Hug the Rails: Internet of Things

User Manual

Group 15

CS 347-D

Jason Ruan, Tomasz Borowiak, Maddie Johnson

Table of Contents

1) INTRODUCTION

1.1 Purpose

The purpose of the test file is to rigorously test IoT in a controlled environment in preparation for its real world release. Preset functions in the test file allows the technician to manually feed sensor data into IoT, enabling a fast, organized, and controlled approach to simulate IoT. Whether to purposely break IoT to detect software faults, or to compare the simulation to the expected output requirements setup by the shareholders of HTR, the role of the test file is imperative in ensuring the quality and satisfaction of the final IoT product.

1.2 Requirements

1.2.1 Java IDE

Before modifying and running the test file, it is recommended to use an installation of the Eclipse IDE for Java. The IoT software was programmed and compiled entirely in Java, so it is imperative to use an appropriate Java IDE. Although other Java IDEs may successfully run and compile the IoT software, we highly recommend the Eclipse IDE for Java because the rest of the documentation and demonstration in this manual uses Eclipse IDE for Java. As a result, reproducibility and correctness of the subsequent test file hinges on your familiarity and experience with your preferred IDE and Java proficiency.

1.2.2 IoT Software

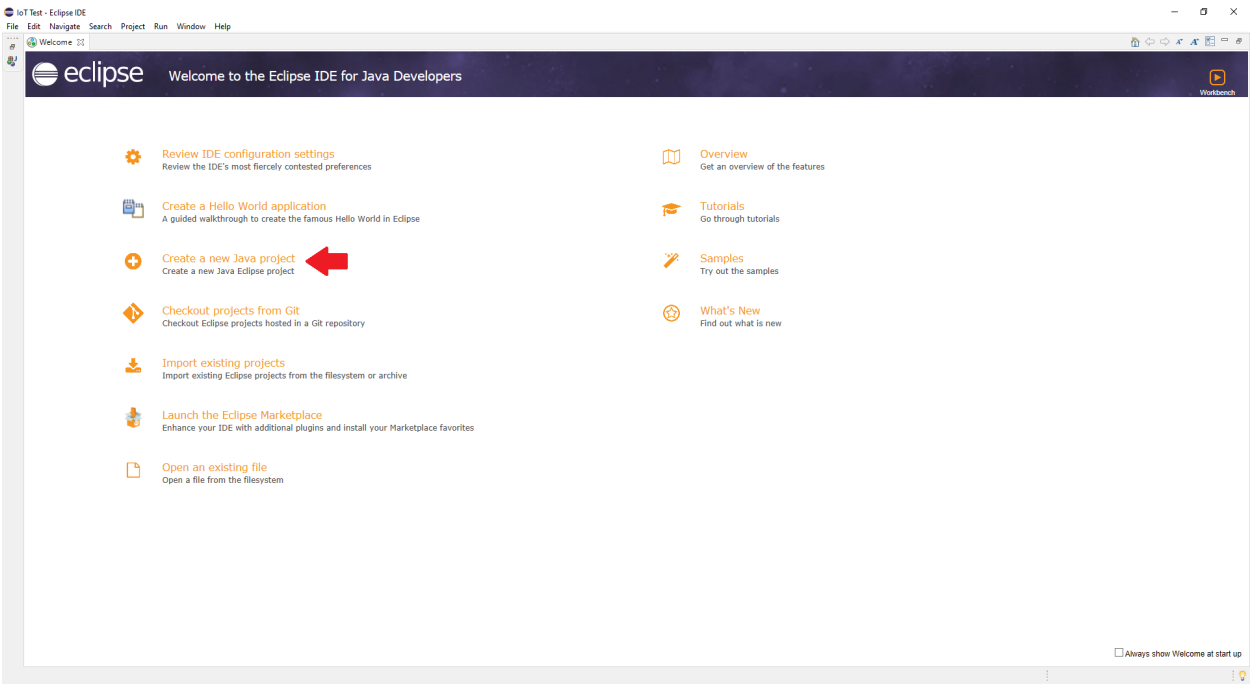
Additionally, the entire IoT software will need to be downloaded for the test file to run properly. To download IoT's software, head to our GitLab repository. Then, open the folder named "Version 5 Code", which contains IoT's software. Download all files in this folder onto your computer.

