OODP- OOA/OOD– Session 3-b

Session times

PT group 1 – Monday	18:00-21:00	room: Malet 403
PT group 2 – Thursday	18:00-21:00	room: Malet 407
FT - Tuesday	13:30-17:00	room: Malet 404

Email: oded@dcs.bbk.ac.uk

Web Page: http://www.dcs.bbk.ac.uk/~oded

Visiting Hours: Tuesday 17:00 to 19:00

Integrated Development Environment

One user interface for (possibly) all local environment tools.

Goal

- Easier access to tools
- Feedback while editing (syntax style)
- Simultaneous use of more than one tool
- Increase code writing productivity

(some people may not use an IDE or only use them in a restricted way, because of speed issues)

Eclipse

There are a number of IDEs available.
In this course we use **Eclipse**.

Why?

Eclipse has more than a thousand different plug-ins, among them many of the tools we want to demonstrate.

Eclipse –Helps (http://help.eclipse.org/indigo/index.jsp)

The plug-ins and help of Eclipse can be used as a starting point for learning Java.

- We will use Eclipse to guide us towards our first Java program.
- We will use eclipse to advise us
- We will use eclipse to write for us as much of the code as possible
- We will use eclipse to do tedious editing for us
- We will use eclipse to help us avoid typos, syntax erros etc.

Learning to use Eclipse

In this module the approach to learning to use Eclipse and almost any other tool is based on the TDD philosophy

Testing is learning

Or in this case

Using is learning

It is advised to extend this approach beyond sessions and homework.

If you follow this advise, then please tell me if there is anything new that you learned and believe should be added to this module.

Project, Packages etc.

The first thing that we will notice when dealing with Eclipse (except for all the buttons) is that

we are actually working on a project

(even if it is just an “Hello world”)

Why?

To force the programmer into a standard structure, instead of a mess.

A project has a clear directory structure

Packages

Packages are used in order to organize java classes into name spaces.

Java packages can be stored into JAR (Java archive files)

- Grouping classes with related functionality
- Easy way to load a large number of java classes (so that java applets don't take forever to load)

Code Refactoring (TDD, Agile)

Restructuring existing code without changing its external behaviour (to improve readability and maintainability)

Eclipse has many options for automating this process.

One of them is renaming.

In the past the simple operation of renaming has caused so much trouble!

We shall see how Eclipse helps us to prevent such trouble.

Source Code Generation

One of the annoying things about Java is that it is so **verbose**.

Eclipse doesn't make that disappear

However it makes it makes it possible to code a new class with minimal typing.

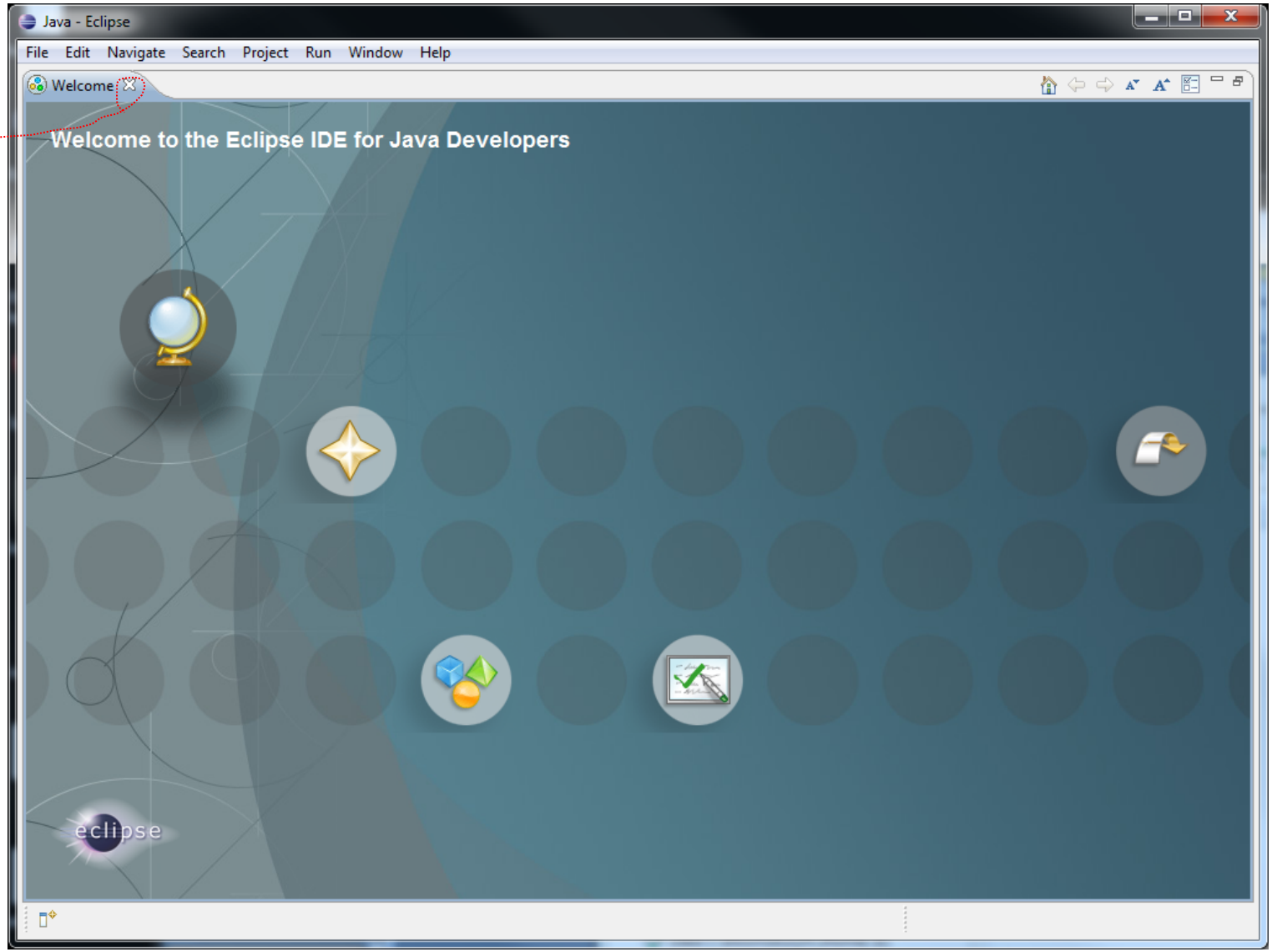
It actually does a lot more as we shall see.

From C++ to Java

No more pointers!

