press the **Finish** button to create the project.

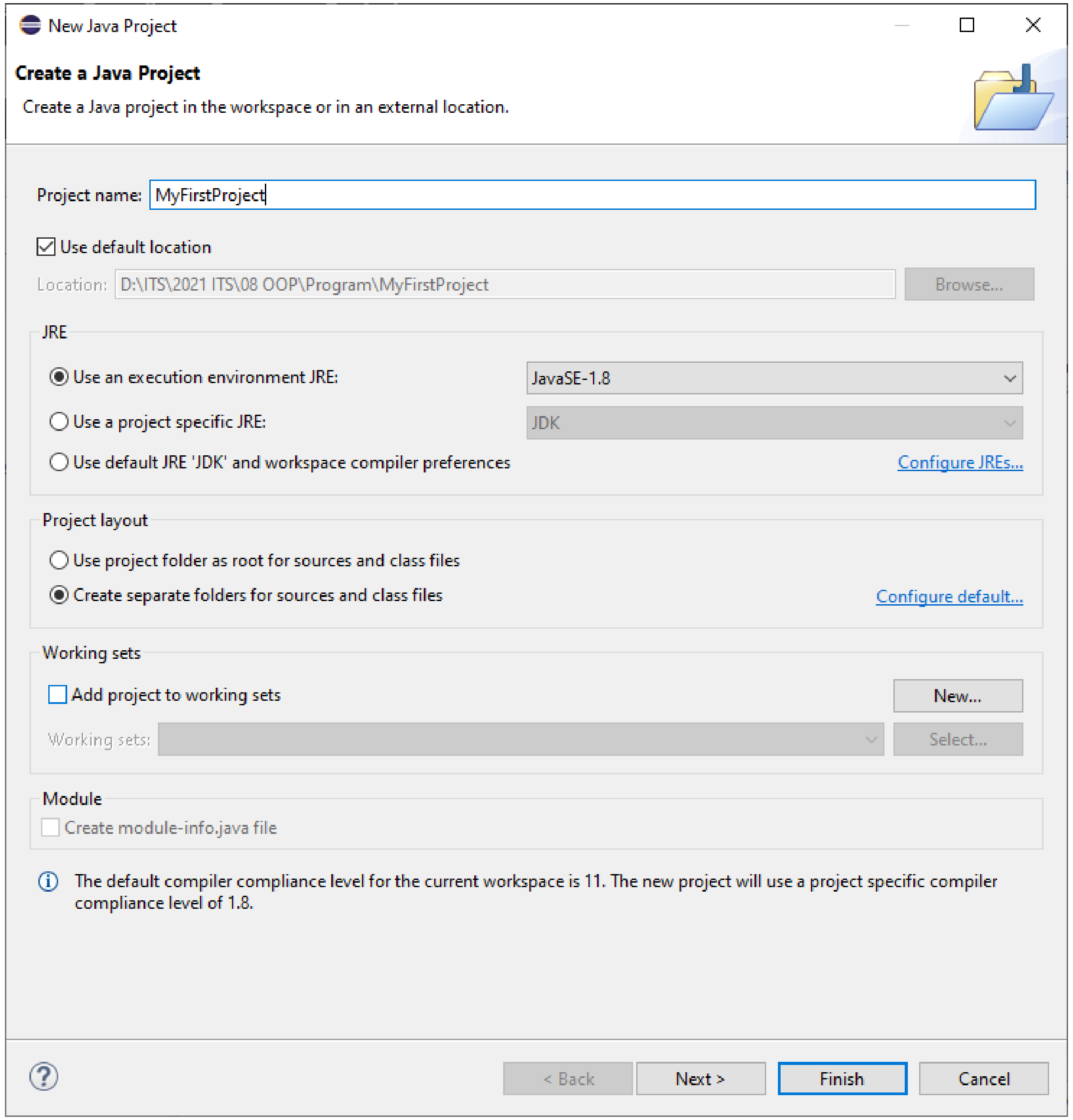
1st Java program (cont'd)

- A new project is created and displayed as a folder. Open the `MyFirstProject` folder and explore the content of this folder.



```
Program
  .metadata
  MyFirstProject
    .settings
    bin
    src
```

- The project is typically named the same as the top-level Java package in the project. This makes it easier to find a project related to a piece of code.



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: MyFirstProject

[✓] Use default location

Location: D:\ITS\2021 ITS\08 OOP\Program\MyFirstProject

JRE
- (●) Use an execution environment JRE: JavaSE-1.8
- ( ) Use a project specific JRE: JDK
- ( ) Use default JRE 'JDK' and workspace compiler preferences

Project layout
- ( ) Use project folder as root for sources and class files
- (●) Create separate folders for sources and class files

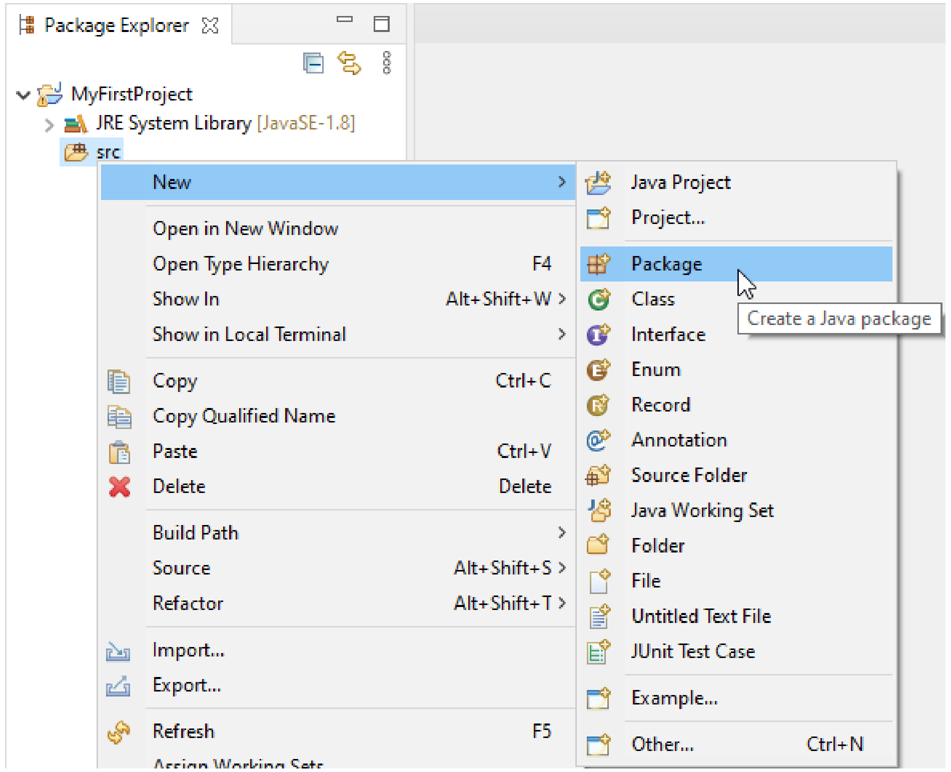
Working sets
- [ ] Add project to working sets

Module
- [ ] Create module-info.java file

The default compiler compliance level for the current workspace is 11. The new project will use a project specific compiler compliance level of 1.8.

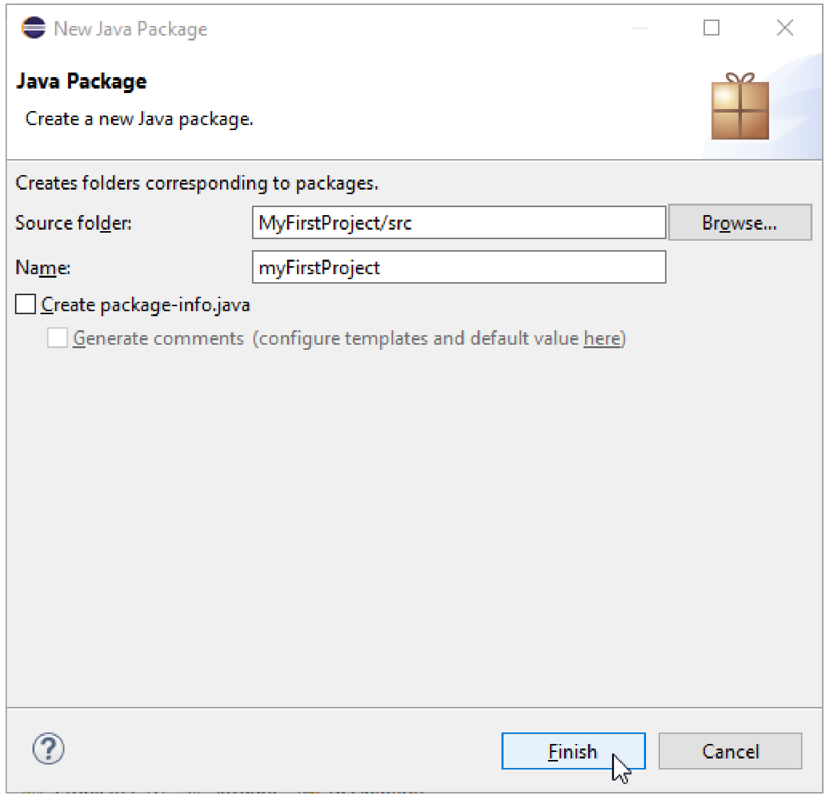
1^{st} Java program (cont'd)

- Create package
  - A good naming convention is to use the same name for the top level package and the project. For example, if you name your project `myproject` you should also use `myproject` as the top-level package name.
    - By convention, package names usually start with a lowercase letter
    - Since we have chosen `MyFirstProject` as our project name then we will use `myFirstProject` as our top level package
  - Create the `myFirstProject` package by selecting the `src` folder, right-click on it and select **New > Package**.