2) TEST FILE MODIFICATION

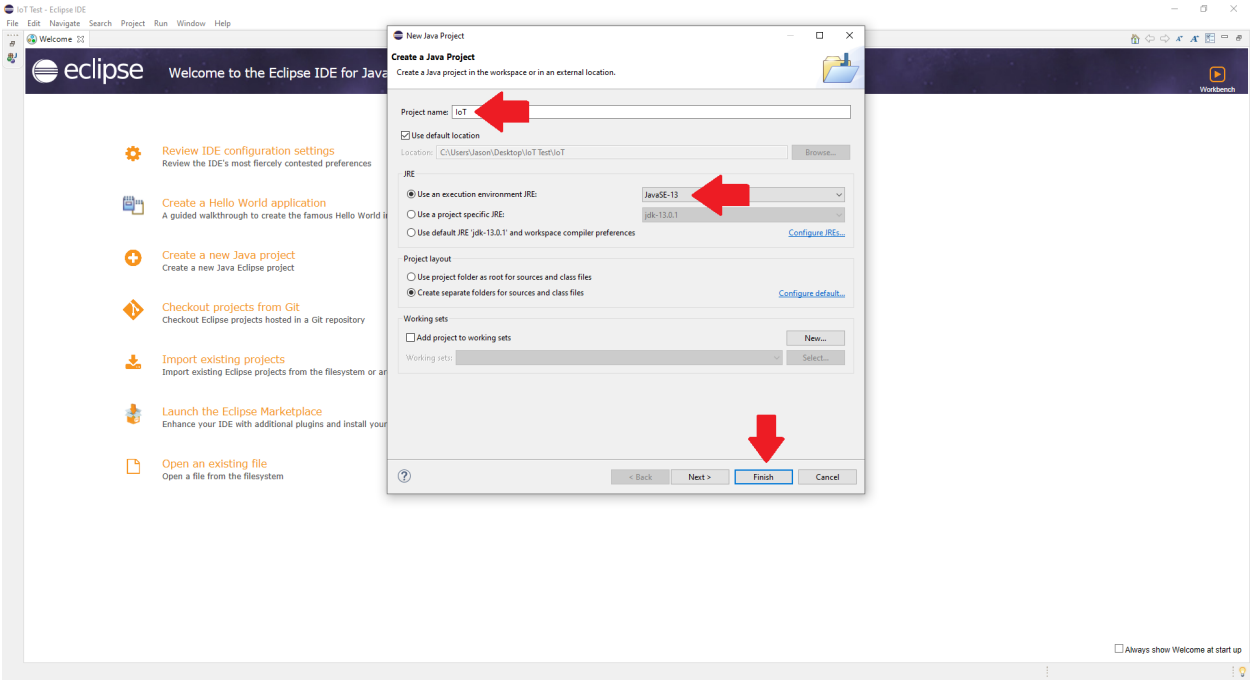
2.1 Installation

In this section, we will be importing the IoT software into the Eclipse IDE. If you have already installed the software, you may skip to the next section.

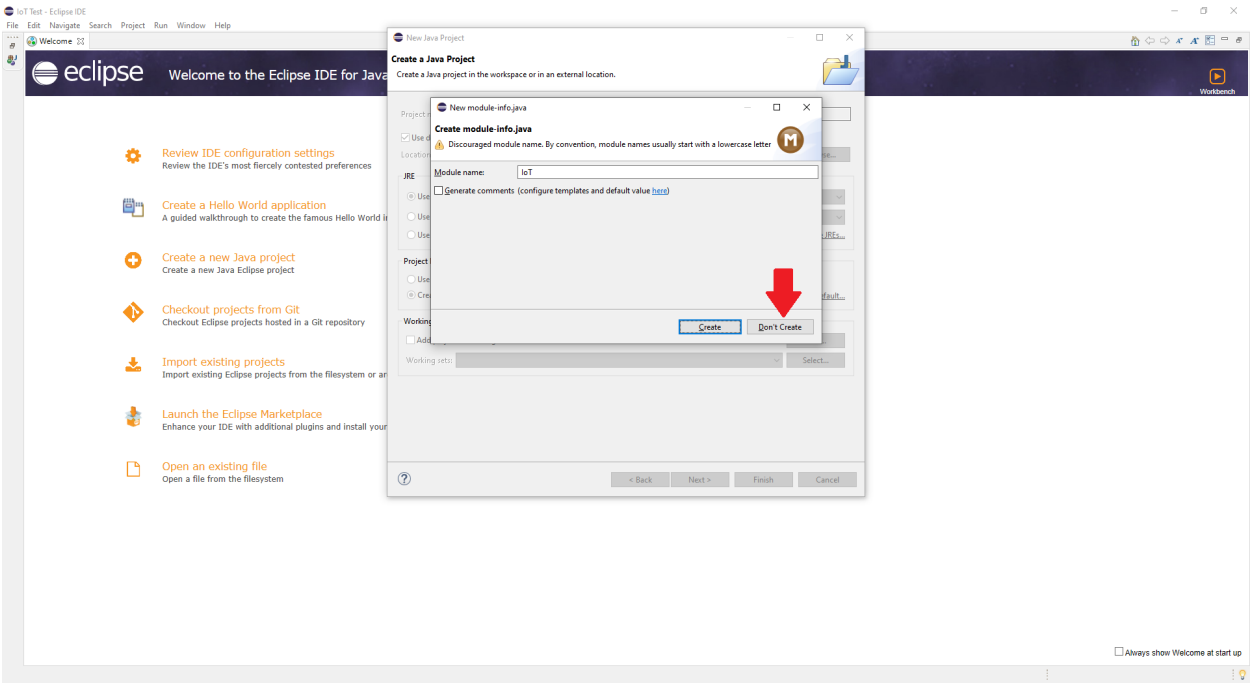
1. Open up Eclipse IDE for Java. We will be starting a new workbench. In this case, we named the workbench **IoT Test**. On the workbench screen, click the button **Create a new Java project**.



