CS 1302 – HW-Copilot

Contents

[1 Introduction 1](#_Toc168577351)

[2 Copilot4Eclipse Introduction 2](#_Toc168577352)

[3 Setup Copilot & Copilot4Eclipse 2](#_Toc168577353)

[4 Initialize Java Project 2](#_Toc168577354)

[5 Overview of Copilot4Eclipse 3](#_Toc168577355)

[6 Chat – Room Example 4](#_Toc168577356)

[7 Chat – OCSF Example 7](#_Toc168577357)

[8 Submission Requirements 7](#_Toc168577358)

[Appendix 1 JUnit in Eclipse 8](#_Toc168577359)

# Introduction

*(Read, no action required)*

[*GitHub Copilot*](https://github.com/features/copilot)is an AI tool that can be used to write, suggest, interpret, and test code. It lives at GitHub. Once you have a GitHub account, you can sign up for Copilot. As a student, it will be free, but you do have to go through a verification process. There will be instructions later for this.

“GitHub Copilot is an AI pair programmer that offers autocomplete-style suggestions as you code. You can receive suggestions from GitHub Copilot either by starting to write the code you want to use, or by writing a natural language comment describing what you want the code to do. GitHub Copilot analyzes the context in the file you are editing, as well as related files, and offers suggestions from within your text editor…

GitHub Copilot is trained on all languages that appear in public repositories. For each language, the quality of suggestions you receive may depend on the volume and diversity of training data for that language. For example, JavaScript is well-represented in public repositories and is one of GitHub Copilot's best supported languages.”[[1]](#footnote-1)

The objective of this lab is to learn to use Copilot. You can use it on GitHub [not currently incorporated into this lab] and you can use it through an IDE. Here, you will use the [*Copilot4Eclipse*](https://www.genuitec.com/products/copilot4eclipse/)(Eclipse) plugin to access GitHub Copilot to generate code, *etc.*

# Copilot4Eclipse Introduction

*(Read, no action required, 20 min)*

*Copilot4Eclipse* is a plugin for Eclipse to use GitHub’s Copilot AI tool. In this section, you will read a few pages and watch a few short videos to get an overview of the features and how to use them in Copilot4Eclipse (C4E). Do the following:

1. Watch 2-minute [Copilot4Eclipse video](https://www.youtube.com/watch?v=q2dhnulEYFk) which overviews: installation, authorization, code suggestions, and customization/preferences. It does not discuss *Chat* which will be in another video. It was helpful for me to slow the video speed, something I don’t think I’ve done before.
2. Scan this overview of the [Copilot4Eclipse UI](https://www.genuitec.com/products/copilot4eclipse/docs/basics/overview/). You can stop at Section 2.1, you will see Menus later.
3. Scan this overview of [Inline Code Completions](https://www.genuitec.com/products/copilot4eclipse/docs/basics/completions/). Read these [tips](https://www.genuitec.com/products/copilot4eclipse/docs/basics/completion-tips) for generating better code suggestions.
4. Scan this overview of [Chat](https://www.genuitec.com/products/copilot4eclipse/docs/basics/chat/) and be sure and watch the video there.

# Setup Copilot & Copilot4Eclipse

Follow these steps to access GitHub Copilot, Copilot4Eclipse, set them up, and connect them.

1. (You may have already done this step) Do the following to obtain access to GitHub Copilot:
2. Create a GitHub account. You will create a username, a password, and specify your email address. [https://github.com/](https://nam05.safelinks.protection.outlook.com/?url=https%3A%2F%2Fgithub.com%2F&data=02%7C01%7Cdgibson%40valdosta.edu%7C4dad75e3ebdc46ab47e208d7d03041a0%7C25a5d3408abc4053b4bddc1213280353%7C0%7C0%7C637206777989958306&sdata=PZKNTqsemFRE6xTle2Uu3aBxyyQYOdF2ToYFrtq3sUo%3D&reserved=0)
3. Follow [steps 1-8](https://techcommunity.microsoft.com/t5/educator-developer-blog/step-by-step-setting-up-github-student-and-github-copilot-as-an/ba-p/3736279). At the conclusion, you will be waiting for your application for Student status to be approved. Once it is approved, continue with the link above, for one more step, *Setting Up Copilot*.
4. Do the following to install Copilot4Eclipse Plugin:
5. Verify that your computer meets the Copilot4Eclipse [prerequisites](https://www.genuitec.com/products/copilot4eclipse/docs/prereqs/prerequisites).
6. Follow these steps to [install](https://www.genuitec.com/products/copilot4eclipse/docs/installation/). Note that on the last step (4), the image shows the Copilot menu, which at the top says, “Status: Copilot4Eclipse ready”. Yours will **not** say that. Yours will say, “Status: GitHub Copilot4Eclipse sign in required”. After you sign in, it will say, “…ready”.
7. Follow these steps to have Copilot4Eclipse [sign in](https://www.genuitec.com/products/copilot4eclipse/docs/basics/signin) to GitHub Copilot.