1st Java program (cont’d)

- Press the **Finish** button



New Java Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

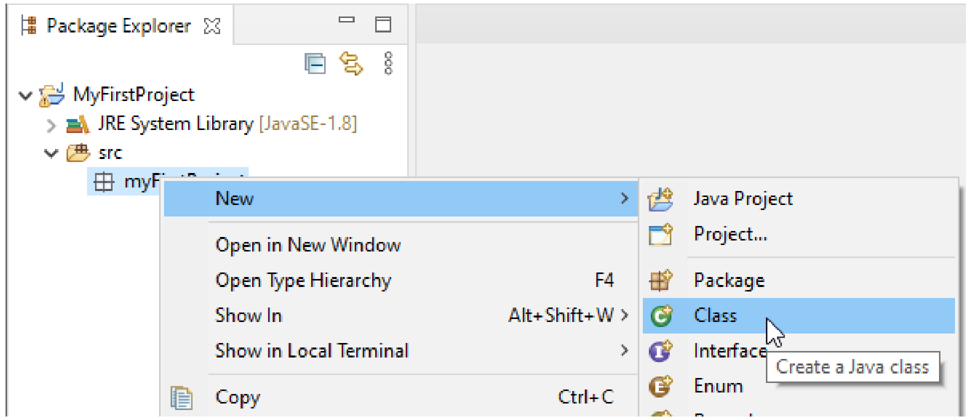
Source folder:	MyFirstProject/src
Name:	myFirstProject

[ ] Create package-info.java

[ ] Generate comments (configure templates and default value here)

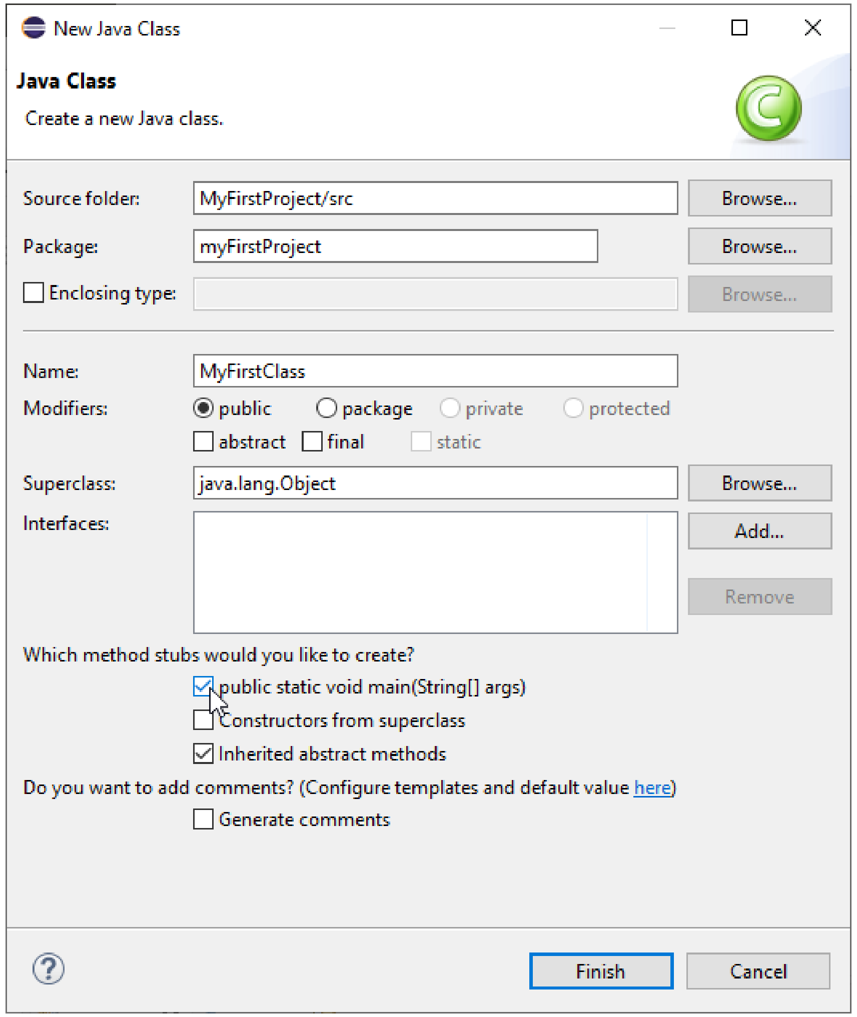
1st Java program (cont'd)

- Create Java class
  - Right-click on your package and select **New > Class** to create a Java class.



1st Java program (cont'd)

- Enter `MyFirstClass` as the class name and select the *public static void main (String[] args)* checkbox.
- Click the **Finish** button

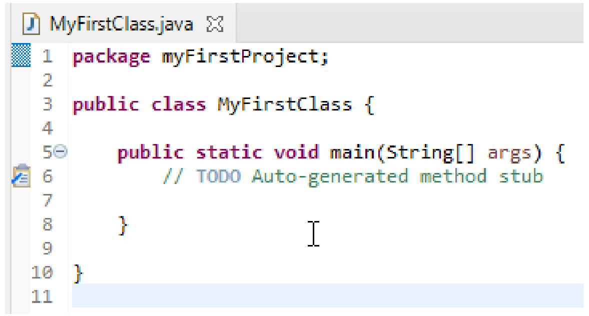


Source folder: MyFirstProject/src
Package: myFirstProject
[ ] Enclosing type:
Name: MyFirstClass
Modifiers: [✓ public] package private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
[✓] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
[ ] Generate comments

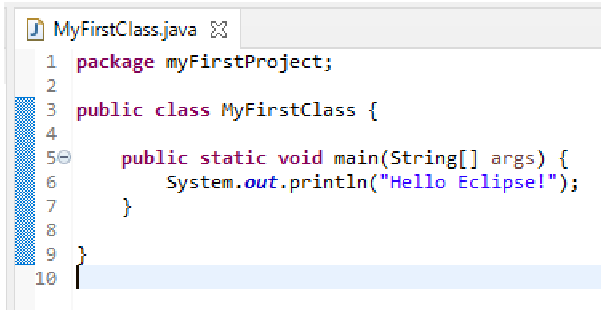
1st Java program (cont'd)

- This creates a new file and opens the Java editor. Change the class based on the following listing.



MyFirstClass.java

```
package myFirstProject;

public class MyFirstClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}

```



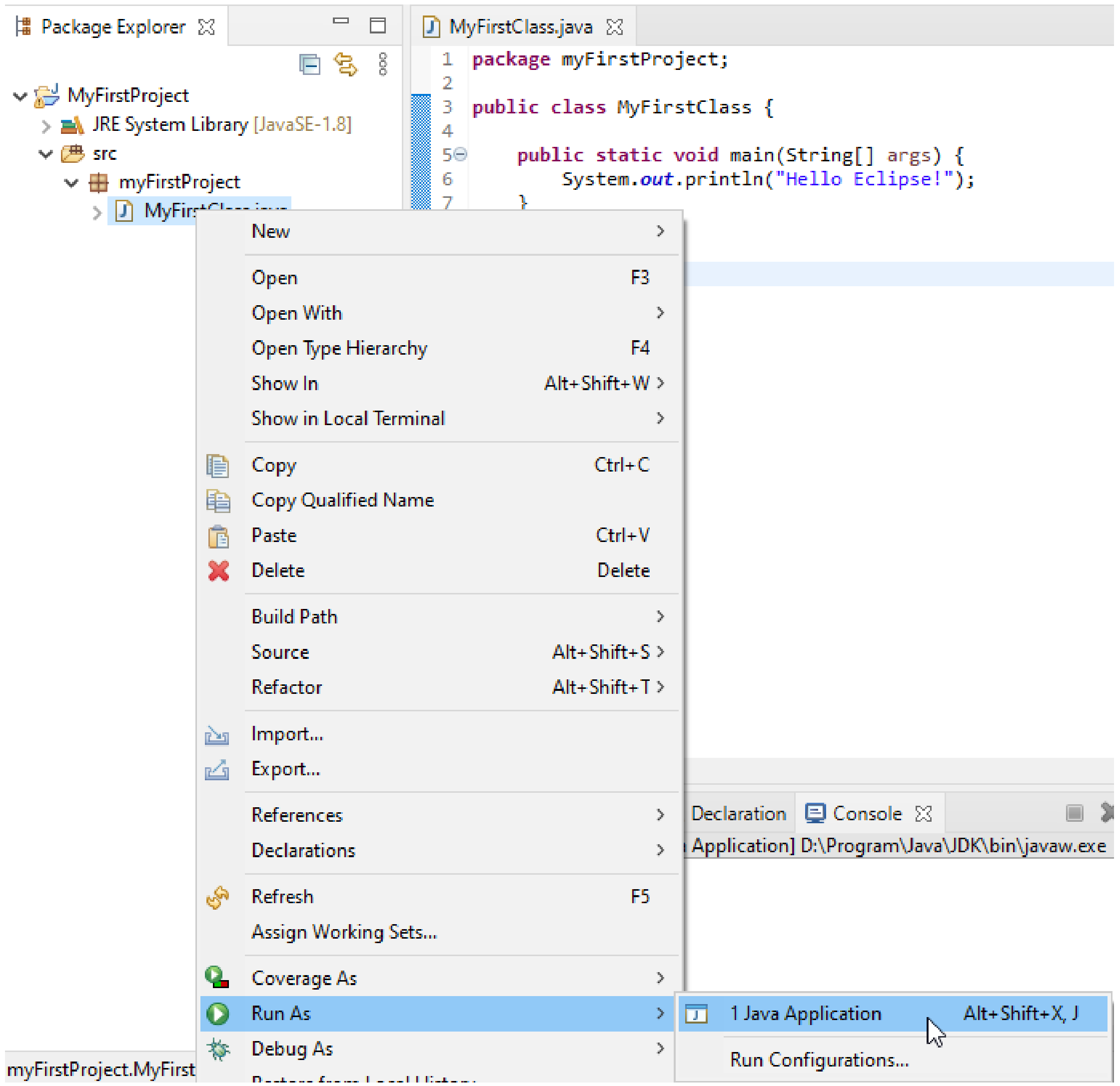
MyFirstClass.java

```
package myFirstProject;

public class MyFirstClass {

    public static void main(String[] args) {
        System.out.println("Hello Eclipse!");
    }

}

```

- You could also directly create new packages via this dialog. If you enter a new package in this dialog, it is created automatically.