2. Then, a window will pop up named **New Java Project**. Change the project name to **IoT**, and select the execution JRE to be **JavaSE-13**. Press the **Finish** button.

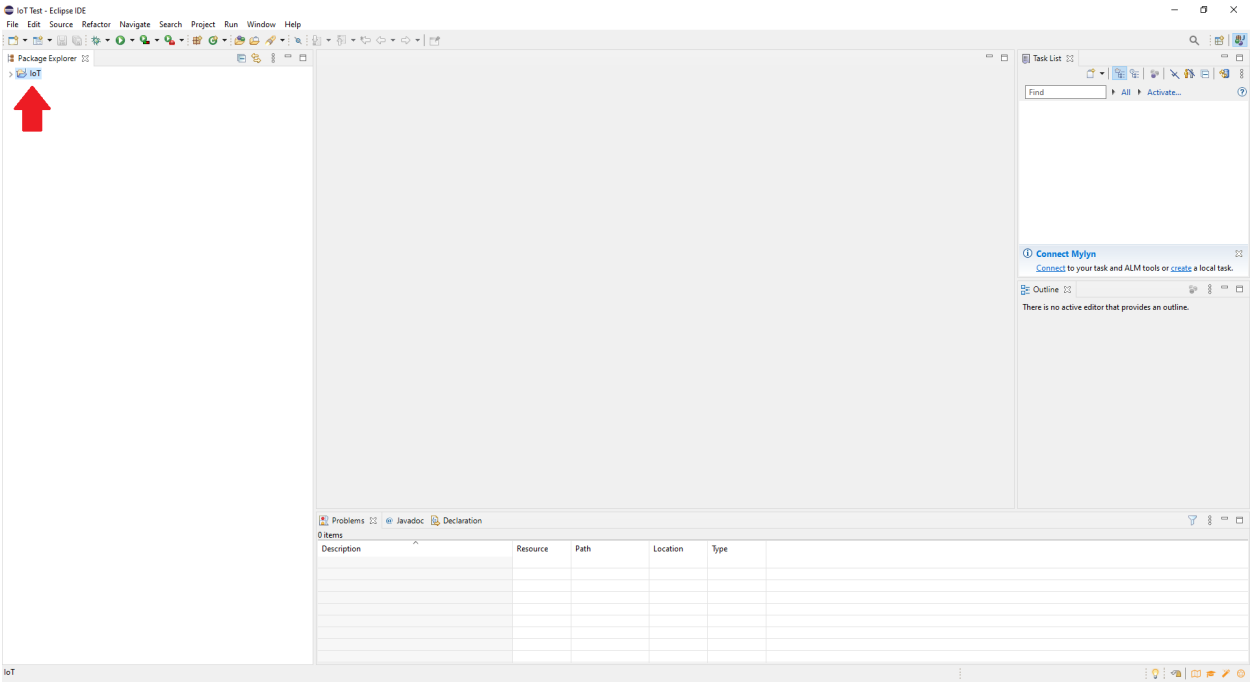


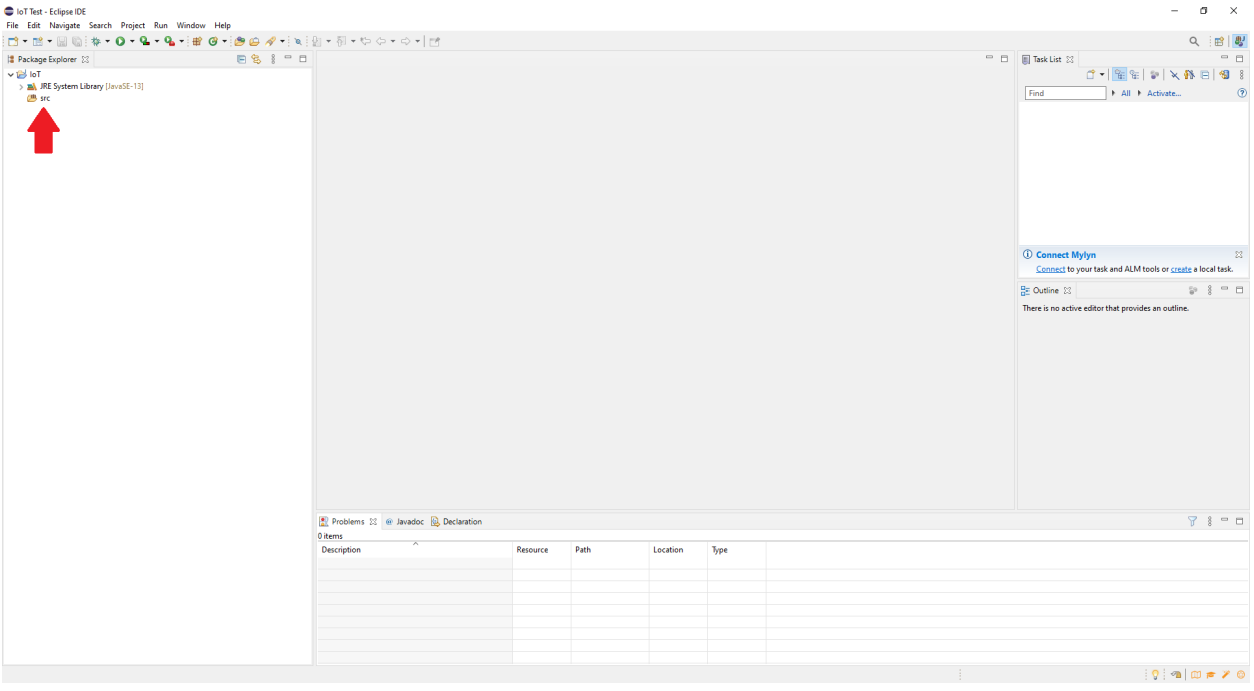
3. Another window will pop up named **New module-info.java**. Click on the **Don't Create** button.