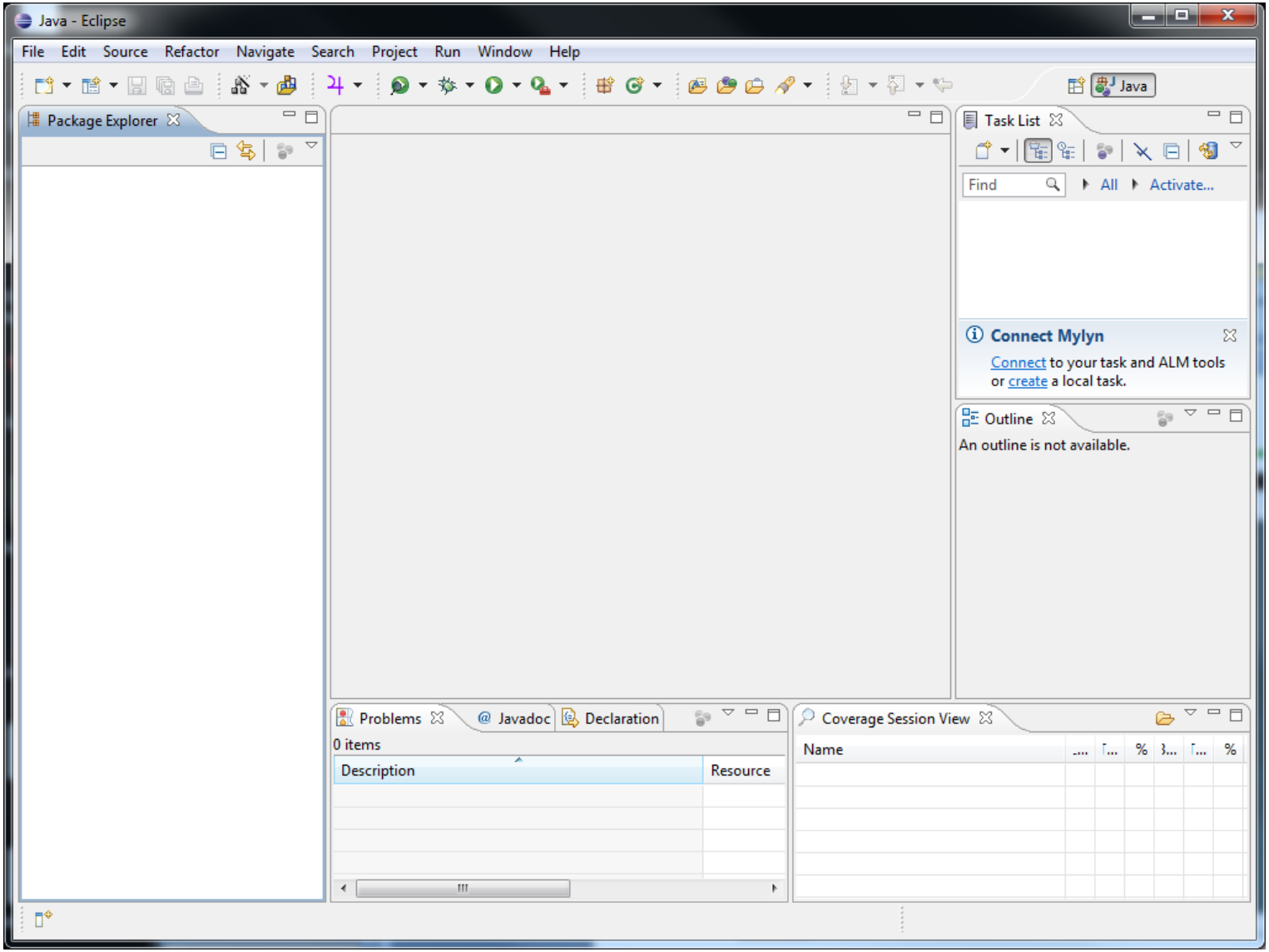
Welcome garbage collection

Welcome to Eclipse

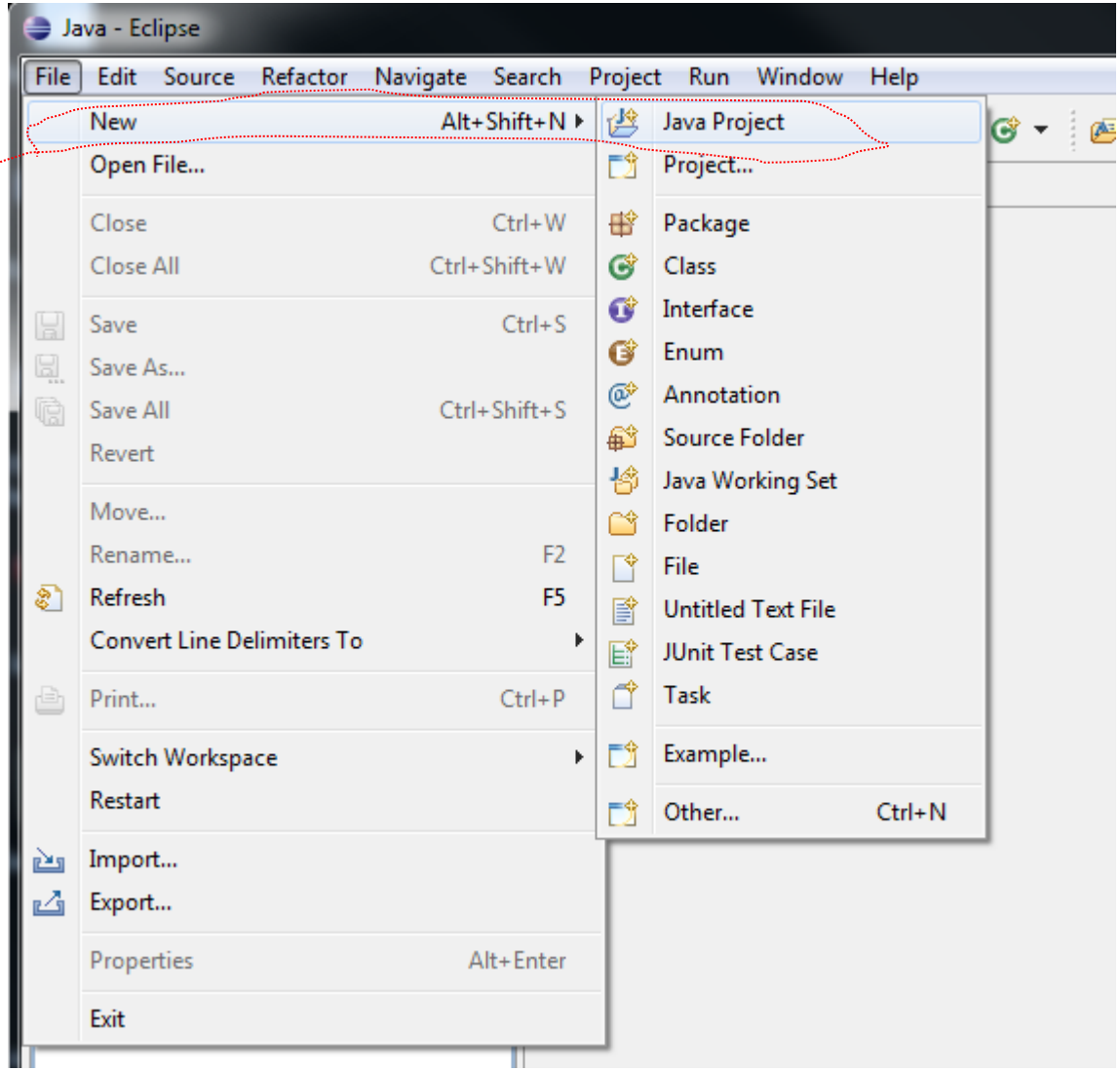


Select

Empty Eclipse

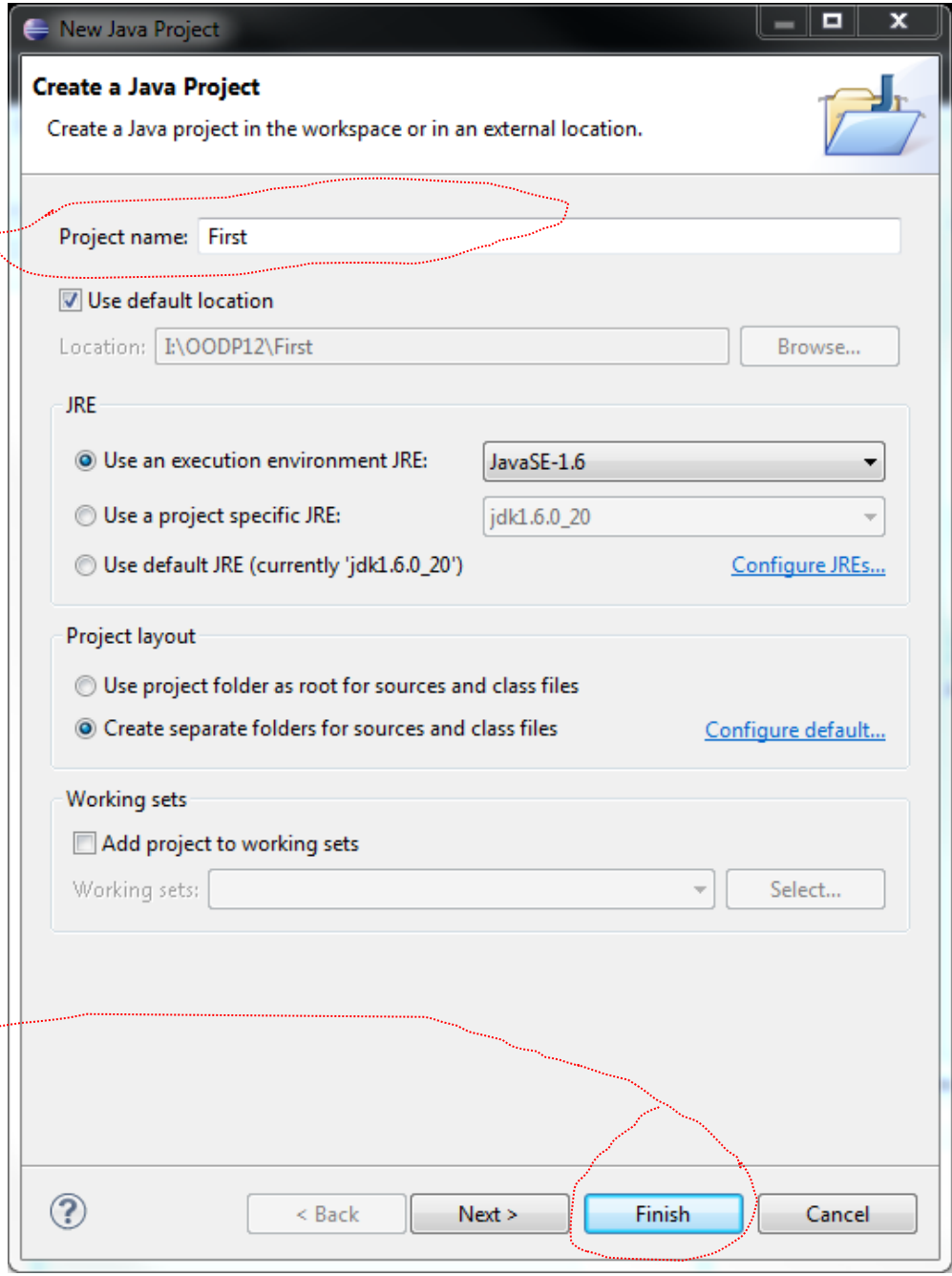


New Project



Select