# Initialize Java Project

Do the following in Eclipse:

1. Create a project named: *labAILastname*.
2. Create a package named: *prob1*
3. Create a class named: *Room*
4. Create a [JUnit test class](#Appendix_JUnitInEclipse) named: *RoomTest*

# Overview of Copilot4Eclipse

*(Read, no action required)*

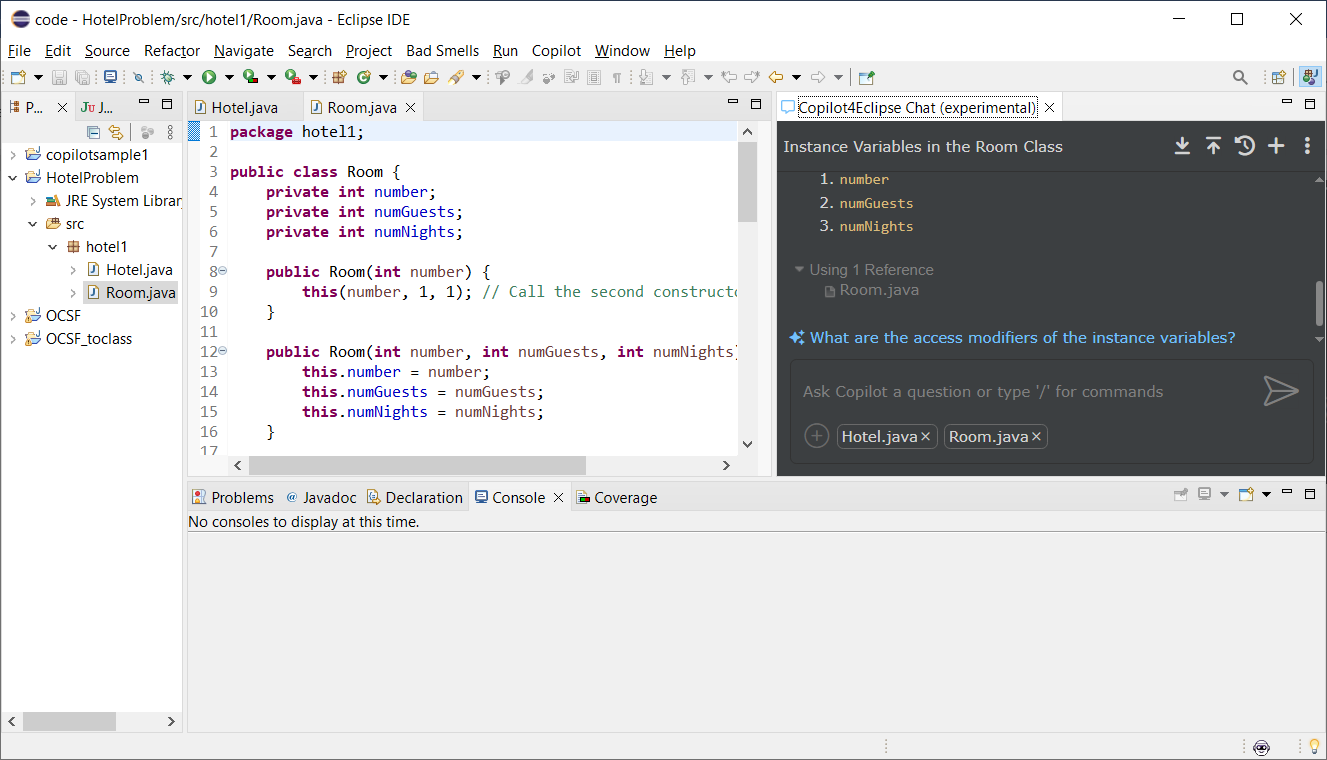
There are two main Copilot4Eclipse has: ghost text and Chat. In the steps that follow, we only show ghost text once; however, it is pretty obvious: you write code and it anticipates what you want. In this lab, we mostly focus on the Chat feature.