4. The main project space should now show up after creating the new Java Project. On the left, you should see a sidebar named **Project Explorer** with a folder icon for **IoT**. Double clicking on the **IoT** folder should now display a

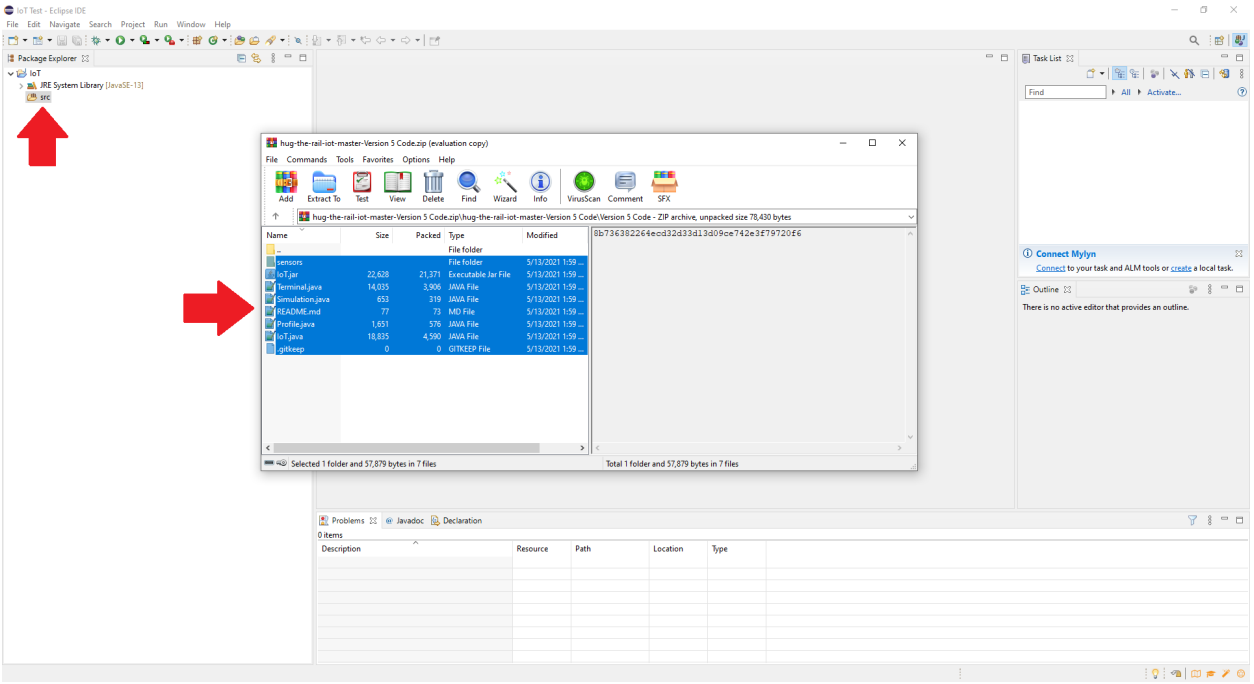
subtree of the **IoT** folder with a JRE library named **JRE System Library** and a folder named **src**.