1st Java program (cont'd)

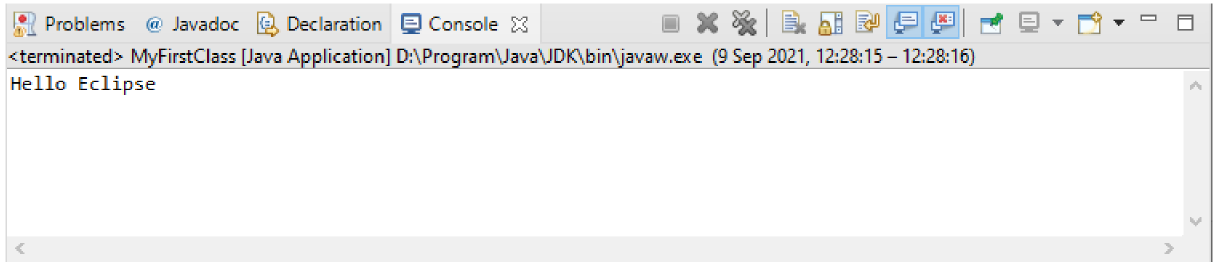
- Run your application code from the IDE
  - Now run your code. Either right-click on your Java class in the *Package Explorer* or right-click in the Java class and select **Run As** > **Java application**.



```
package myFirstProject;

public class MyFirstClass {

    public static void main(String[] args) {
        System.out.println("Hello Eclipse!");
    }
```

1st Java program (cont'd)

- Eclipse will run your Java program. You should see the output in the *Console* view.

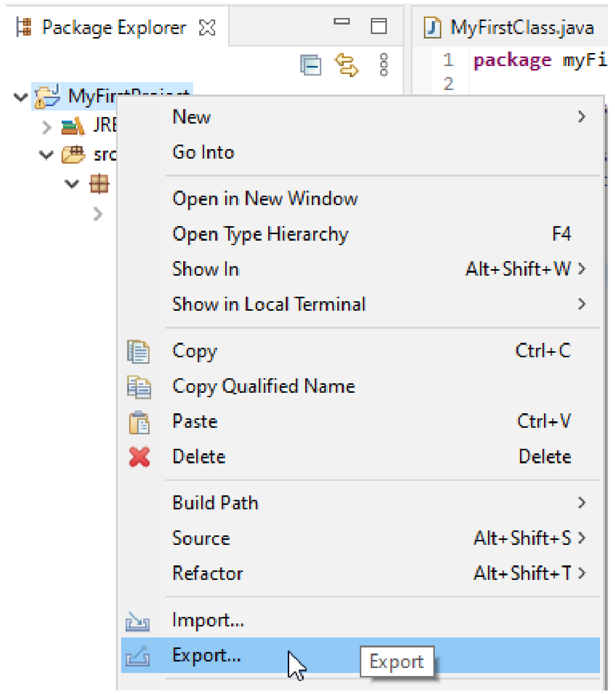


```
<terminated> MyFirstClass [Java Application] D:\Program\Java\JDK\bin\javaw.exe (9 Sep 2021, 12:28:15 – 12:28:16)
Hello Eclipse
```

- Congratulations! You created your first Java project, a package, a Java class and you ran this program inside Eclipse! 👍 😊

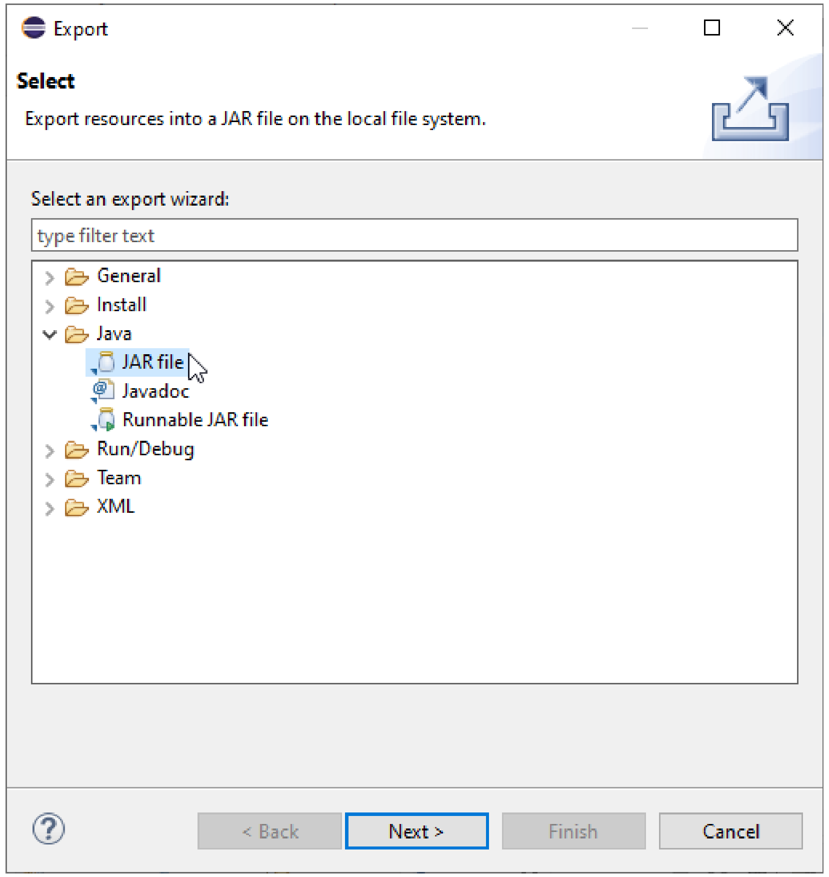
Run Java program outside Eclipse

- Create JAR file
  - To run the Java program outside of the Eclipse IDE, you need to export it as a *JAR* file. A *JAR* file is the standard distribution format for Java applications.
  - Select your project, right-click it and select the *Export* menu entry.



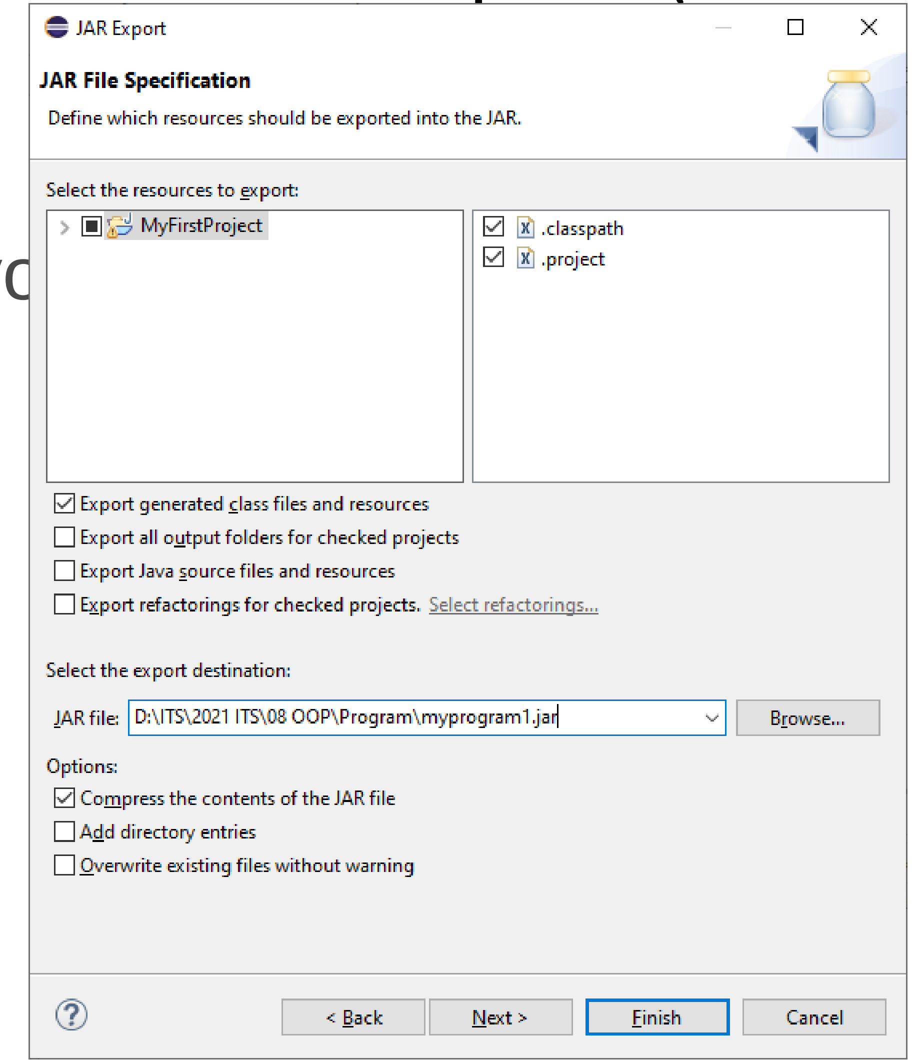
Run Java program outside Eclipse (cont'd)

- Select *JAR file* and select the **Next** button
- Select your project and enter the export destination and a name for the *JAR* file, for example `myprogram1.jar`



Run Java program outside Eclipse (cont'd)

- Press the **Finish** button
- This creates a *JAR* file in yo
  selected output directory.



JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- ■ MyFirstProject
- ✓ .classpath
- ✓ .project

- [✓] Export generated class files and resources
- [ ] Export all output folders for checked projects
- [ ] Export Java source files and resources
- [ ] Export refactorings for checked projects. Select refactorings...