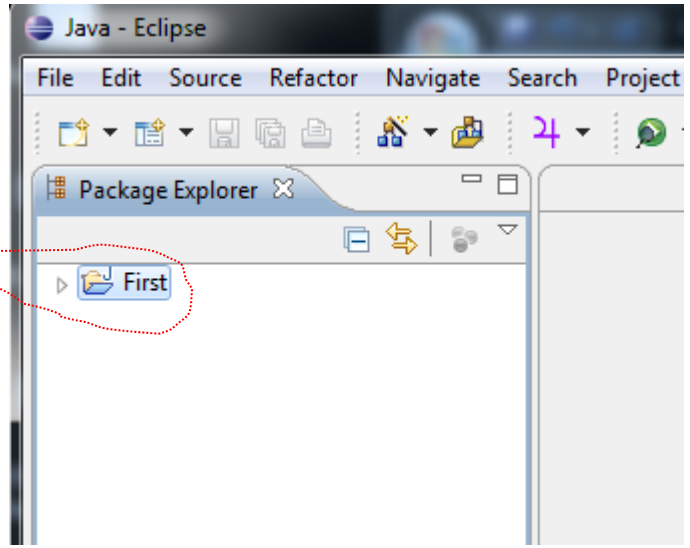
Empty Eclipse



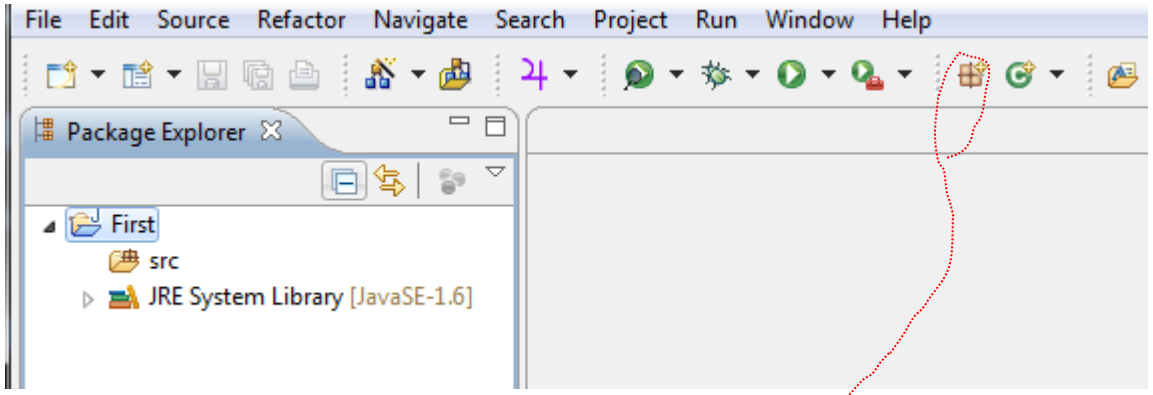
1. Fill

2. Select

New Project

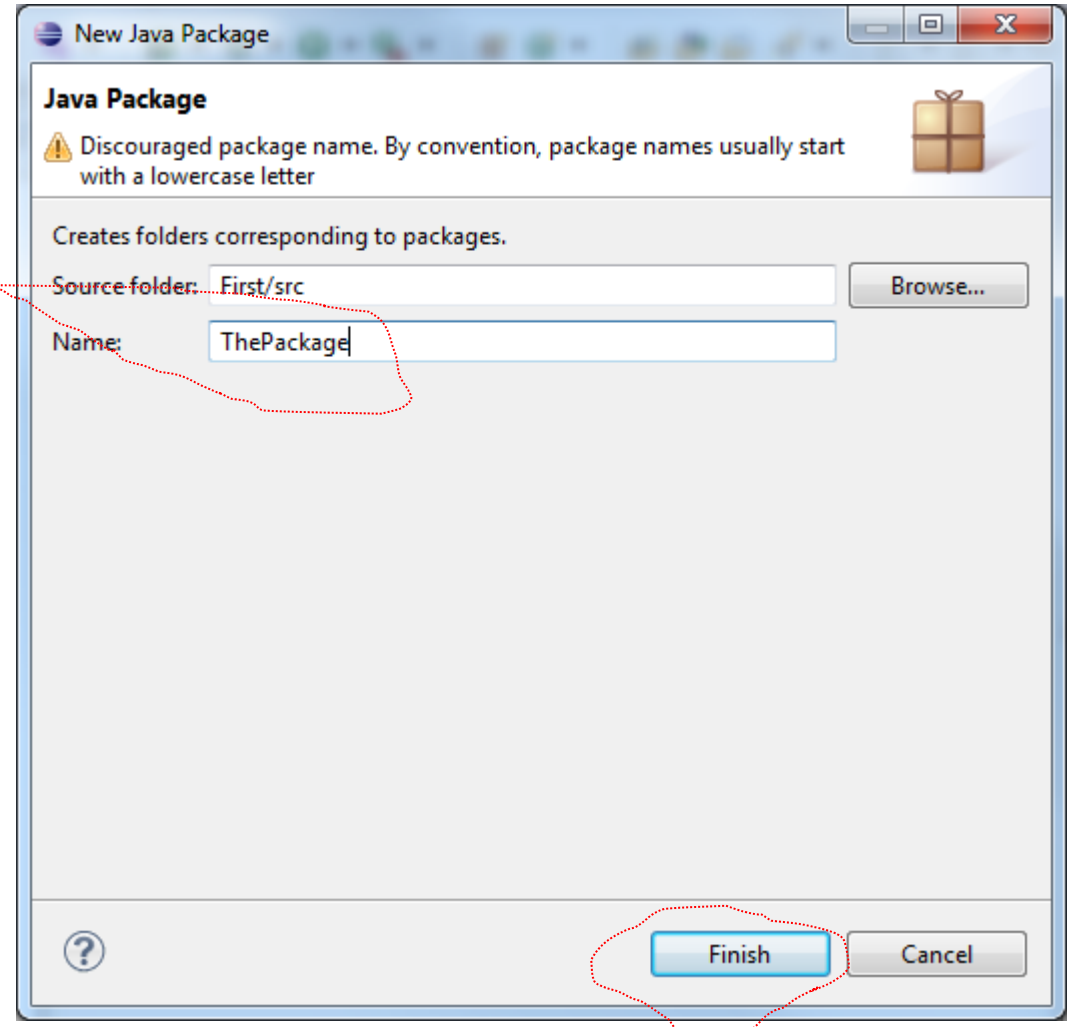


1. Select



2. Select

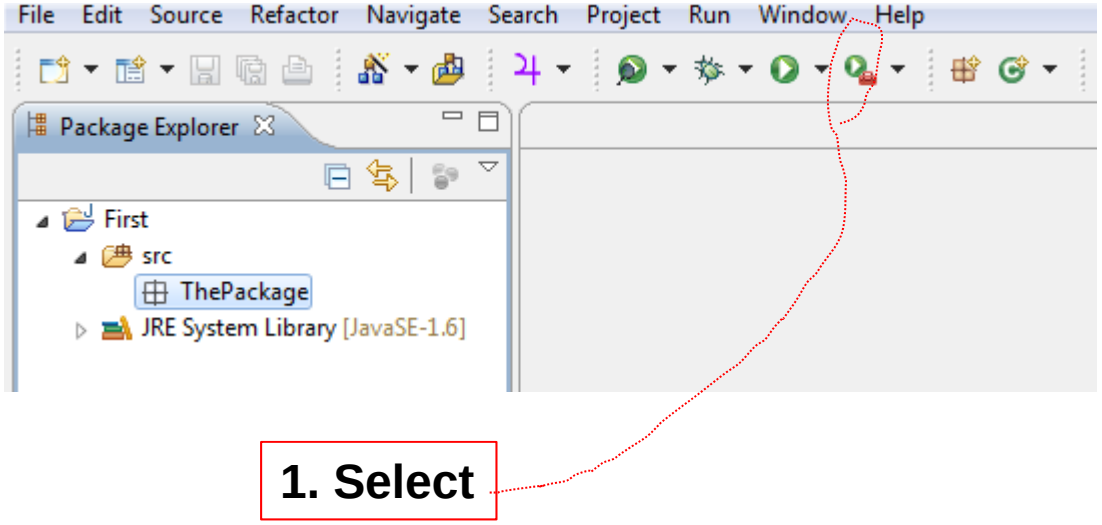