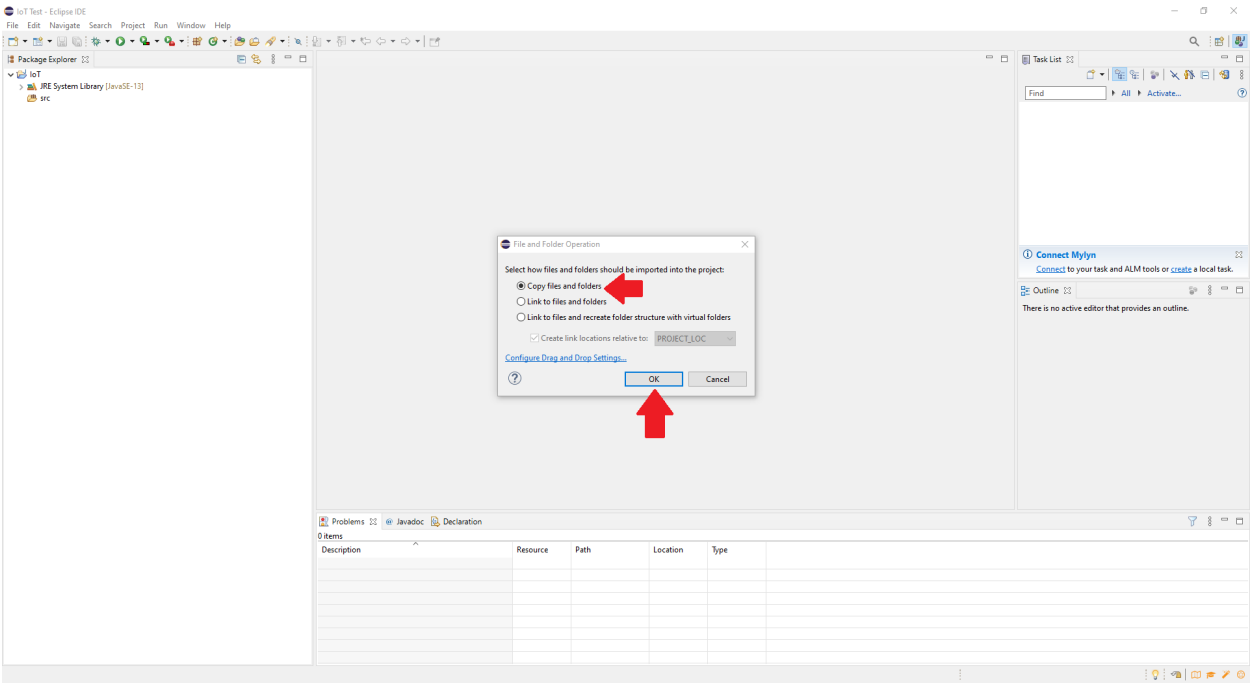




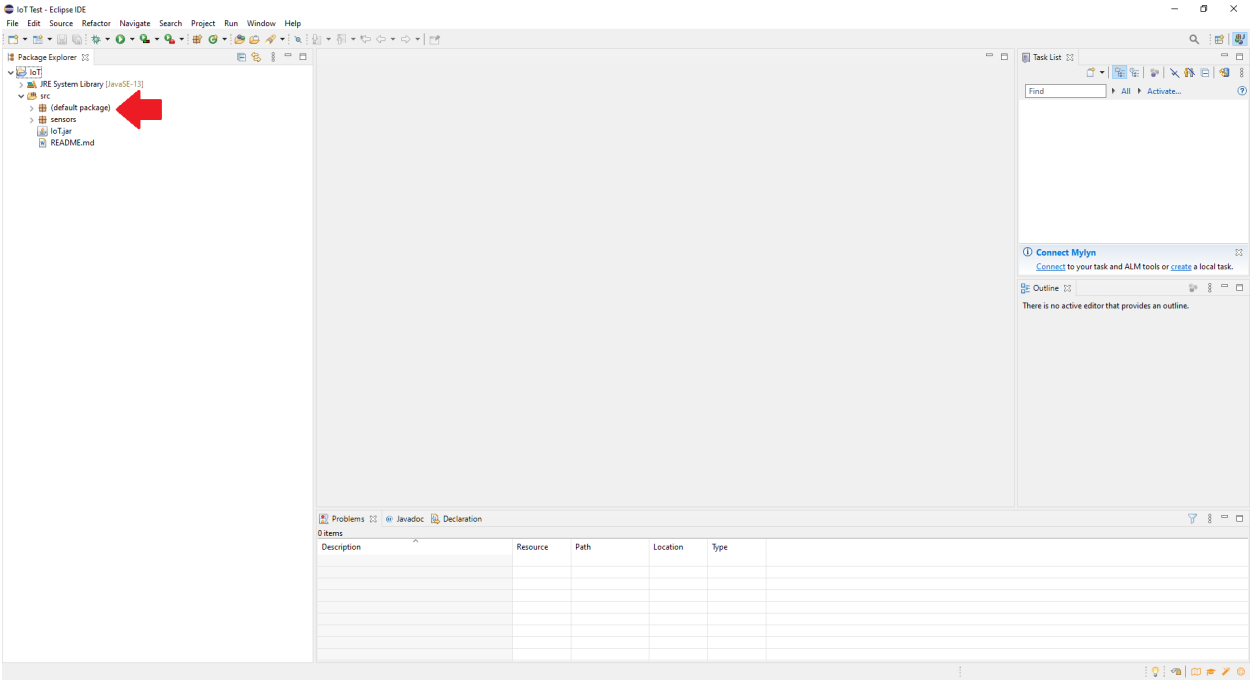
5. Open up the folder containing IoT's software that you have downloaded from the GitLab repository. Select **all** files in that folder and drag it into the