1. As discussed in one of the readings or videos above, Chat will answer your question with a frame of reference. If not explicitly stated, it will typically just assume your active, open code window(s). The “+” at the bottom-left of the dialog on the left allows you to explicitly define the context.

For example, I have a *Room* class open and ask Chat a question. The figure on the right shows the reference it used (toward the middle of the image): *Room.java*.

|  |  |
| --- | --- |
|  |  |

The figure below (at the bottom left), shows that I have added explicit references to the *Hotel* and *Room* classes for the next Chat.



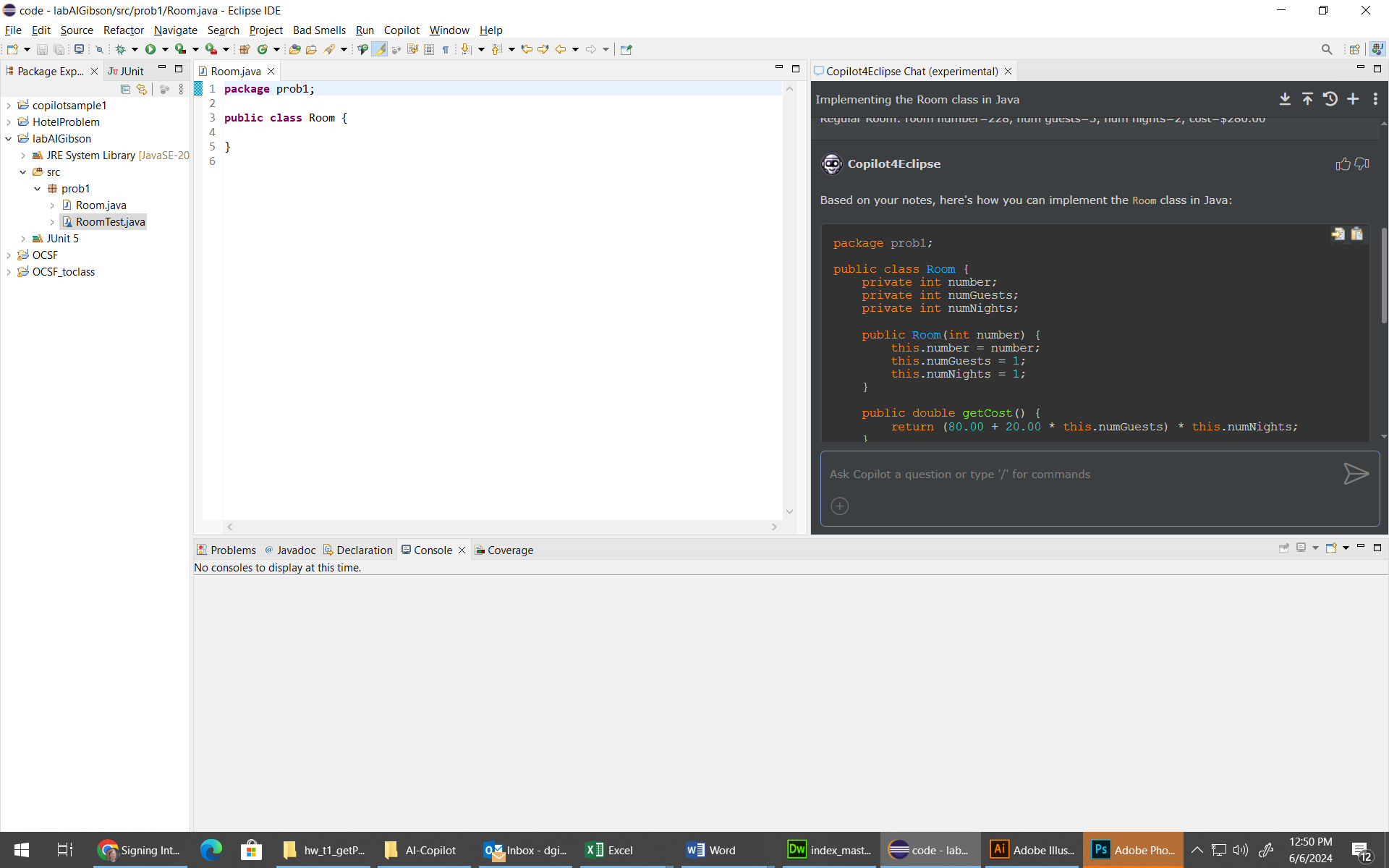
1. A *conversation* occurs when you ask questions repeatedly about some context, and as I understand it, the context is not only the references you define or it infers, but also the conversation itself. In the upper-right is another “+” which will start a new conversation. To the left of this, the reverse, circular arrow, shows the conversations. There is not a way to export conversations.
2. [Best practices for using GitHub Copilot in VS Code](https://code.visualstudio.com/docs/copilot/prompt-crafting) – This is for the VS Code plugin, but it has a Chat feature that is virtually the same as Copilot4Eclipse. These are really good suggestions.