Select the export destination:

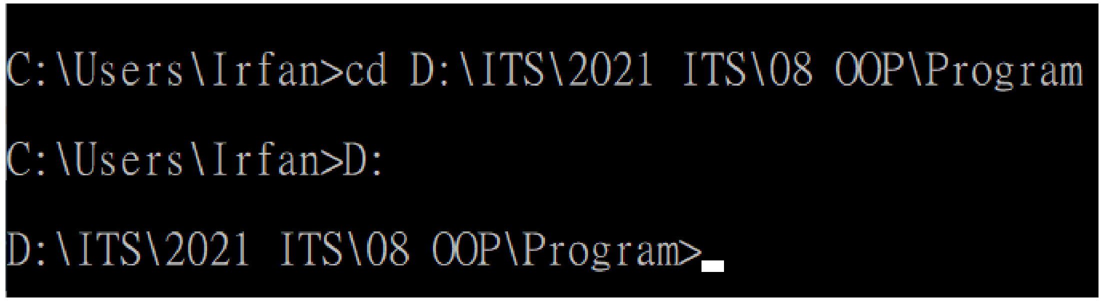
JAR file: D:\ITS\2021 ITS\08 OOP\Program\myprogram1.jar

Options:

- [✓] Compress the contents of the JAR file
- [ ] Add directory entries
- [ ] Overwrite existing files without warning

Run Java program outside Eclipse (cont'd)

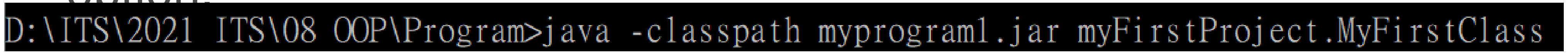
- Open a command shell, e.g., under Microsoft Windows select **Start > Run** and type `cmd` and press the **Enter** key
- This should open a console window.
- Switch to the directory which contains the *JAR* file, by typing `cd path`
- For example, if your *JAR* is located in `D:\ITS\2021 ITS\08 OOP\Program`, use the following command.



```
C:\Users\Irfan>cd D:\ITS\2021 ITS\08 OOP\Program

C:\Users\Irfan>D:

D:\ITS\2021 ITS\08 OOP\Program>_
```

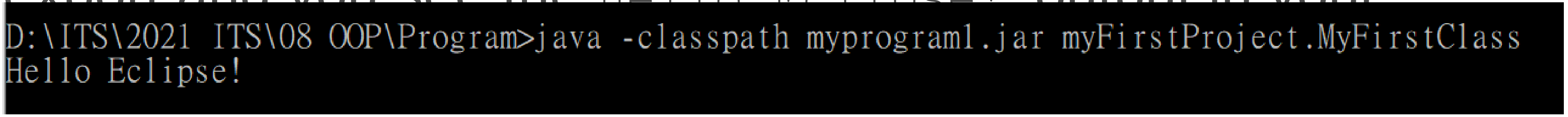
Run Java program outside Eclipse (cont’d)

- To run this program, include the *JAR* file in your `classpath`. The `classpath` defines which Java classes are available to the Java runtime.
  - You can add a *JAR* file to the classpath with the `-classpath` or `-cp` option.



```
D:\ITS\2021 ITS\08 OOP\Program>java -classpath myprogram1.jar myFirstProject.MyFirstClass
```

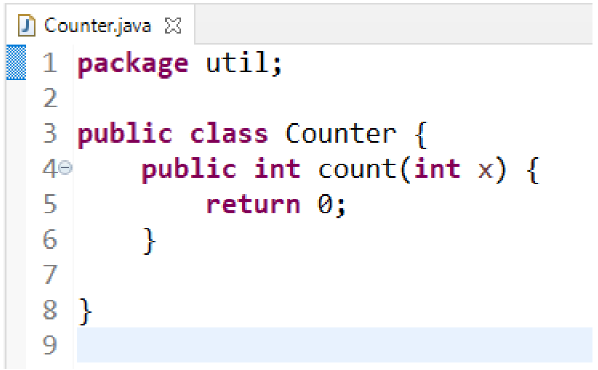
- Type the above command in the directory you used for the export and you see the `Hello Eclipse!` output in your



```
D:\ITS\2021 ITS\08 OOP\Program>java -classpath myprogram1.jar myFirstProject.MyFirstClass
Hello Eclipse!
```

Project, packages & import statements (1)

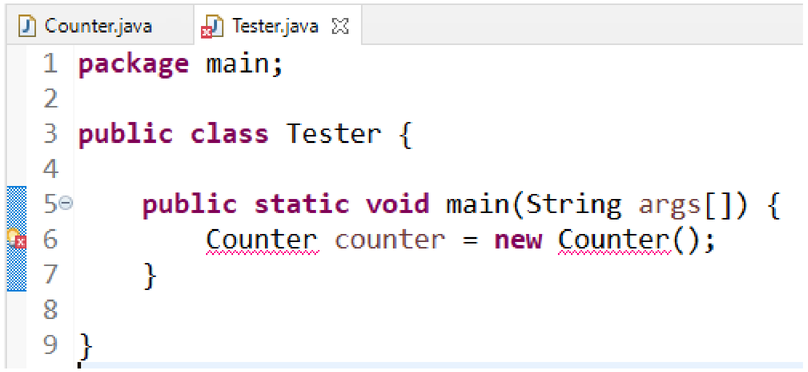
- Create project
  - Create a new project called `counter`
  - Creating the following packages
    - `util`
    - `main`
- Create classes
  - Create the following `Counter` class in the `*.util` package



```java
package util;

public class Counter {
    public int count(int x) {
        return 0;
    }

}
```

Project, packages & import statements (2)

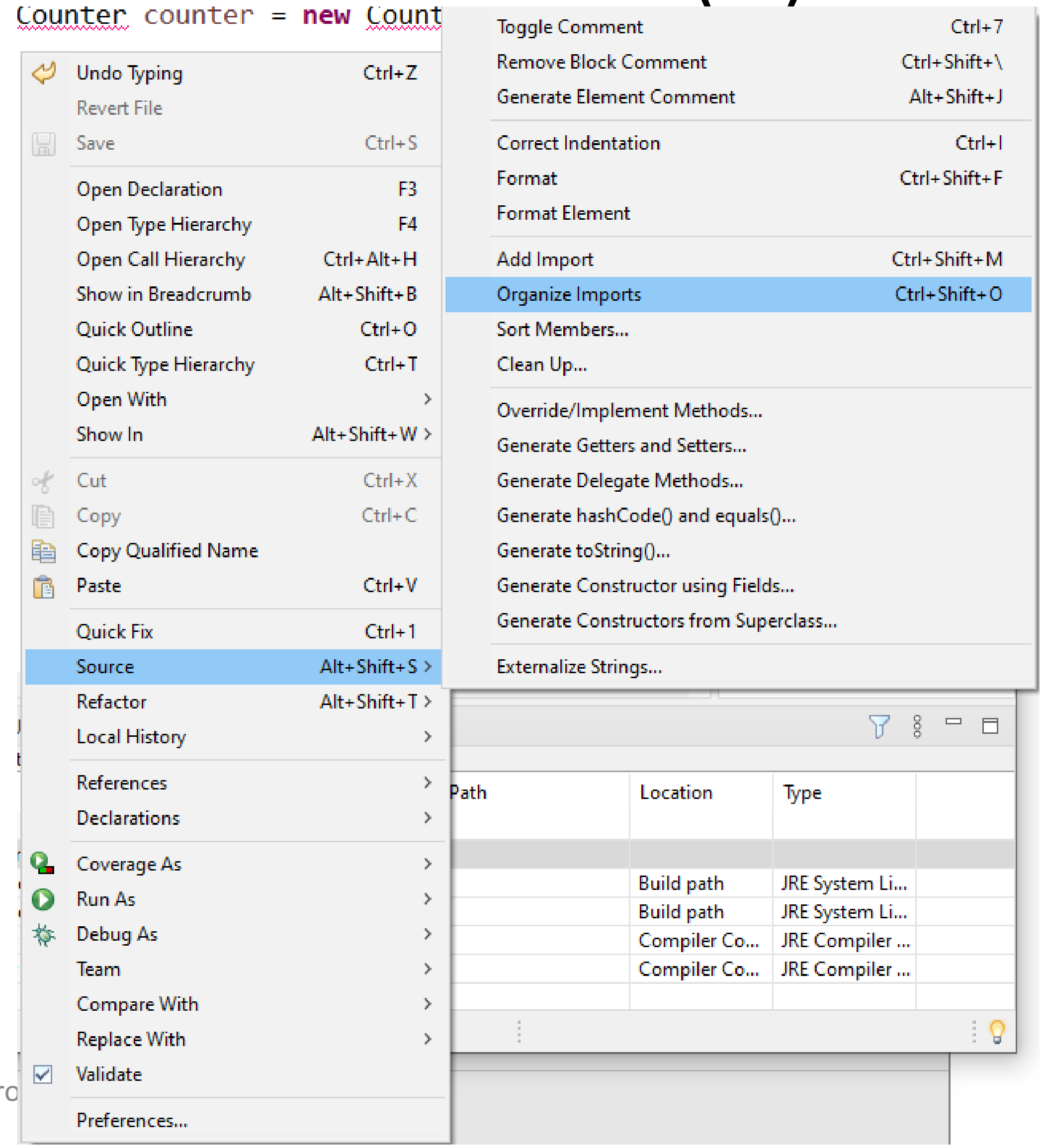
- Create the following `Tester` class in the `*.main` package. This is a simple class without the usage of any unit testing framework like `JUnit`.
- The Eclipse editor should mark the created class with an error because the required `import` statements are missing.



```
package main;

public class Tester {

    public static void main(String args[]) {
        Counter counter = new Counter();
    }

}
```

Project, packages & import statements (3)

- Right-click in your Java editor and select **Source > Organize Imports** to add the required import statements to your Java class.

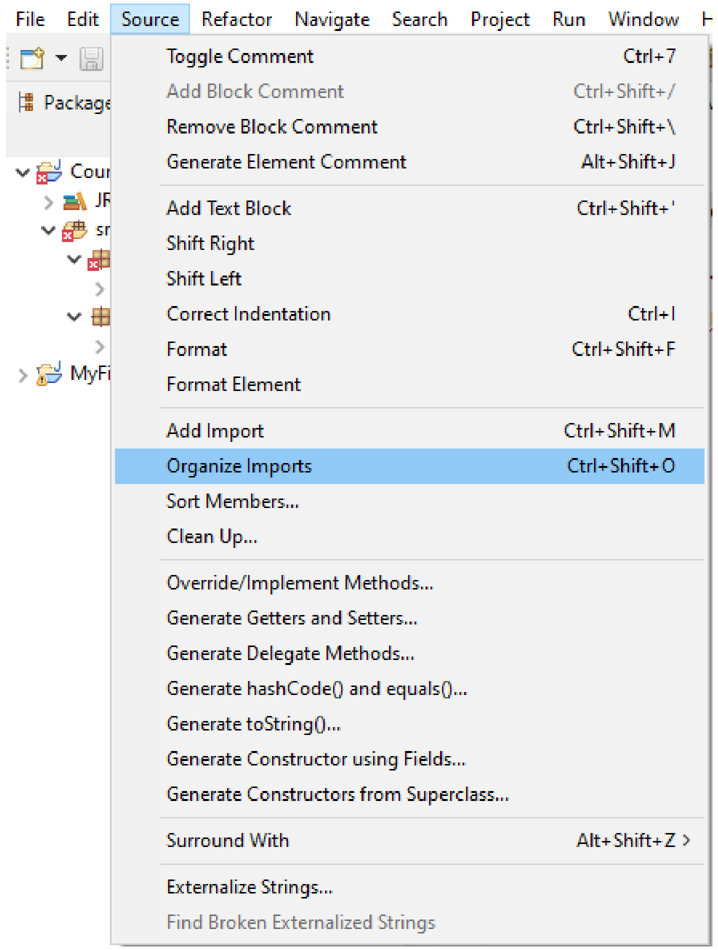


```
Counter counter = new Count
```

Path	Location	Type
	Build path	JRE System Li...
	Build path	JRE System Li...
	Compiler Co...	JRE Compiler ...
	Compiler Co...	JRE Compiler ...

