

# Supercharge Java Development with GitHub Copilot Custom Instructions in IntelliJ!

Here's a **step-by-step guide** for setting up **GitHub Copilot Custom Instructions** to tailor your Java code generation preferences.

## Objective

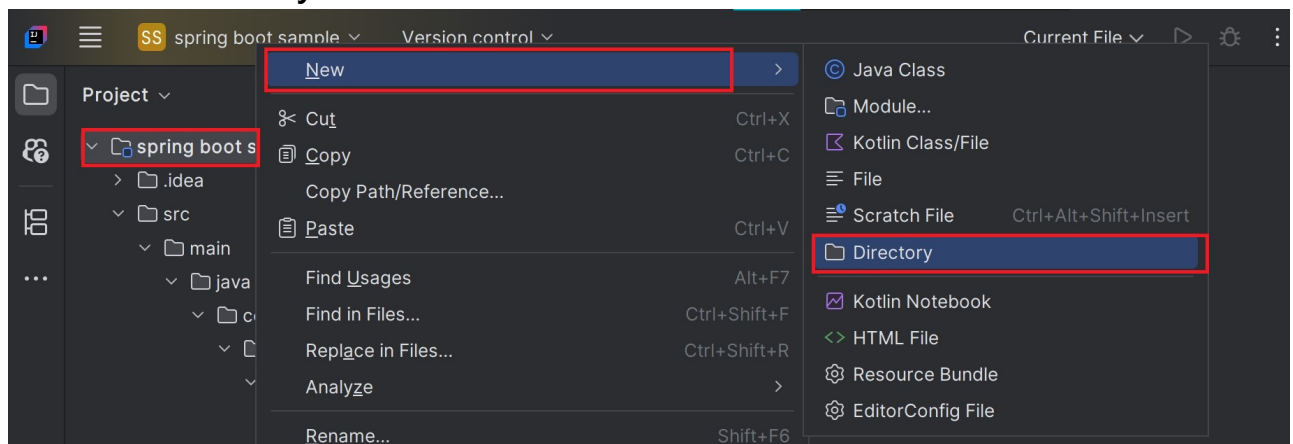
Ensure GitHub Copilot:

1. Adds `@Override` to all overridden methods.
2. Initializes loggers using `LoggerFactory.getLogger`.
3. Adds Javadoc, constructor, getter/setter methods by default.

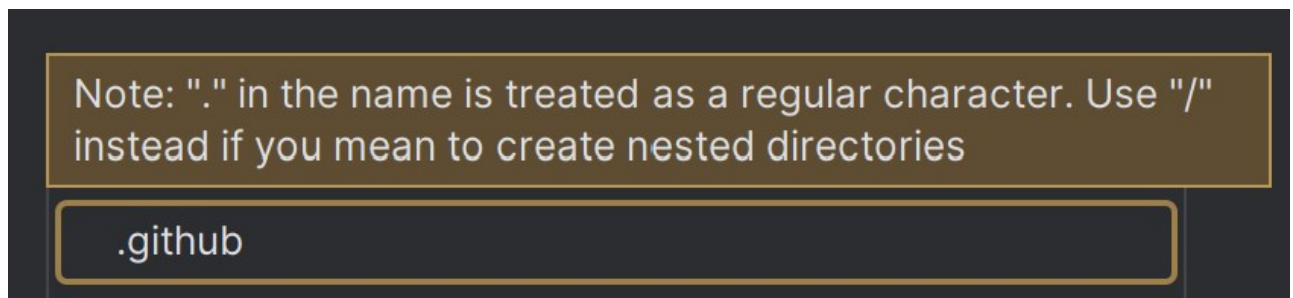
## Steps to Set Up Copilot Custom Instructions for Java

### Step 1: Create `.github` Directory

- Navigate to the **root directory** of your project.
- **Right-click** on the root folder in the Project Explorer.
- Select **New** → **Directory** from the context menu.