New Package Window

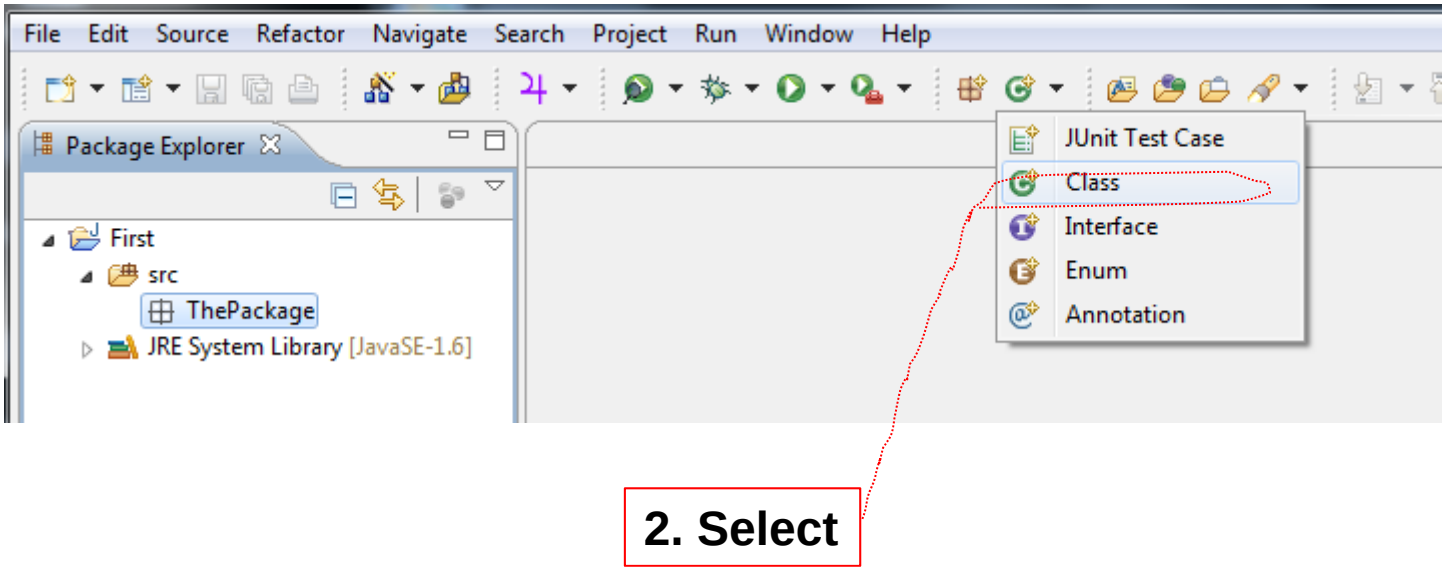


1. Fill in

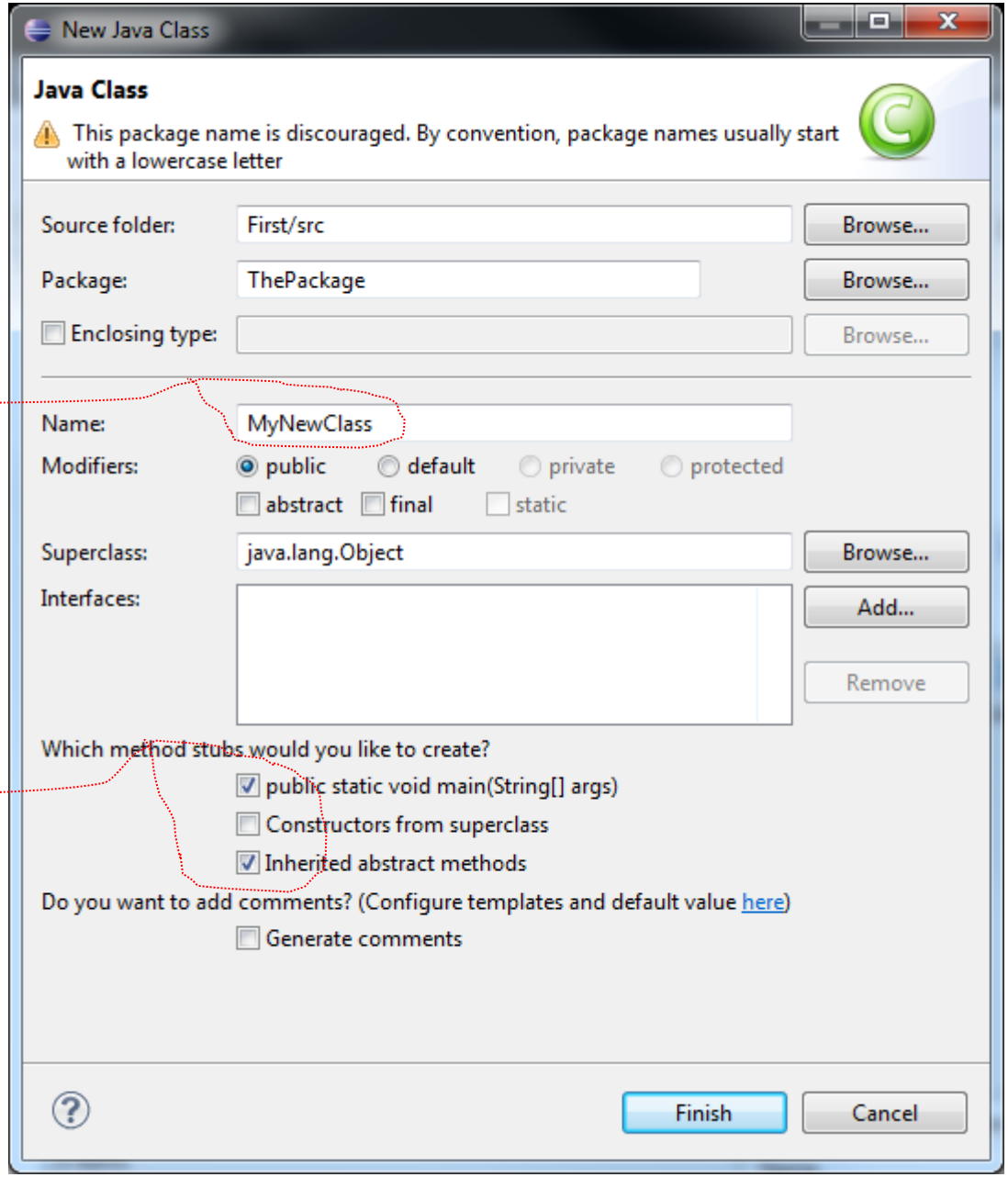
2. Select

New Class


File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
First
src
ThePackage
JRE System Library [JavaSE-1.6]
1. Select


File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
First
src
ThePackage
JRE System Library [JavaSE-1.6]
JUnit Test Case
Class
Interface
Enum
Annotation
2. Select