# Chat – Room Example

Do the following:

1. Copy the text below and past directly into Chat and Enter.

|  |
| --- |
| Notes on *Room* class:   * *number* – the room number (any integer) * *Room(number:int)* – Creates a room with 1 guest for 1 night, with room number, *number.* * *equals* – rooms are considered equal if they have the same (room) *number.* * *getCost –* The cost per night for a regular room is $80.00 plus $20.00 per guest, and this value is then multiplied by the number of nights. * *toString() –* returns a string like this (should be a single line):   Regular Room: room number=228, num guests=3, num nights=2, cost=$280.00 |

1. You’ll see that it creates a Room class. You can copy (or insert) in your code window using the two icons in the upper-right of the Copilot code window as shown below. The first icon, inserts the code where the cursor is in the actual code window. The other icon copies the code to the clipboard.

**Copy/Insert the generated code into your class.**

1. Tell Chat:

*Add a constructor that accepts all three instance variables and have the first constructor call the second*

Copy and paste the result in. I would just manually, copy/paste the two constructors into the code, overwriting the existing one.

Note: you have to be careful copy/paste a whole class it generates, based on your modifications (*i.e.* a continuing conversation), because sometimes the class will be different, or leave things out. However, it generally does what you ask it to do.

1. In the code window, start a new method by typing (and noting the ghost text):

*public Boolean isLong*

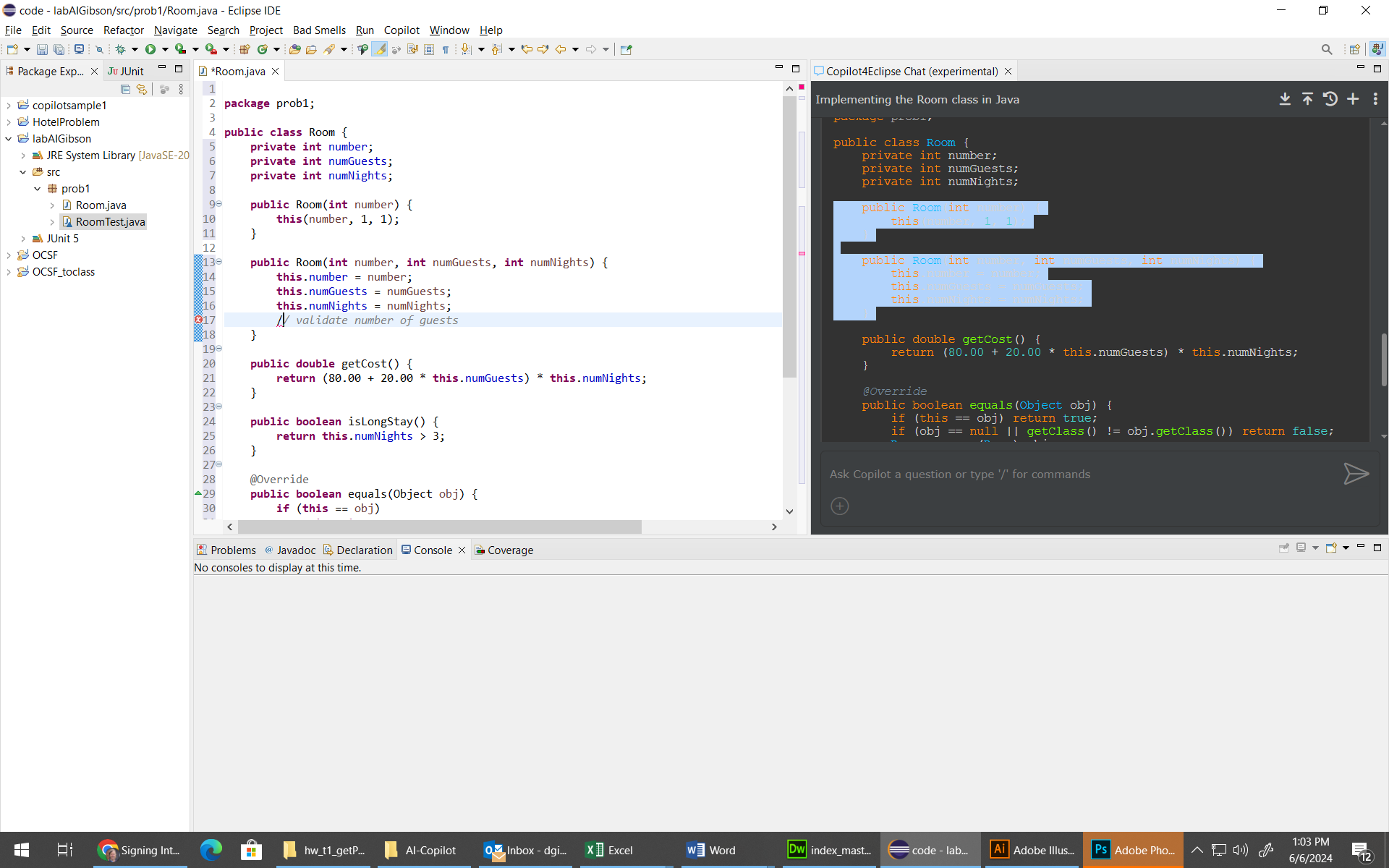
Eventually, the ghost text will show something like this, which you should accept.

**public** **boolean** isLongStay() {

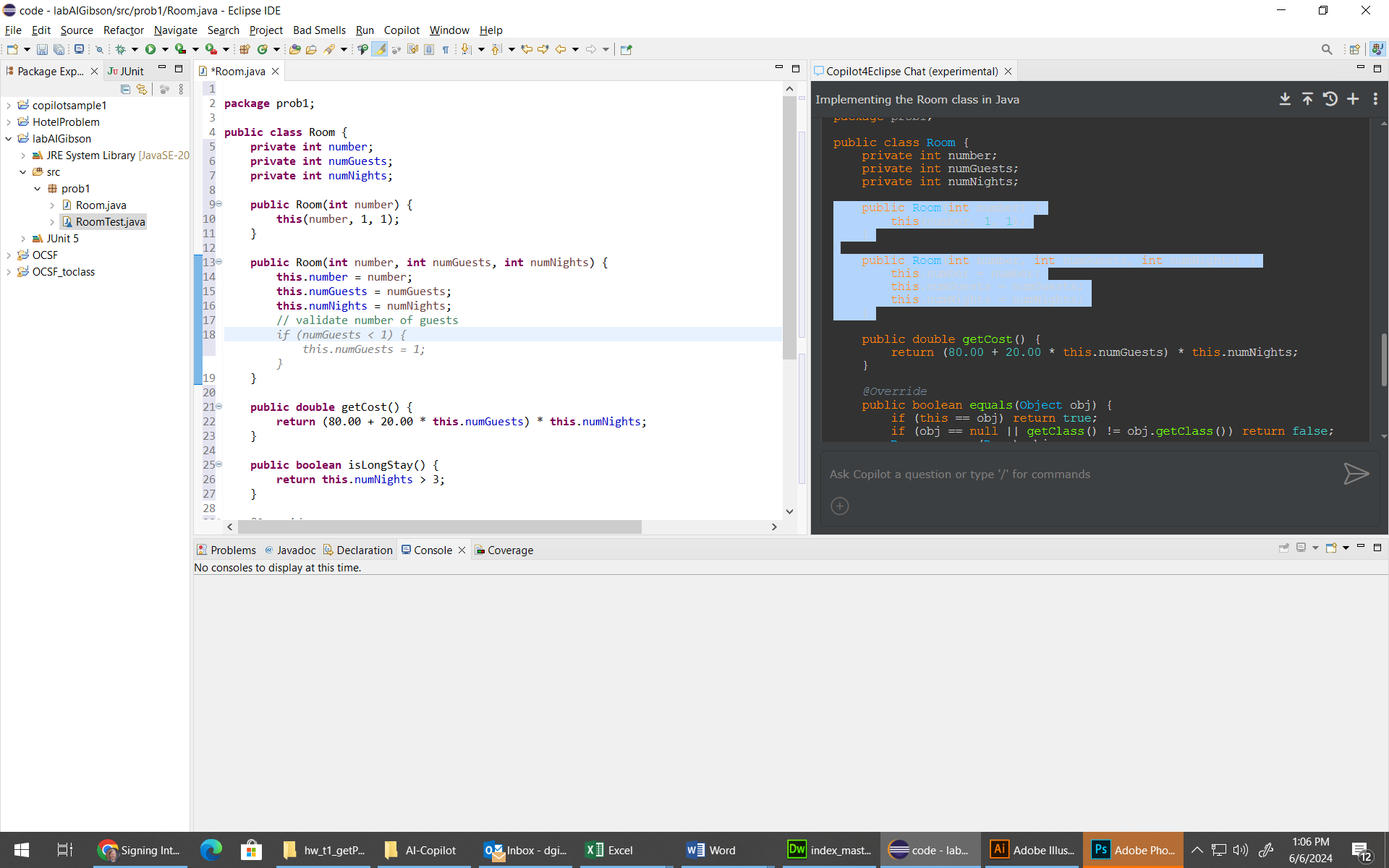
**return** numNights > 2;