folder named **src** in the **Project Explorer** on Eclipse. A new window will pop up

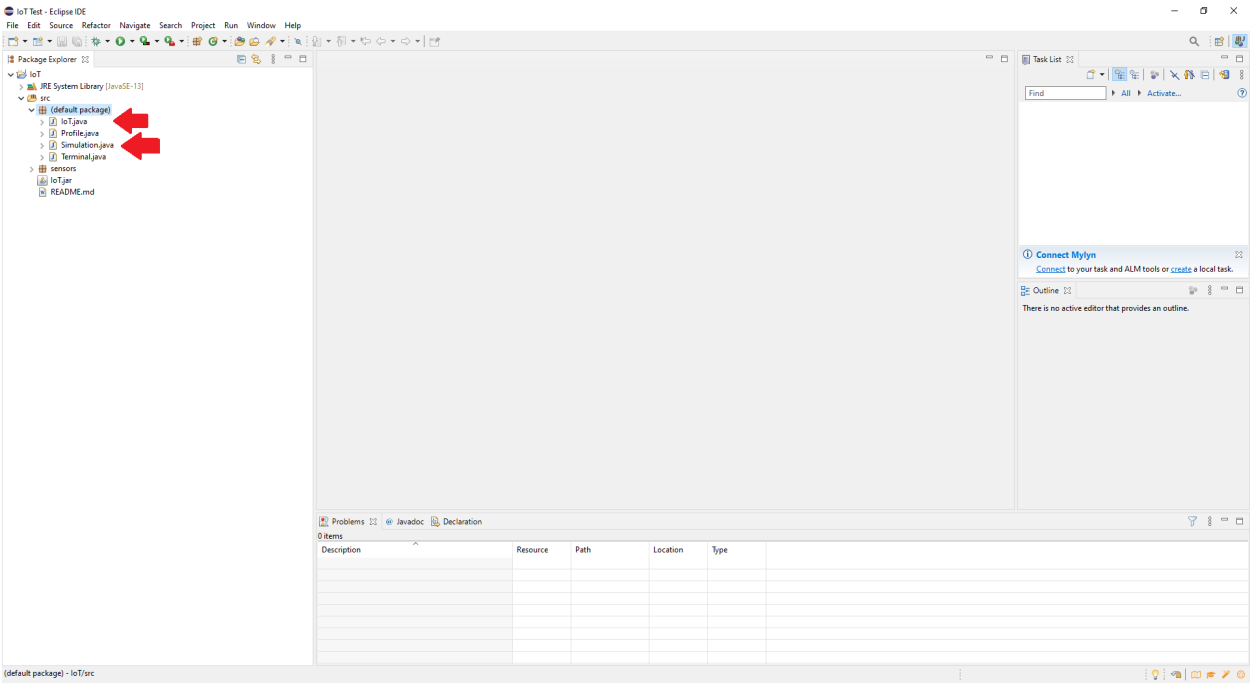
named **File and Folder Operation**. Select the option **Copy files and folders** and press the **Ok** button.