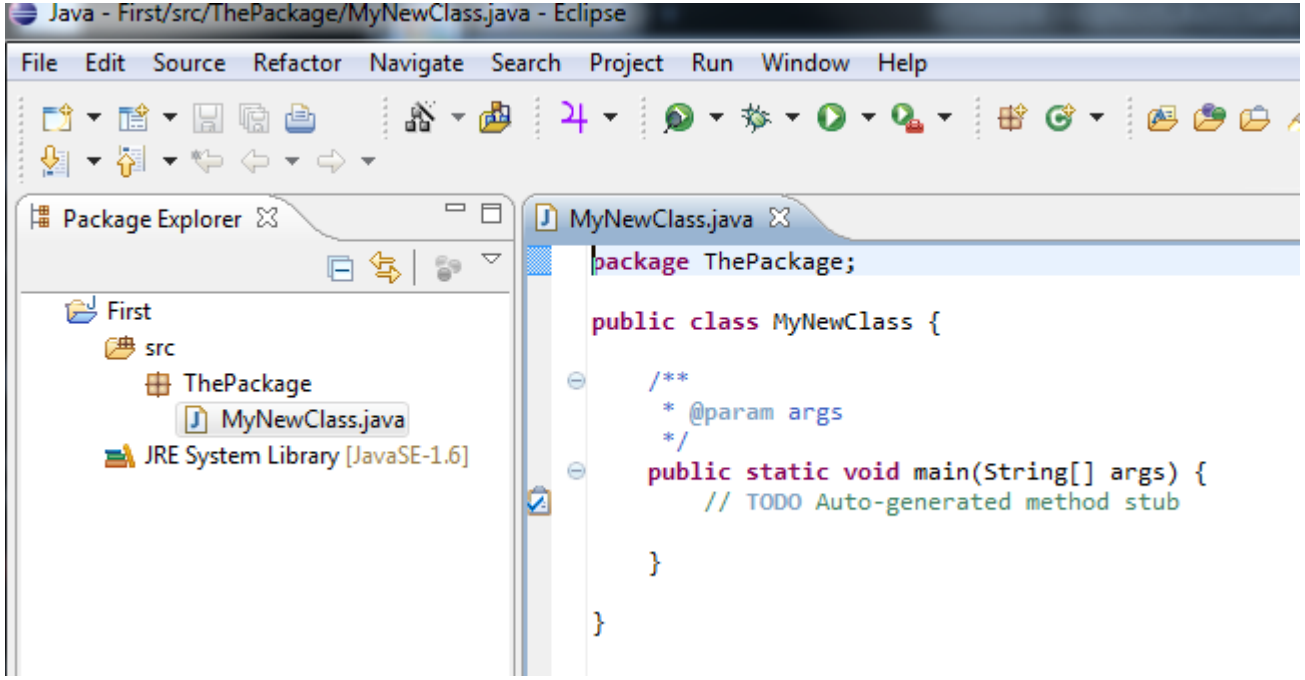
New Class



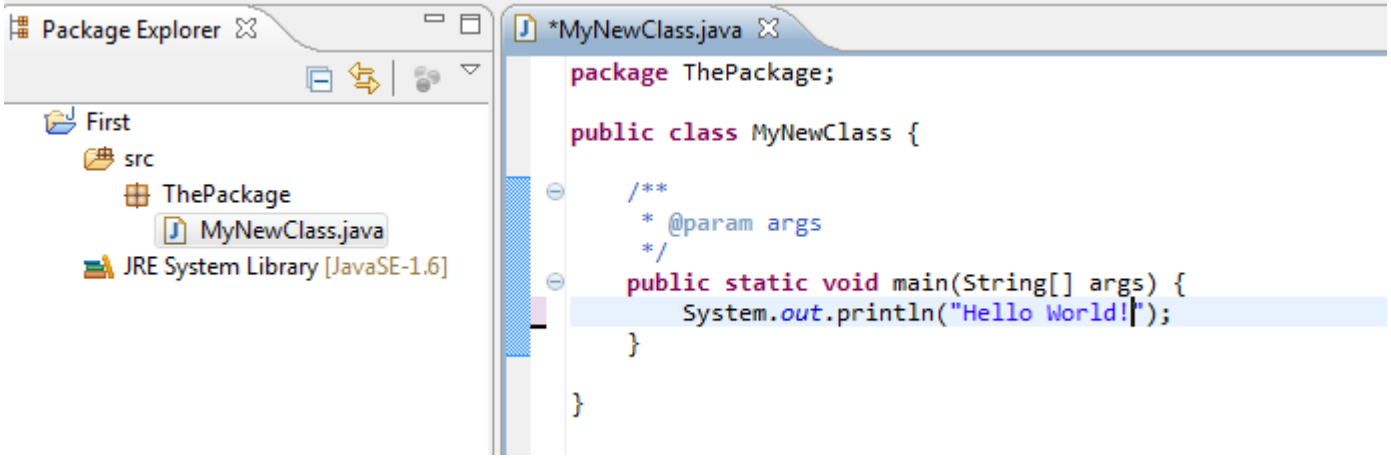
1. Fill in

2. Tick

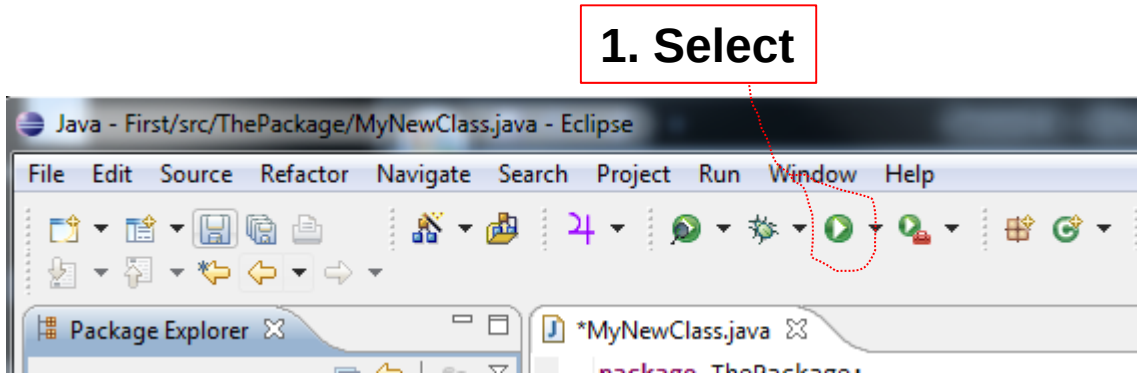
New-Code

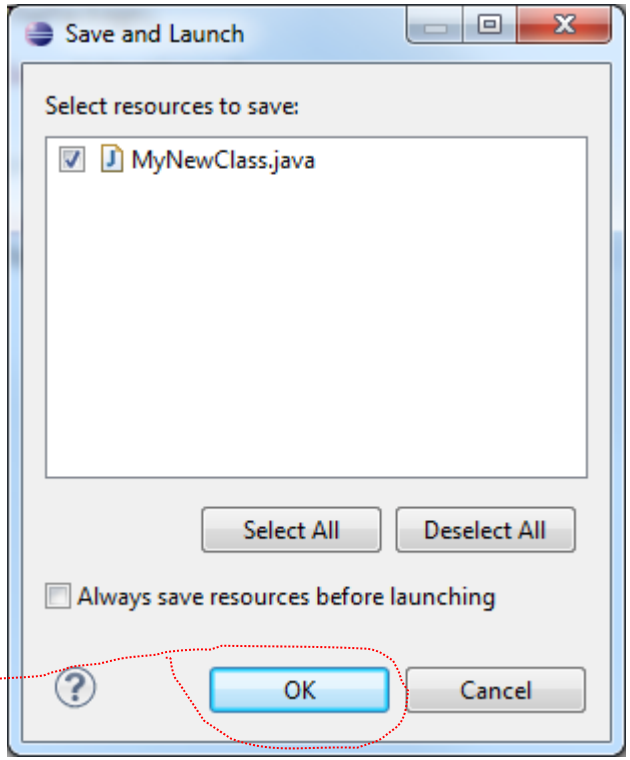


After Editing



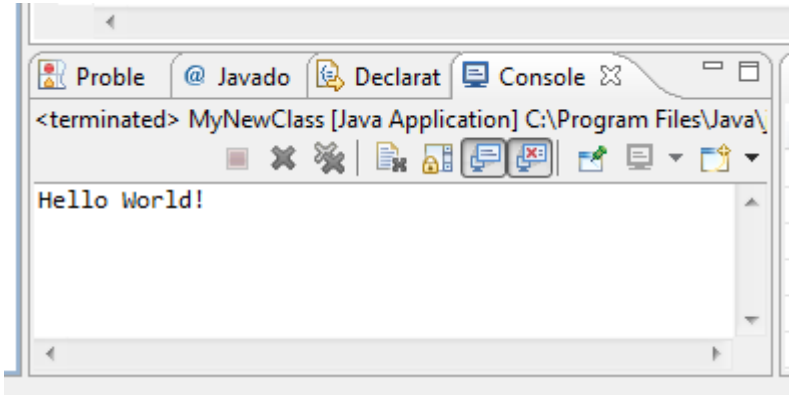