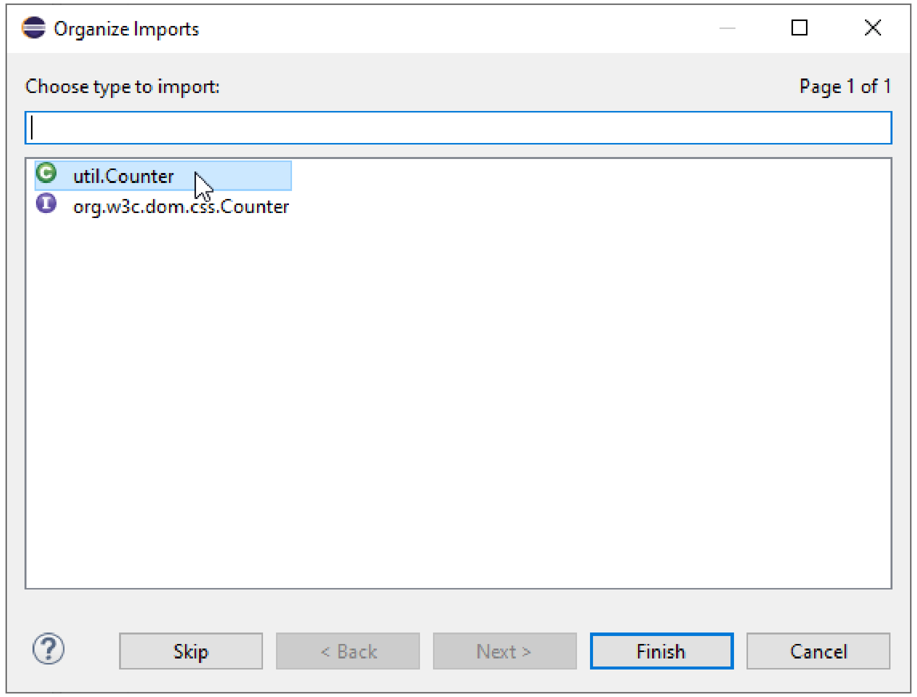
Project, packages & import statements (4)

- Or, simply click menu **Source > Organize Imports** to add the required import statements to your Java class.


File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Package
Toggle Comment Ctrl+7
Add Block Comment Ctrl+Shift+/
Remove Block Comment Ctrl+Shift+\
Generate Element Comment Alt+Shift+J
Add Text Block Ctrl+Shift+'
Shift Right
Shift Left
Correct Indentation Ctrl+I
Format Ctrl+Shift+F
Format Element
Add Import Ctrl+Shift+M
Organize Imports Ctrl+Shift+O
Sort Members...
Clean Up...
Override/Implement Methods...
Generate Getters and Setters...
Generate Delegate Methods...
Generate hashCode() and equals()...
Generate toString()...
Generate Constructor using Fields...
Generate Constructors from Superclass...
Surround With Alt+Shift+Z
Externalize Strings...
Find Broken Externalized Strings

Project, packages & import statements (5)

- And choose Import ‘Counter’ (util)

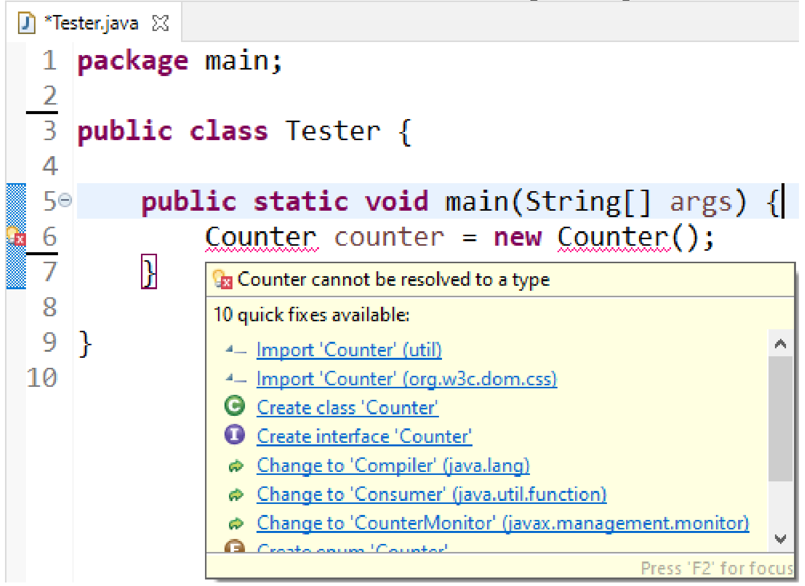


Choose type to import: Page 1 of 1

util.Counter
org.w3c.dom.css.Counter

Project, packages & import statements (6)

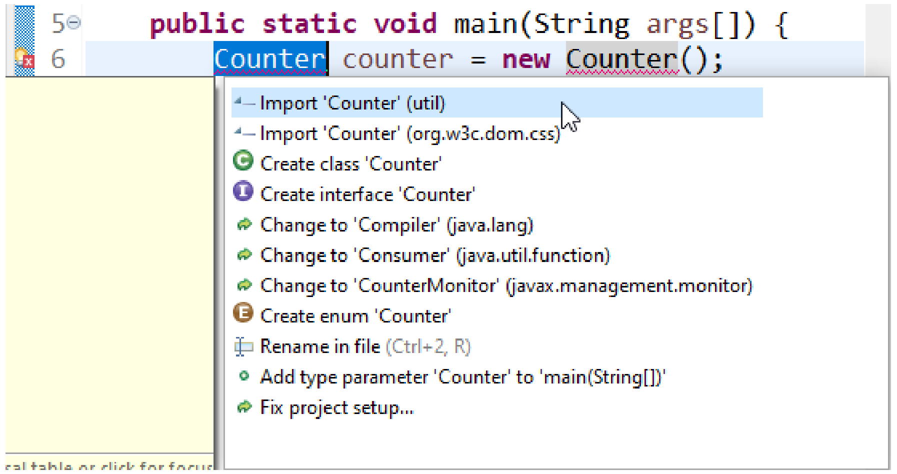
- Alternatively, you may hover your mouse at the code which red colour underline and you will see the error's description. That red colour marker before the line number also can be clicked to show the error's description – as another alternative
- And choose Import 'Counter' (util)



```
*Tester.java
package main;

public class Tester {

    public static void main(String[] args) {
        Counter counter = new Counter();
    }

}
```

Counter cannot be resolved to a type
10 quick fixes available:
- Import 'Counter' (util)
- Import 'Counter' (org.w3c.dom.css)
- Create class 'Counter'
- Create interface 'Counter'
- Change to 'Compiler' (java.lang)
- Change to 'Consumer' (java.util.function)
- Change to 'CounterMonitor' (javax.management.monitor)
- Create enum 'Counter'

Press 'F2' for focus

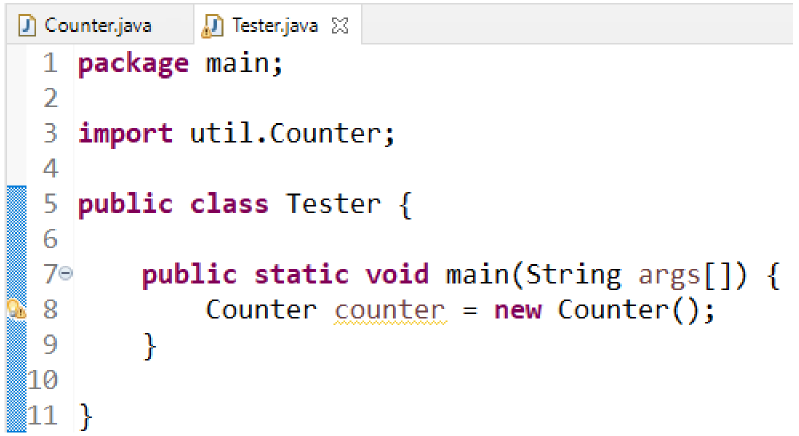


```
    public static void main(String args[]) {
        Counter counter = new Counter();
```

- Import 'Counter' (util)
- Import 'Counter' (org.w3c.dom.css)
- Create class 'Counter'
- Create interface 'Counter'
- Change to 'Compiler' (java.lang)
- Change to 'Consumer' (java.util.function)
- Change to 'CounterMonitor' (javax.management.monitor)
- Create enum 'Counter'
- Rename in file (Ctrl+2, R)
- Add type parameter 'Counter' to 'main(String[])'
- Fix project setup...