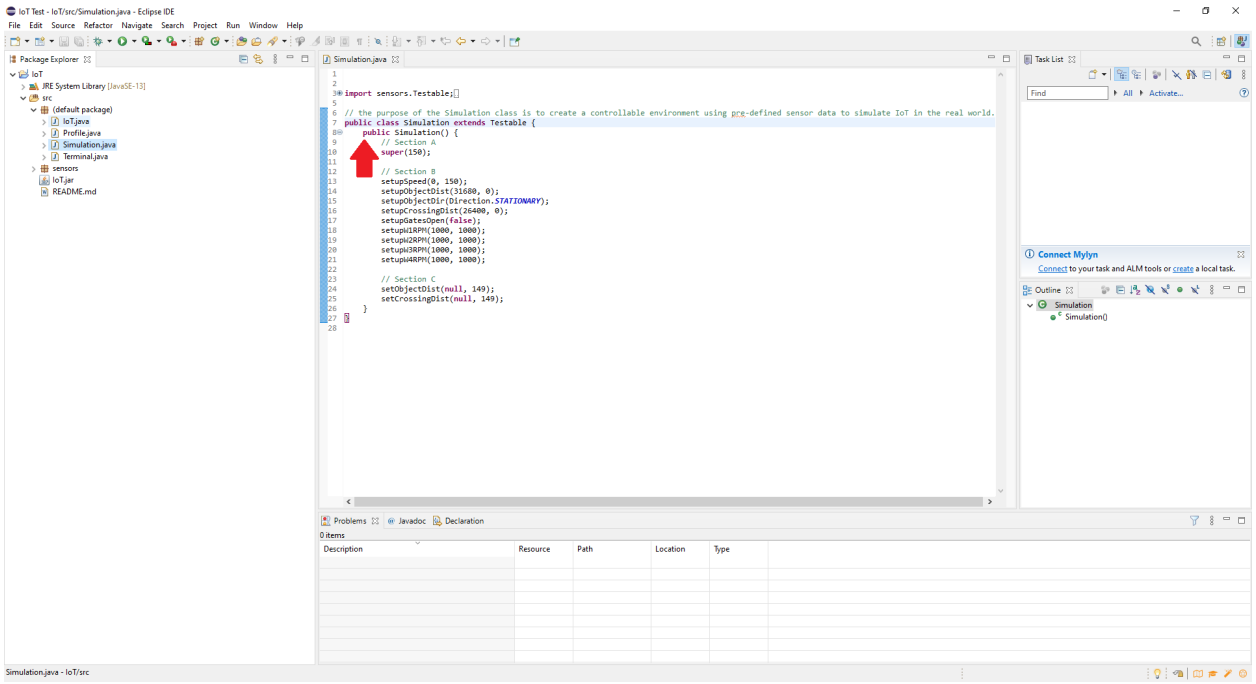


6. After importing IoT's software, double clicking on the **src** folder should display all of IoT's software, including a package named **(default package)**. Double clicking on the **(default package)** package should display some more .java files, including **IoT.java** and **Simulation.java**.





7. Double click on **Simulation.java** to view its source code on the center panel. The **Simulation.java** file serves as the main edible test file to simulate IoT. There, you will find the public initializer class **Simulation()** with comments for a **Section A**, a **Section B**, and a **Section C**. References to pre-defined functions are already inserted in each section to serve as an example simulation in the test file.



2.2 Section A

2.2.1 Purpose

The purpose of Section A is to provide the technician with the power to control how many consecutive TSNR packets the simulation will run on inside the test file. Section A will always contain the **super()** function.

2.2.2 Functions

2.2.2.1 super(int steps):

Variables:

1. **int steps:** specifies the desired number of consecutive TSNR packets the simulation will run on. The number of **steps** must be greater than or equal to 1.