Hello World



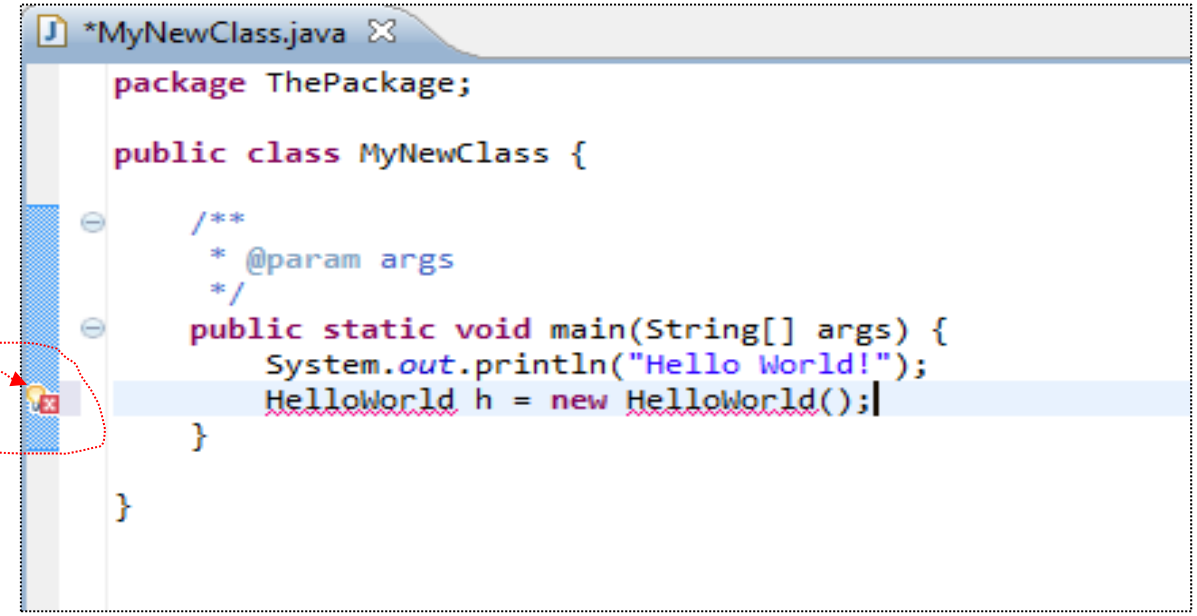


2. Select

Will not appear if class was saved



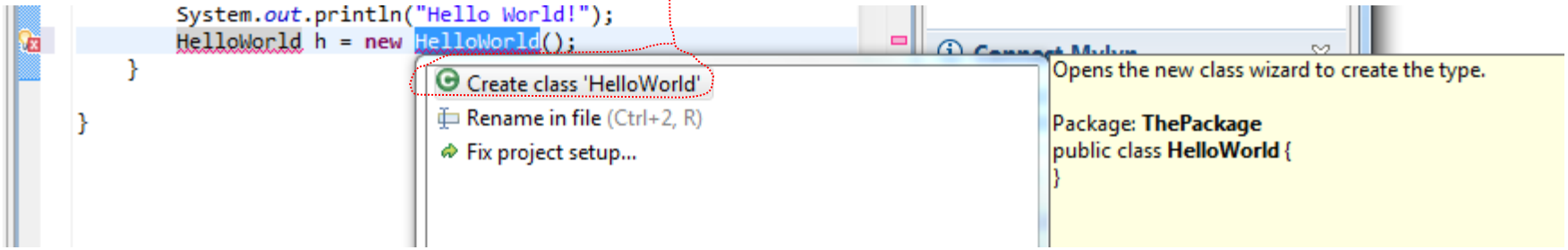
Another Class



1. Select

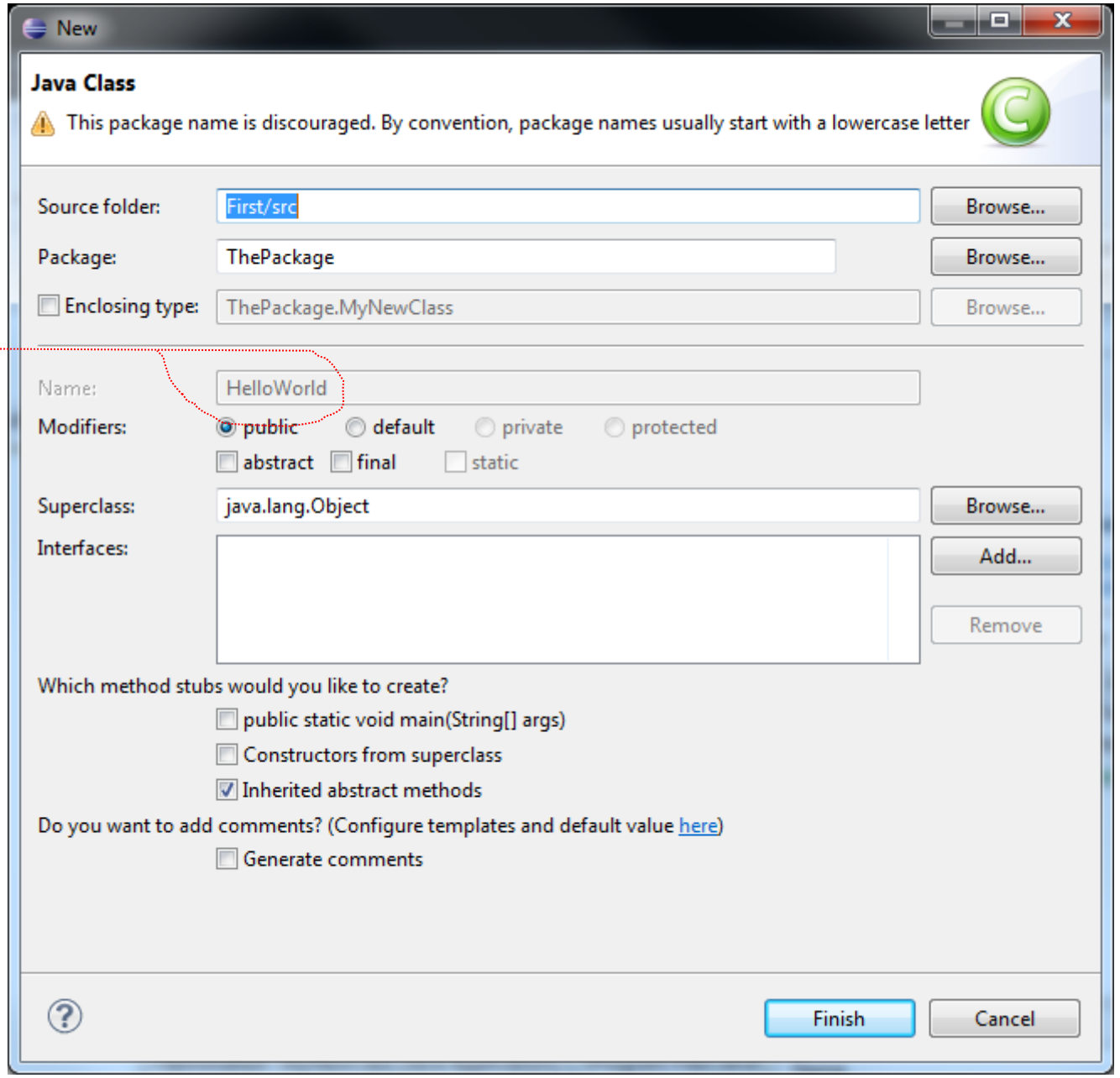
Class hasn't been written so it is complaining