}

1. Go to the end of the *Room* constructor that defines three parameters. Type a single: “/”. This is a way to explicitly invoke Copilot. On my computer, it is as shown below. What ever it shows on yours, accept it.



After accepting, your cursor will be at the end of the line. Press Enter and some suggested code will appear. Accept it.



1. Do the following:
2. Highlight the *getCost* method in the code window.
3. Then, in Chat, type: “/” and you will see options. Choose: “/explain”. You could type something after this, but instead we will just press Enter in which case it is operating on the selected text.

Read the description.

1. Tell Chat:

*Change getCost so that the there is only a 20 charge when there are more than 2 guests*

When I did this yesterday, I got:

public double getCost() {

double guestCharge = (numGuests > 2) ? 20.00 \* (numGuests - 2) : 0.00;

return (80.00 + guestCharge) \* numNights;

}

I then told it to: “simplify” and got:

public double getCost() {

return 80.00 \* numNights + ((numGuests > 2) ? 20.00 \* (numGuests - 2) : 0.00) \* numNights;

}

This was not what I wanted, so I said, “don't use the thing with the ?” which gave me what I was looking for:

public double getCost() {

double guestCharge = 0.00;

if (numGuests > 2) {

guestCharge = 20.00 \* (numGuests - 2);

}

return (80.00 + guestCharge) \* numNights;

}

When I did it today, I got closer to the last one above.

1. Play with this for a while. Experiment!
2. Tell Chat:

*Write a main method to do informal testing*

Then tell it:

*Don’t explicitly call toString*

Then tell it:

*Call getCost*

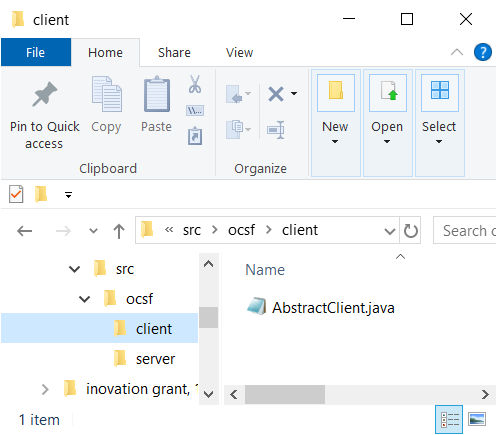
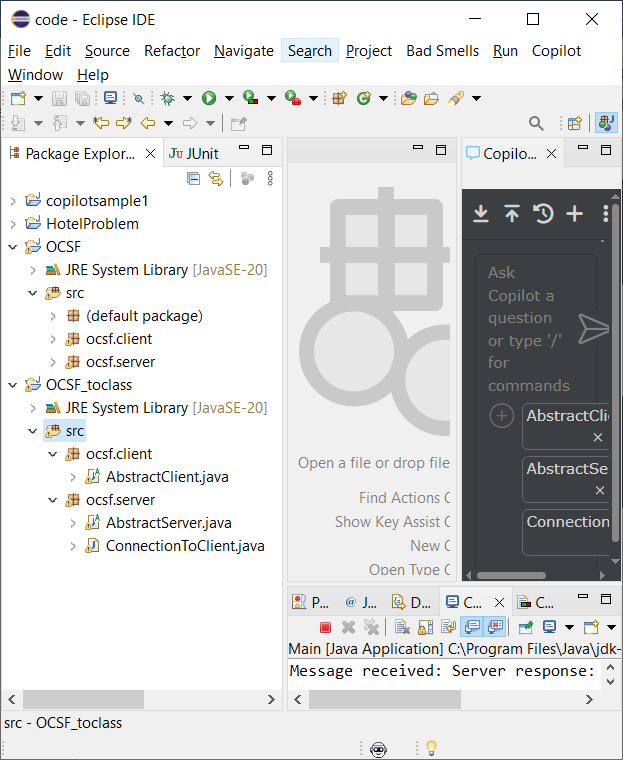
In this case, it lost the previous code in *main.*

**Experiment for a bit.**

1. Type: “/” into Chat and choose: tests. Add the generated tests to *RoomTest* (be careful, it seems to not declare the package name in the test class)

# Chat – OCSF Example

Next, we show an example of explaining a piece of code.

1. Create a new project in Eclipse named: *OCSF*
2. Download and unzip, *hw-copilot.zip* from the Schedule. Inside, you will find an *ocsf* folder that contains two subfolders: *client* and *server.* Inside these are several Java files.
3. Drag the *ocsf* folder into the *src* node the the Package Explorer in Eclipse.
4. Issue this prompt in Chat: /explain what is the purpose of these classes?
5. Read through the description thoroughly.
6. Issue this prompt in Chat: show an example of how to use these classes
7. Read through the description thoroughly.

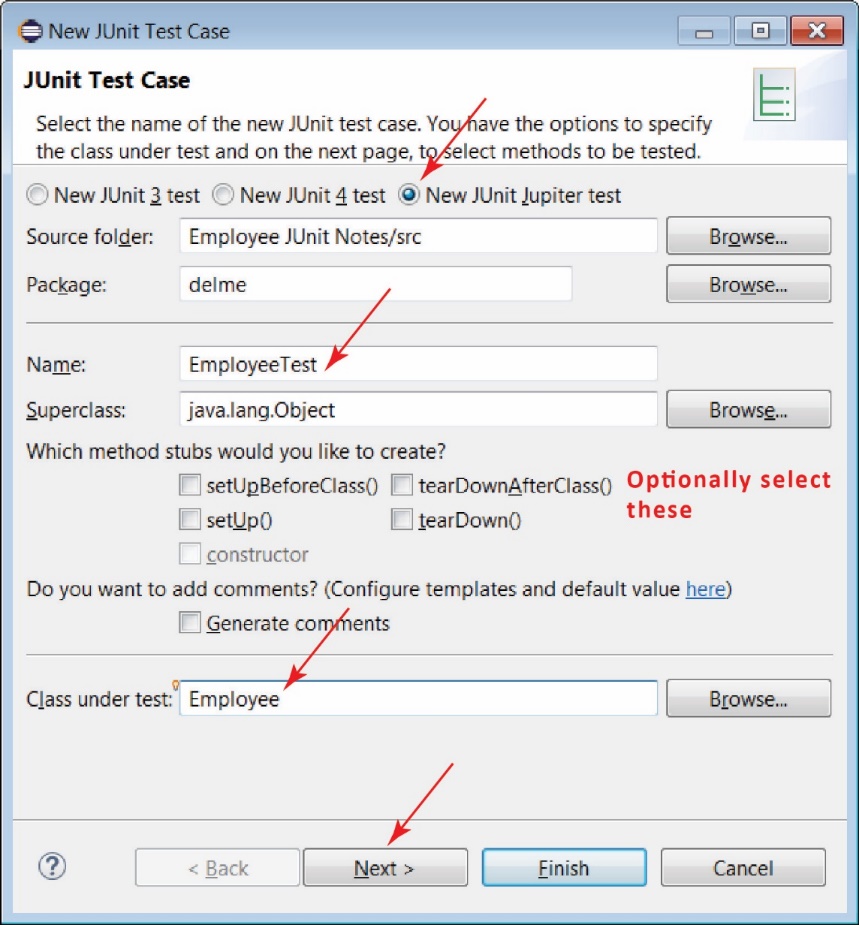
The resulting *Main* class will run and illustrate using the framework. However, it will only work once! It doesn’t close the client connection nor close the server. If you do these things, then the client will be stopped before it receives its reply from the server. Thus, in a real application, you would close the connection on demand. You can run more times incrementing the port for the server constructor; however, I think something still doesn’t work right.