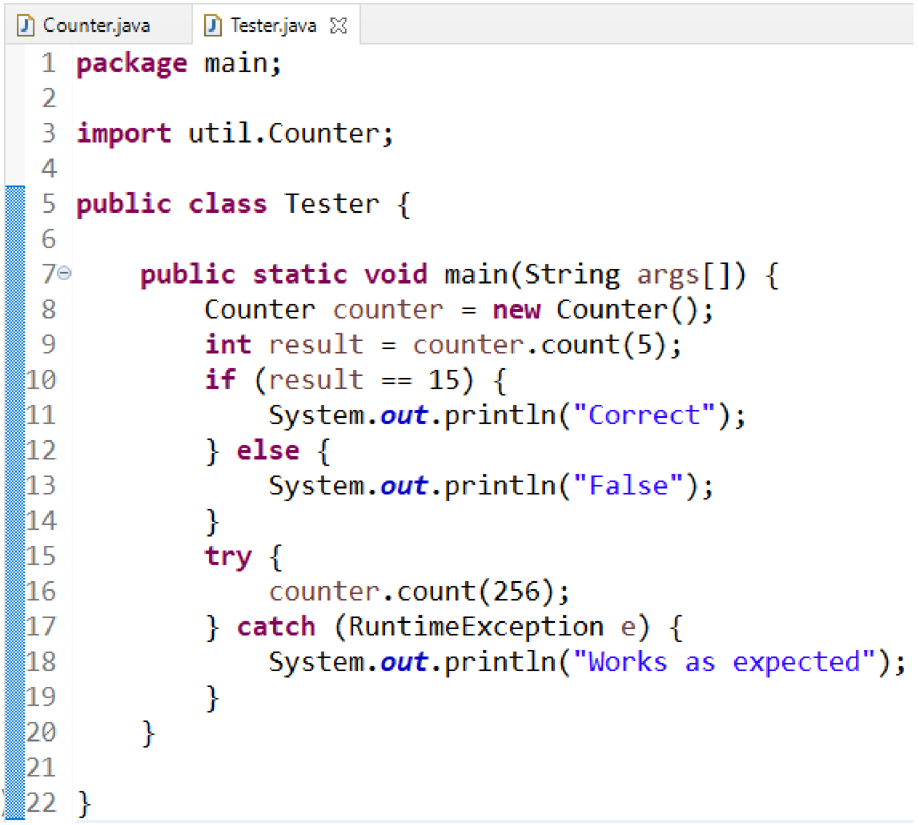
Project, packages & import statements (7)

- This should remove the syntax error. Finish the implementation for the `Tester` class based on the following code.



Counter.java Tester.java

```java
package main;

import util.Counter;

public class Tester {

    public static void main(String args[]) {
        Counter counter = new Counter();
    }

}
```

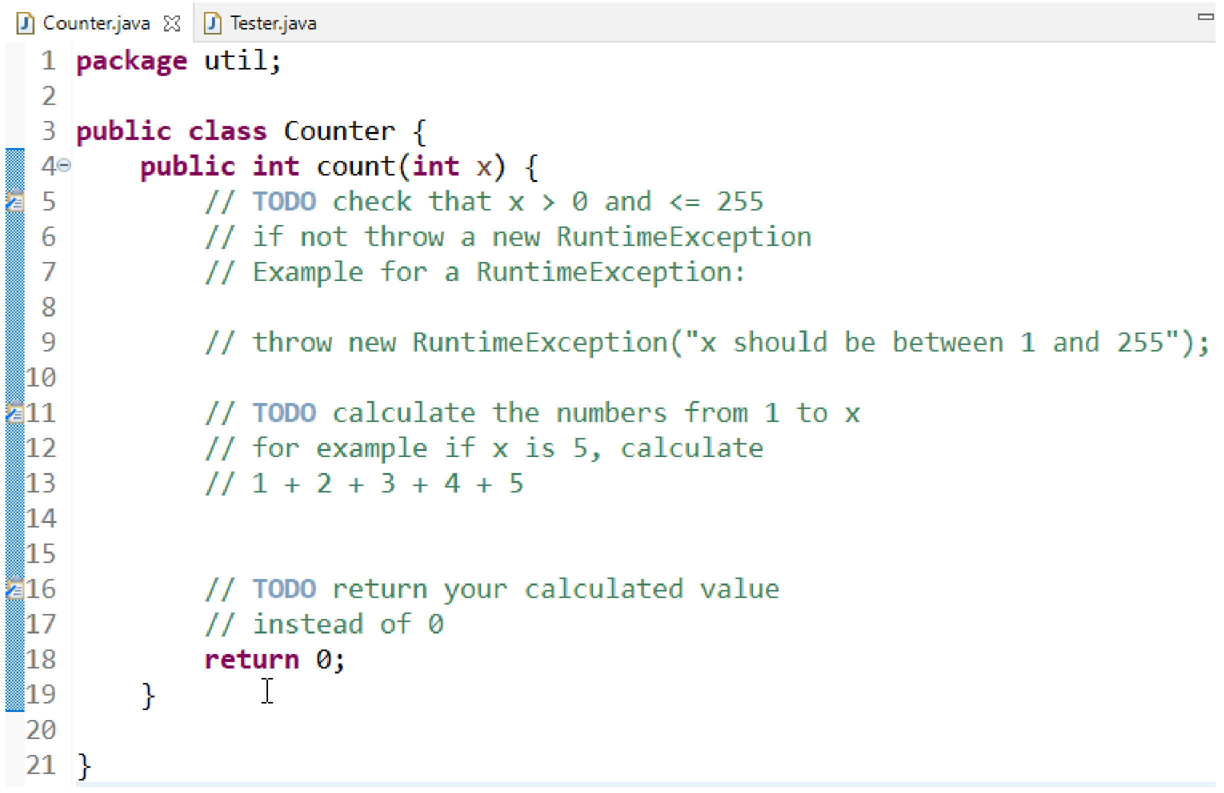


Counter.java Tester.java

```java
package main;

import util.Counter;

public class Tester {

    public static void main(String args[]) {
        Counter counter = new Counter();
        int result = counter.count(5);
        if (result == 15) {
            System.out.println("Correct");
        } else {
            System.out.println("False");
        }
        try {
            counter.count(256);
        } catch (RuntimeException e) {
            System.out.println("Works as expected");
        }
    }

}
```

Project, packages & import statements (8)

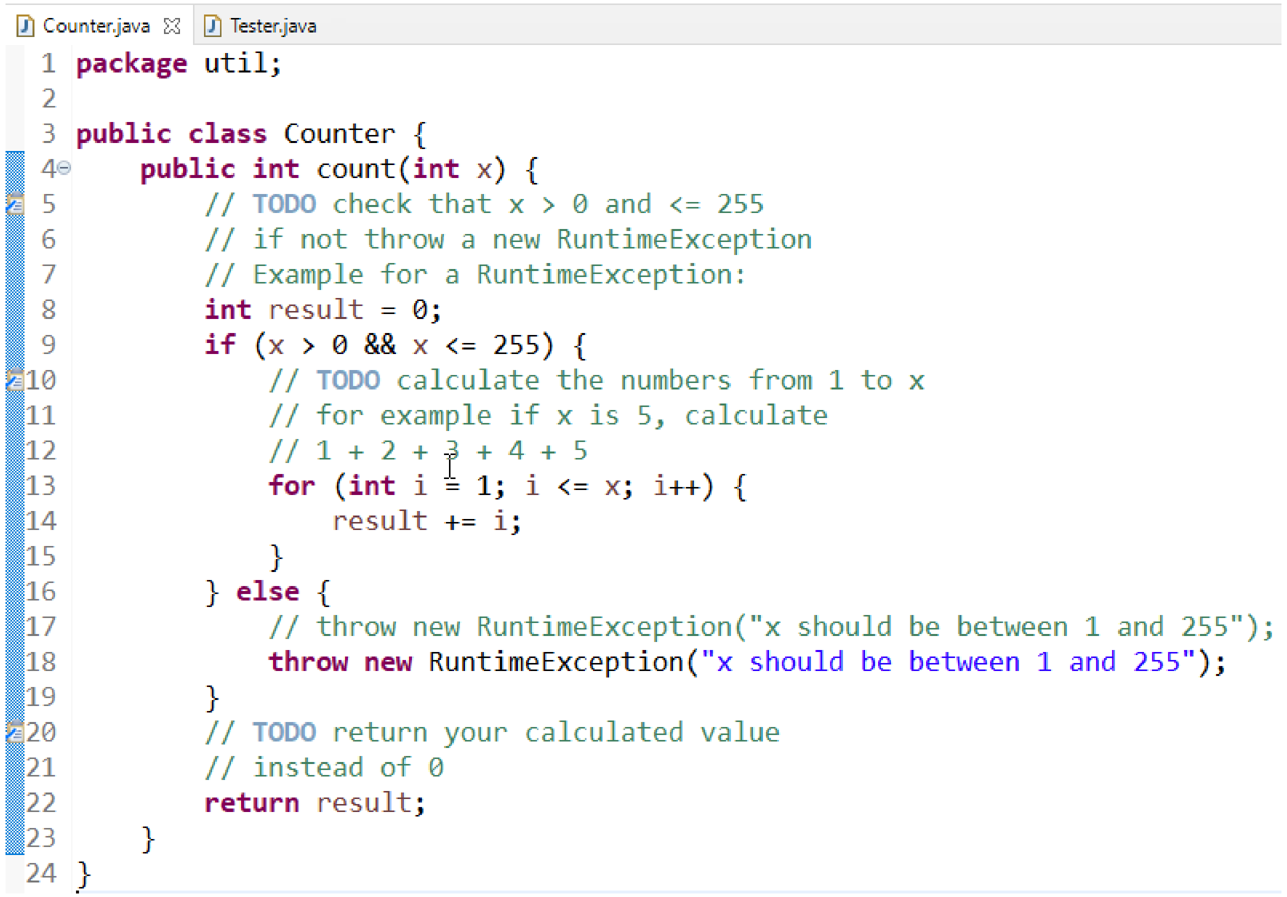
- The `Counter` class had in its source code a comment starting with *TODO*. Finish the source code and calculate the correct values.
- Run the `Tester` class and validate that your implementation is correct. The `Tester` class checks for an example value but the method should work for different input values.



```java
package util;

public class Counter {
    public int count(int x) {
        // TODO check that x > 0 and <= 255
        // if not throw a new RuntimeException
        // Example for a RuntimeException:

        // throw new RuntimeException("x should be between 1 and 255");

        // TODO calculate the numbers from 1 to x
        // for example if x is 5, calculate
        // 1 + 2 + 3 + 4 + 5


        // TODO return your calculated value
        // instead of 0
        return 0;
    }

}
```

Project, packages & import statements (9)