2. Select



Another Class

Someone filled it in for us



Another Class New Code

*MyNewClass.java | HelloWorld.java

```
package ThePackage;

public class HelloWorld {

}
```

New Method in Old Class

*MyNewClass.java | HelloWorld.java

```
package ThePackage;

public class MyNewClass {

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("Hello World!");
        HelloWorld h = new HelloWorld();
        h.sayHelloWorld();
    }

}
```

New Method in HelloWorld Class

*MyNewClass.java | *HelloWorld.java

```
package ThePackage;

public class HelloWorld {

    public void sayHelloWorld() {
        // TODO Auto-generated method stub

    }

}
```

Adding Code

*MyNewClass.java | *HelloWorld.java

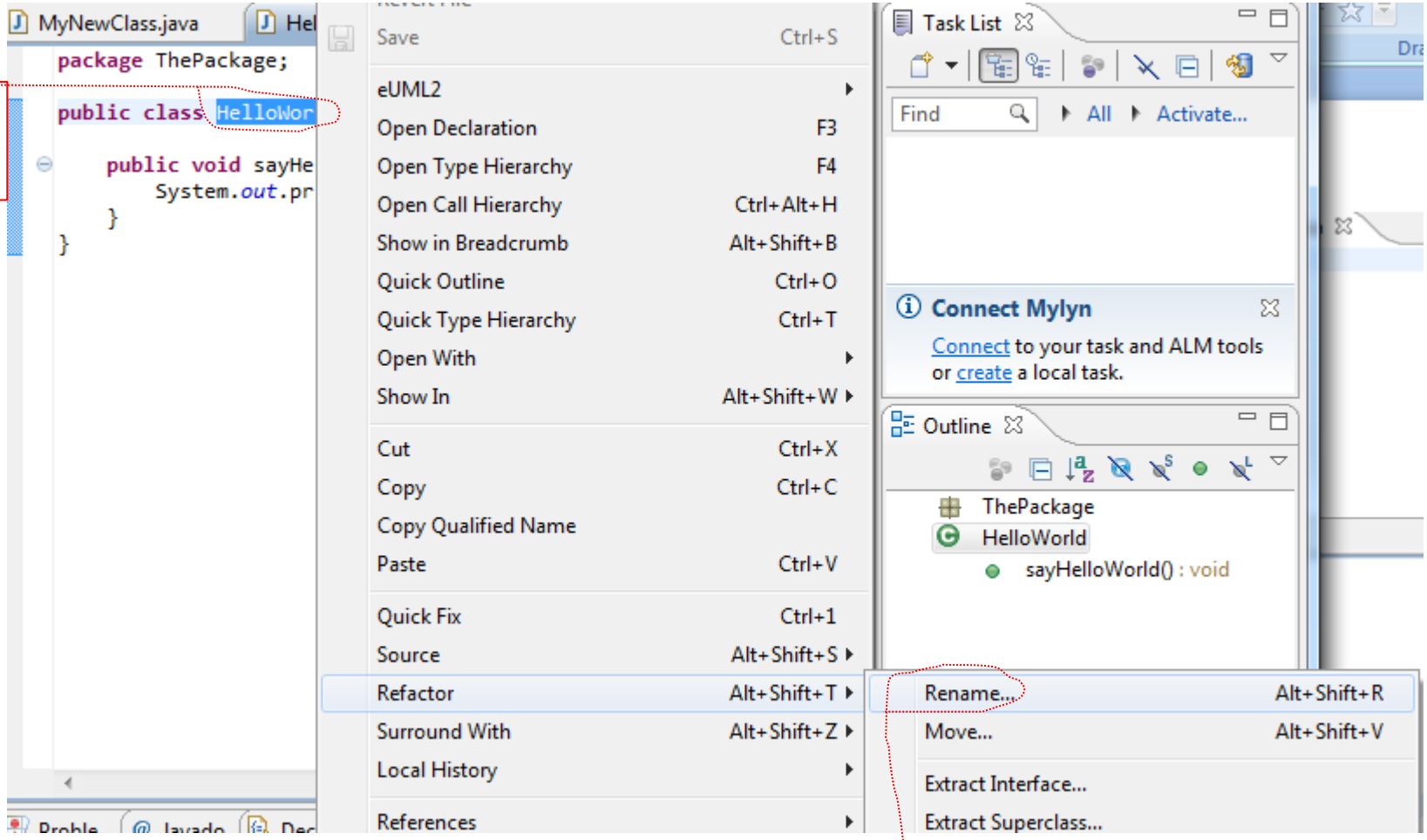
```
package ThePackage;

public class HelloWorld {

    public void sayHelloWorld() {
        System.out.println("Hello World!!!");
    }
}
```

Proble | Javado | Declarat | Console

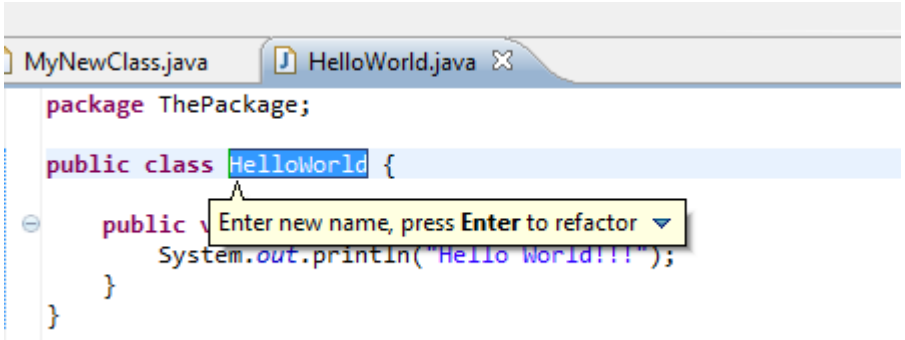
<terminated> MyNewClass [Java Application] C:\Prog

```
Hello World!
Hello World!!!
```

Refactoring - renaming



Refactor-Rename



New Code