# Submission Requirements

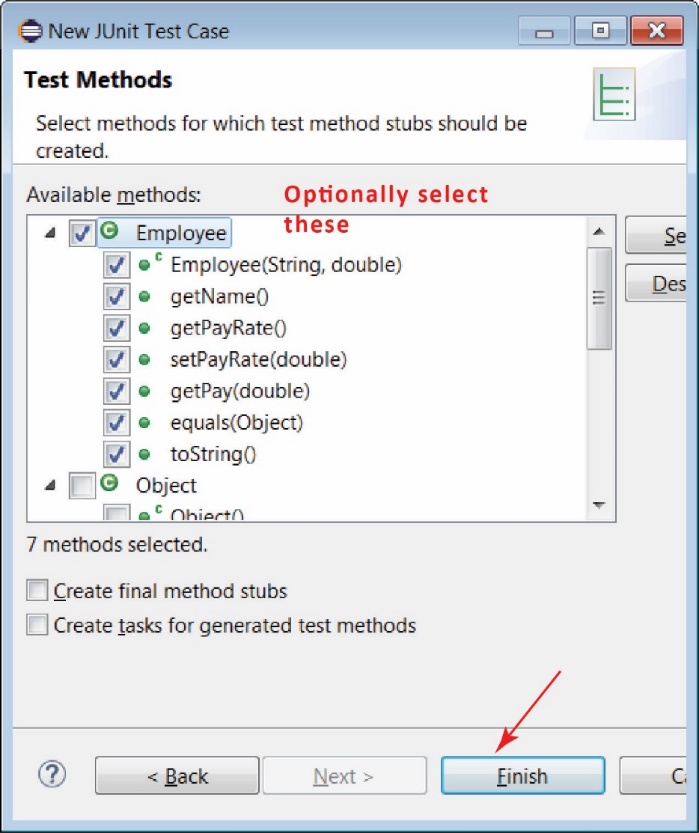
Submit your *Room.java* and *RoomTest.java* files in the *HW-Copilot* dropbox on Blazeview by the deadline

Appendix

1. JUnit in Eclipse

JUnit 5 (also called JUnit Jupiter) is the most current framework for doing unit testing in Java. JUnit 5 support is built-in to Eclipse since 2017.

To create a JUnit test class in Eclipse, choose:

1. File, New, JUnit Test Case
2. Make sure *New JUnit Jupiter test* is selected
3. Supply a class name. The convention for naming test classes is: *ClassNameTest*. For example shown on the right, the class under test is *Employee* and the test class name is: *EmployeeTest*
4. Specify the class under test (not necessary, but then it won’t generate test method stubs for you as in the next step)
5. Next.
6. On the next dialog, you can choose the methods you want to test and it will create test method stubs for these. I sometimes do this as a reminder of all the methods. But, two points: (1) we don’t have to test all methods, we only test methods that have some logic (we say more about this later), (2) there will be several or more tests for each method that we do test.
7. The resulting test class will look like this:

**class** EmployeeTest {

@Test

**void** testEmployee() {

*fail*("Not yet implemented");

}

@Test

**void** testGetName() {

*fail*("Not yet implemented");

}

...

}

1. <https://docs.github.com/en/copilot/copilot-individual/about-github-copilot-individual> [↑](#footnote-ref-1)