- An example of the solution

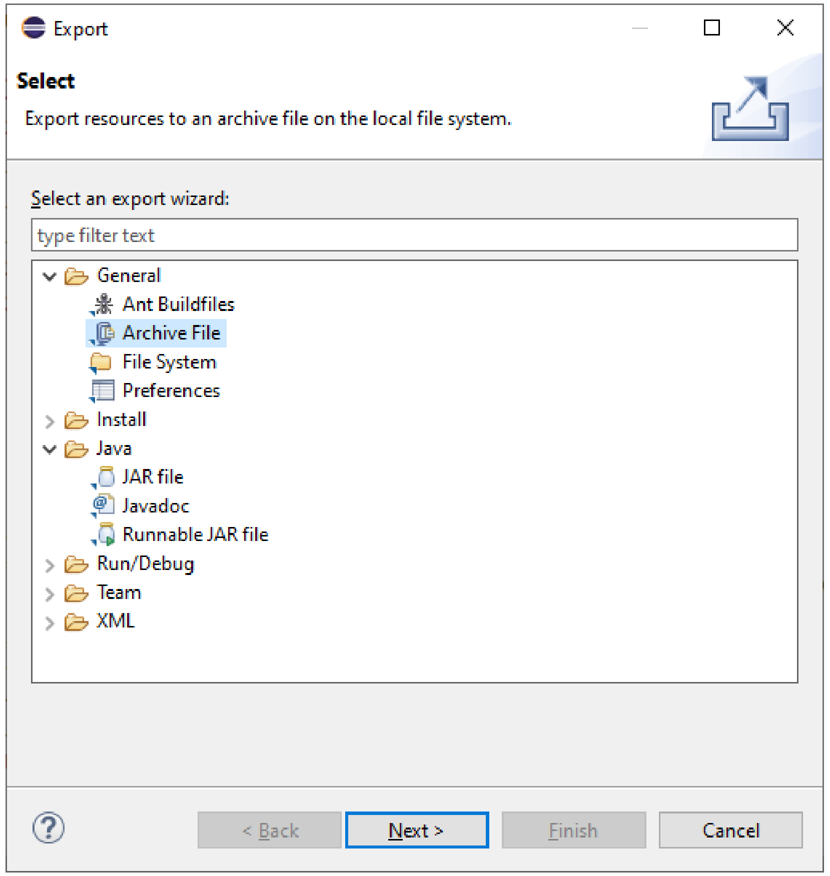


Counter.java | Tester.java

```java
package util;

public class Counter {
    public int count(int x) {
        // TODO check that x > 0 and <= 255
        // if not throw a new RuntimeException
        // Example for a RuntimeException:
        int result = 0;
        if (x > 0 && x <= 255) {
            // TODO calculate the numbers from 1 to x
            // for example if x is 5, calculate
            // 1 + 2 + 3 + 4 + 5
            for (int i = 1; i <= x; i++) {
                result += i;
            }
        } else {
            // throw new RuntimeException("x should be between 1 and 255");
            throw new RuntimeException("x should be between 1 and 255");
        }
        // TODO return your calculated value
        // instead of 0
        return result;
    }
}
```

Exporting & importing projects

- Exporting projects
  - You can export and import Eclipse projects. This allows you to share projects with other people and to import existing projects.
  - To export Eclipse projects, select **File** > **Export** > **General** > **Archive File** and select the projects you want to export.


Export

Select

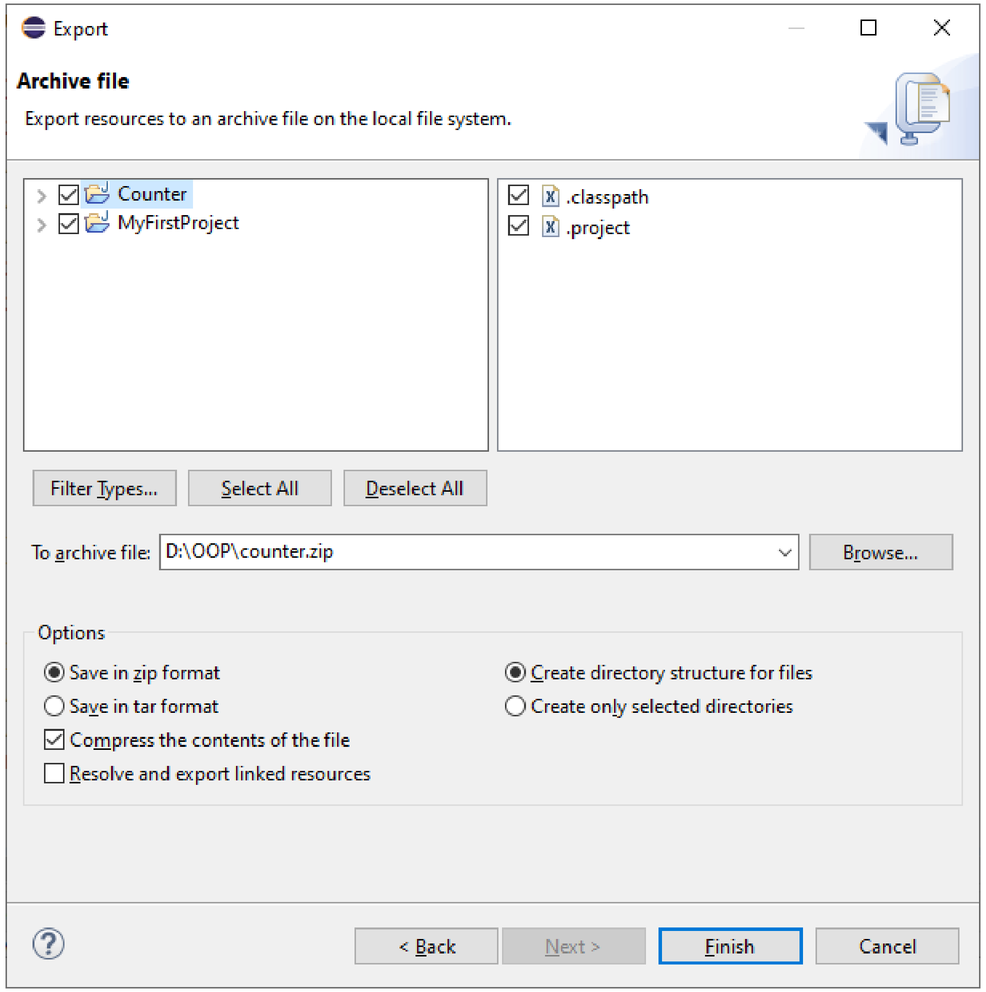
Export resources to an archive file on the local file system.

Select an export wizard:

type filter text

- General
  - Ant Buildfiles
  - Archive File
  - File System
  - Preferences
- Install
- Java
  - JAR file
  - Javadoc
  - Runnable JAR file
- Run/Debug
- Team
- XML

Exporting & importing projects (cont'd)


Export

Archive file

Export resources to an archive file on the local file system.

- [✓] Counter
- [✓] MyFirstProject

- [✓] .classpath
- [✓] .project

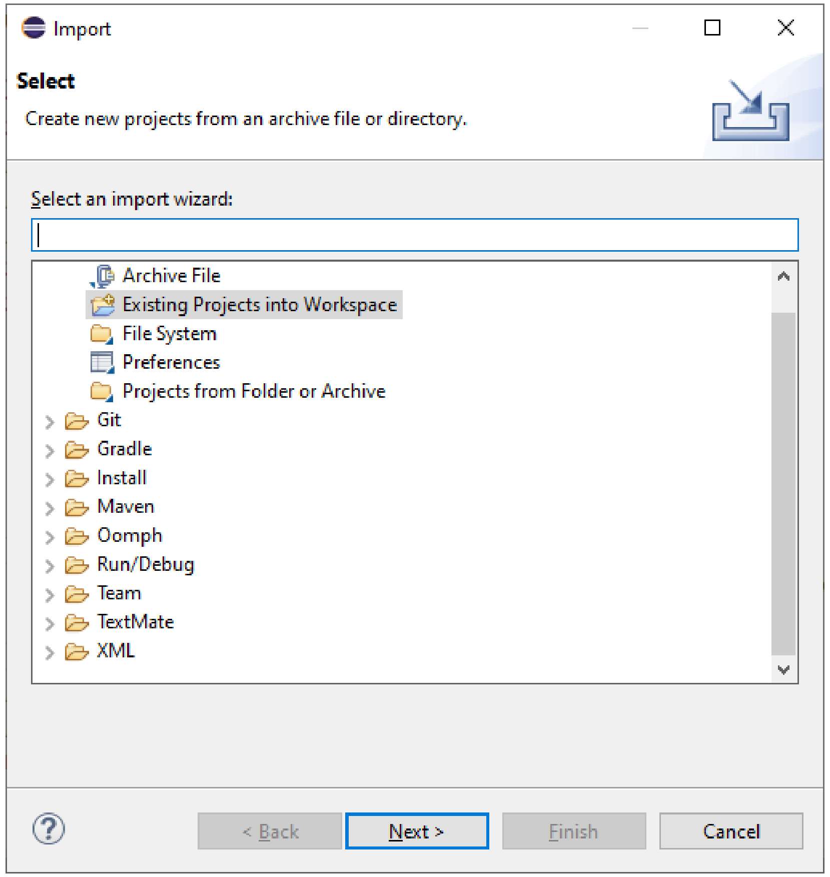
To archive file: D:\OOP\counter.zip

Options

- (●) Save in zip format
- ( ) Save in tar format
- [✓] Compress the contents of the file
- [ ] Resolve and export linked resources
- (●) Create directory structure for files
- ( ) Create only selected directories