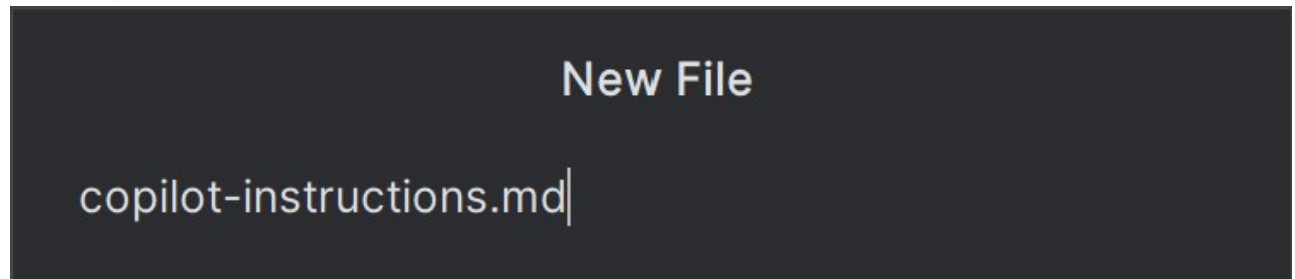
- Enter the folder name as:



⚠ Make sure to include the dot (.) at the beginning. This makes it a hidden directory in Unix-like systems.

### Step 2: Create `copilot-instructions.md` File

- Inside `.github`, create a file:



### Step 3: Add Your Custom Instructions

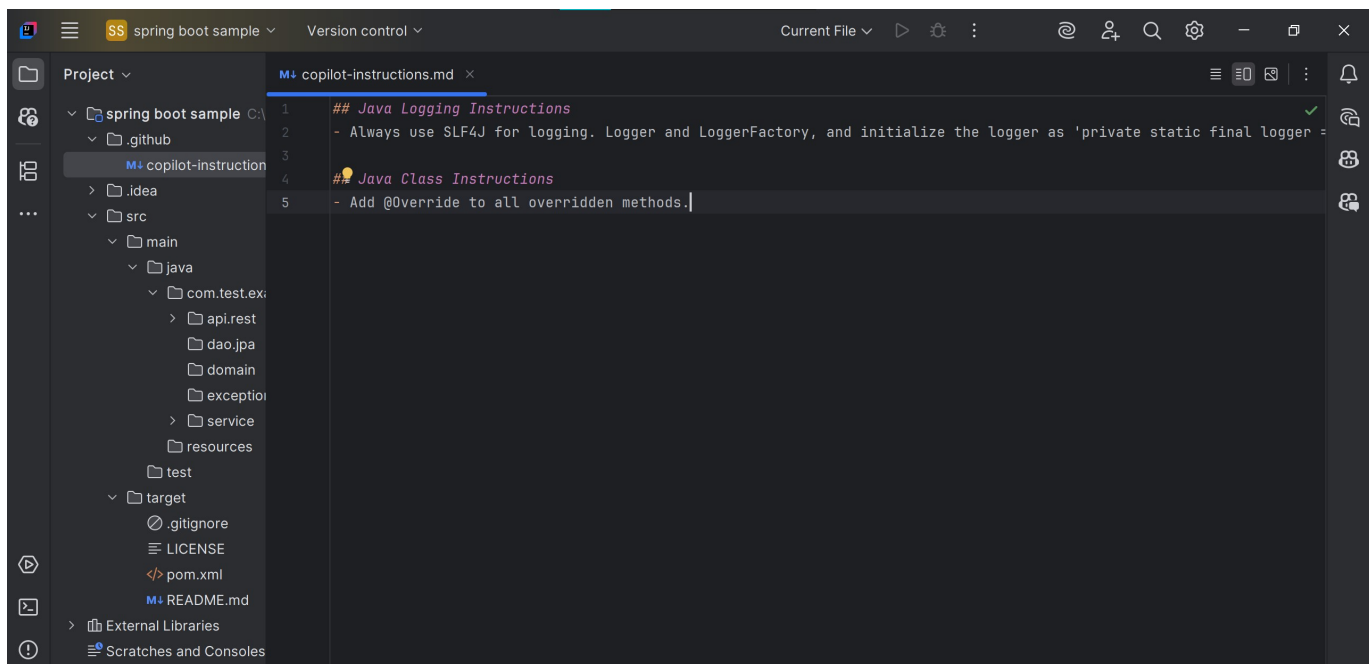
Paste the following content inside the `copilot-instructions.md` file:

#### **\*\*Java Logging Instructions\*\***

- Always use SLF4J for logging. Logger and LoggerFactory, and initialize the logger as 'private static final logger = LoggerFactory.getLogger(ClassName.class);'.

#### **\*\*Java Class Instructions\*\***

- Add `@Override` to all overridden methods.



### Example Prompt:

Ask Copilot in your IDE:

"create a sample Java class the implements an interface"

create a sample Java class the implements an interface



GPT-4o



### Copilot Output:

The screenshot shows the VS Code interface with a file named `copilot-instructions.md` open. The file contains two sections of instructions:

```
1 ## Java Logging Instructions
2 - Always use SLF4J for logging. Logger and LoggerFactory, and in
3
4 ## Java Class Instructions
5 - Add @Override to all overridden methods.
```

The Copilot output panel on the right shows the generated Java code, which is titled "Creating Java Classes and Interfaces". The output includes the following code:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

interface SampleInterface {
    void performAction();
}

public class SampleClass implements SampleInterface {
    private static final Logger logger = LoggerFactory.getLogger(SampleClass.class);

    @Override
    public void performAction() {
        logger.info("Action performed successfully");
    }
}
```

The code is highlighted with red boxes. Below the code, there is a question: "Can you show me how to add a new method to the interface?". At the bottom of the panel, there is a text input field with the placeholder "Ask Copilot a question or type '/' for commands", a plus icon, and a dropdown menu showing "GPT-4o".

### Java Class Instructions

## ## Java Class Instructions

Whenever generating a new Java Class:

- Add JavaDoc for every class and method.
- Include a private logger using SLF4J.
- Add a constructor initializing all fields.
- Use Lombok for getters/setters where possible.
- Make code clean, readable, and follow Java best practices.
- Add `@Override` to all overridden methods.

Whenever generating a new Java Class:

- Add JavaDoc for every class and method.
- Include a private logger using SLF4J.
- Add a constructor initializing all fields.
- Use Lombok for getters/setters where possible.
- Make code clean, readable, and follow Java best practices.
- Add `@Override` to all overridden methods.

---

### Step 4: Use GitHub Copilot With These Instructions

You don't need to reference the file manually every time. Once it's in the project, GitHub Copilot uses it automatically as context.

#### Example Prompt:

Ask Copilot in your IDE:

```
"Create a Java class named Booking that implements a Payment interface."
```

#### Copilot Output:

Ask Edit

Implementing Java Classes and Interfaces

Here is the **Booking** class implementing the **Payment** interface, following the provided coding instructions:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import lombok.Getter;
import lombok.Setter;

/**
 * Represents a booking that implements payment
 */
@Getter
@Setter
public class Booking implements Payment {
    private static final Logger logger = LoggerFactory.getLogger(Booking.class);

    private String bookingId;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import lombok.Getter;
import lombok.Setter;

/**
 * Represents a booking that implements payment functionality.
 */
@Getter
@Setter
public class Booking implements Payment {
    private static final Logger logger = LoggerFactory.getLogger(Booking.class);

    private String bookingId;
    private double amount;
```

```
/**
 * Constructor to initialize all fields.
 *
 * @param bookingId the unique ID of the booking
 * @param amount    the payment amount for the booking
 */
public Booking(String bookingId, double amount) {
    this.bookingId = bookingId;
    this.amount = amount;
}

/**
 * Processes the payment for the booking.
 */
@Override
public void processPayment() {
    logger.info("Processing payment of ${} for booking ID: {}", amount,
bookingId);
}
}
```