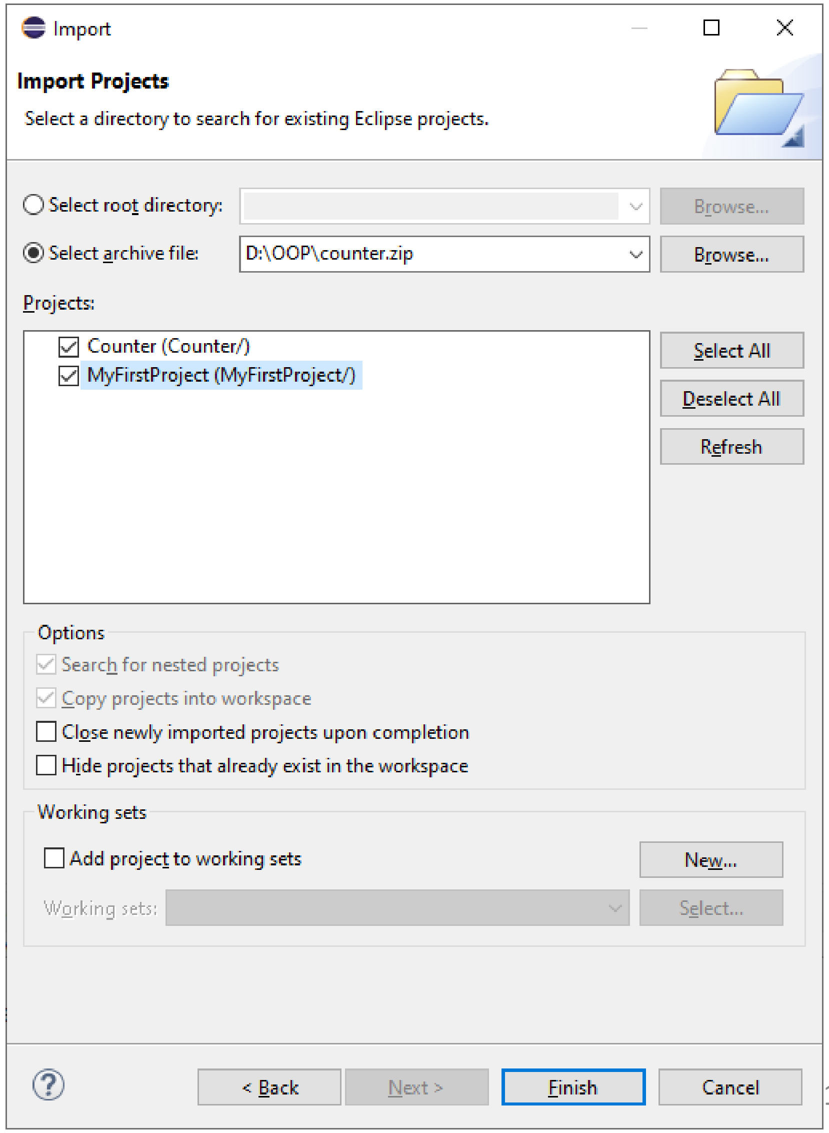
Exporting & importing projects (cont’d)

- Importing projects
  - To import projects, select **File > Import > Existing Projects into Workspace**. You can import from an archive file, i.e., zip file or directly import the project in case you have extracted the zip file.



Exporting & importing projects (cont’d)

- Exercise 1
  - Importing project(s) can be done if the project(s) hasn’t existed in our Workspace
  - Delete the existing one(s) before importing the project(s). Make sure the project(s) has already been exported (zipped) before deleting it.
- Exercise 2
  - Export one of your projects into a zip file. Switch into a new workspace and import the project into your new workspace based on the zip file you exported.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

( ) Select root directory: Browse...

(●) Select archive file: D:\OOP\counter.zip Browse...

Projects:

- ☑ Counter (Counter/)
- ☑ MyFirstProject (MyFirstProject/)

Select All | Deselect All | Refresh

Options

- ☑ Search for nested projects
- ☑ Copy projects into workspace
- ☐ Close newly imported projects upon completion
- ☐ Hide projects that already exist in the workspace

Working sets

- ☐ Add project to working sets

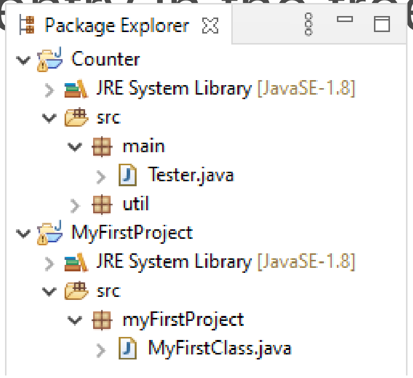
New...

Working sets: Select...

< Back | Next > | Finish | Cancel

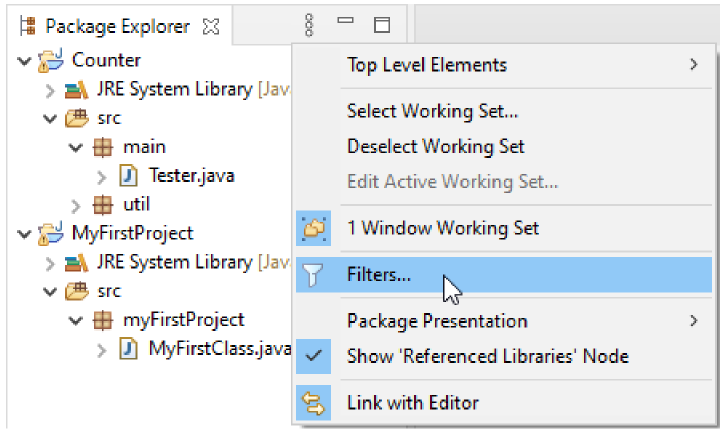
Source navigation

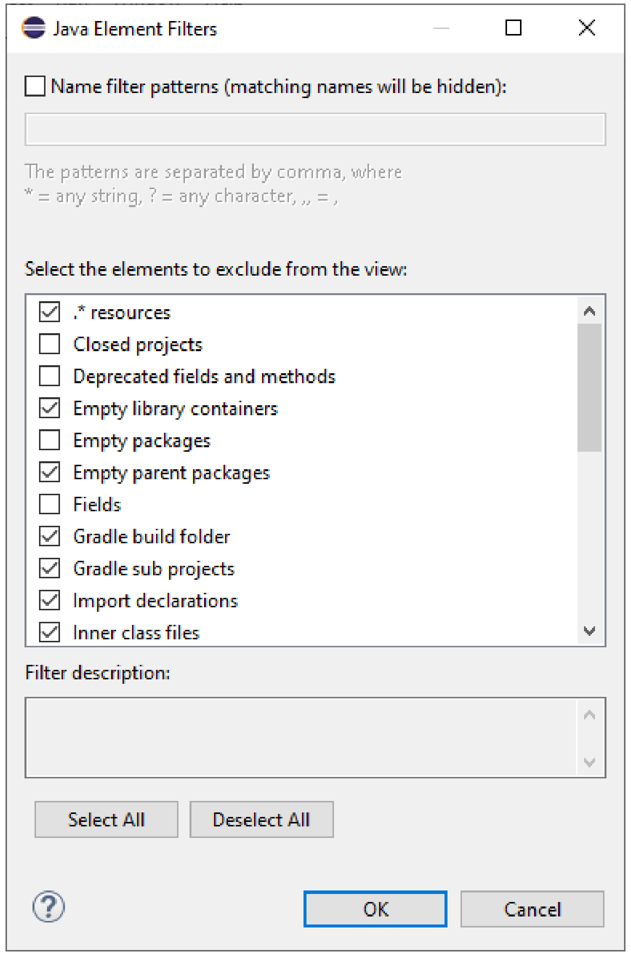
- Package Explorer or Project Explorer
  - The primary way of navigating through your project is the *Package Explorer* or alternatively the *Project Explorer* view. You can open nodes in the tree and open a file in an editor by double-clicking on the corresponding entry in the tree hierarchy.



Source navigation (continued)

- The drop-down menu in the *Package Explorer* allows you to filter the resources which should be displayed or hidden.




Java Element Filters

☐ Name filter patterns (matching names will be hidden):

The patterns are separated by comma, where
* = any string, ? = any character, ,, = ,

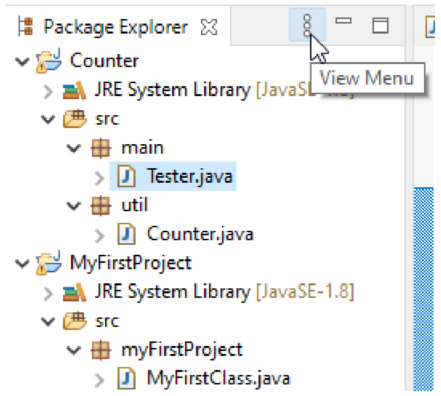
Select the elements to exclude from the view:

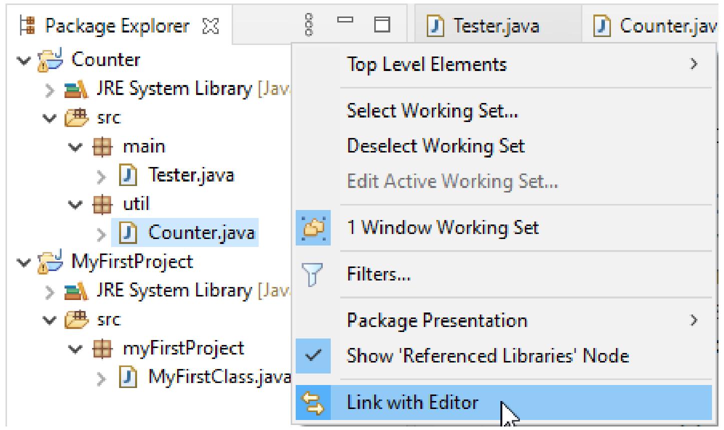
- ☑ .* resources
- ☐ Closed projects
- ☐ Deprecated fields and methods
- ☑ Empty library containers
- ☐ Empty packages
- ☑ Empty parent packages
- ☐ Fields
- ☑ Gradle build folder
- ☑ Gradle sub projects
- ☑ Import declarations
- ☑ Inner class files

Filter description:

Link Package Explorer with editor

- The *Package Explorer* view allows you to display the associated file from the currently selected editor. For example, if you are working on the `Tester.java` file in the Java editor and switch to the Java editor of the `Counter.java` file, then the corresponding file will be selected in the *Package Explorer* view.
- To activate this behaviour, press the **Link with Editor** button in the *Package Explorer* view as depicted in the following screenshot.


Package Explorer
View Menu
Counter
JRE System Library [JavaSE-1.8]
src
main
Tester.java
util
Counter.java
MyFirstProject
JRE System Library [JavaSE-1.8]
src
myFirstProject
MyFirstClass.java


Package Explorer
Tester.java
Counter.jav
Counter
JRE System Library [Jav
src
main
Tester.java
util
Counter.java
MyFirstProject
JRE System Library [Jav
src
myFirstProject
MyFirstClass.java
Top Level Elements
Select Working Set...
Deselect Working Set
Edit Active Working Set...
1 Window Working Set
Filters...
Package Presentation
Show 'Referenced Libraries' Node
Link with Editor