Purpose: This function initializes the simulation by allocating **step** number of consecutive TSNR packets. Each TSNR packet will run in 1 second intervals, with the whole simulation running for a total of **step** seconds. This function must be called for IoT to run.

Example: Using **super(150)** allows the technician to test 150 consecutive TSNR packets for a total simulation time of 150 seconds.

2.3 Section B

2.3.1 Purpose

The purpose of Section B is to provide the technician with the power to automatically populate each sensor output over the course of **step** packets inside the test file. Each sensor output is populated over a given range by simply providing the start and end values to each sensor. The interval in the range increments based on how many **steps** were created using **super()** in **Section A**.

Note: Packet index refers to the current time that has elapsed since the start of the simulation. For example, a packet index of 0 means 0 seconds since the start of the simulation, while a packet index of 100 means 100 seconds since the start of the simulation.

2.3.2 Functions

2.3.2.1 setupSpeed(double start_val, double end_val):

Variables:

1. **double start_val:** specifies the desired starting train speed in mph at packet index 0. The **start_val** must be greater than or equal to 0.

2. **double end_val:** specifies the desired ending train speed in mph at packet index **(steps - 1)**. The **end_val** must be greater than or equal to 0.

Purpose: This function automatically populates the GPS unit's output over the course of **step** packets. The range of values outputted by the GPS unit starts with the train speed at **start_val** mph and ends with the train speed at **end_val** mph. Not calling this function defaults the train speed to 0 mph throughout the simulation.

Example: Using **super(150)** and **setupSpeed(0,150)** allows the technician to start the train speed at 0 mph at the start of the simulation and end the train speed at 150 mph by the end of the simulation. Throughout the simulation, the train speed will increase by 150/149 mph increments.

2.3.2.2 setupObjectDist(double start_val, double end_val):

Variables:

1. **double start_val:** specifies the desired starting distance between the object and the train in feet at packet index 0. The **start_val** must be greater than or equal to 0.

2. **double end_val:** specifies the desired ending distance between the object and the train in feet at packet index **(steps - 1)**. The **end_val** must be greater than or equal to 0.

Purpose: This function automatically populates the first Lidar sensor's output over the course of **step** packets. The range of values outputted by the first Lidar sensor starts with the object being **start_val** feet from the train and ends with the object being **end_val** feet from the train. Not calling this function defaults the distance between the object and the train to be **null** throughout the simulation, i.e. the Lidar sensor does not detect any objects.

Example: Using **super(150)** and **setupObjectDist(31680,0)** allows the technician to start the object at 36,680 feet from the train at the start of the simulation and end the object at 0 feet from the train by the end of the simulation. Throughout the simulation, the distance between the object and the train will decrease by 31680/149 feet increments.

2.3.2.3 setupObjectDir(Direction val):

Variables:

1. **Direction val:** specifies the desired direction that the object is currently heading. Valid directions for **val** are **Direction.AWAY** when the object is moving away from the train, **Direction.STATIONARY** when the object is stationary, **Direction.TOWARDS** when the object is moving towards the train, and **null** when there are no nearby objects.

Purpose: This function automatically populates the proximity sensor's output over the course of **step** packets. The range of values outputted by the proximity sensor is all set to the direction **val**. Not calling this function defaults the object direction to be **null** throughout the simulation, i.e. the proximity sensor does not detect any objects.

Example: Using **super(150)** and **setupObjectDir(DIRECTION.STATIONARY)** allows the technician to set the direction of the object to be stationary throughout the simulation.

2.3.2.4 setupCrossingDist(double start_val, double end_val):

Variables:

1. **double start_val:** specifies the desired starting distance between the Railroad Crossing and the train in feet at packet index 0. The **start_val** must be greater than or equal to 0.

2. **double end_val:** specifies the desired ending distance between the Railroad Crossing and the train in feet at packet index **(steps - 1)**. The **end_val** must be greater than or equal to 0.

Purpose: This function automatically populates the second Lidar sensor's output over the course of **step** packets. The range of values outputted by the second Lidar sensor starts with the Railroad Crossing being **start_val** feet from the train and ends with the Railroad Crossing being **end_val** feet from the train. Not calling this function defaults the distance between the Railroad Crossing and the train to be **null** throughout the simulation, i.e. the Lidar sensor does not detect any Railroad Crossings.

Example: Using **super(150)** and **setupCrossingDist(26400,0)** allows the technician to start the Railroad Crossing at 26,400 feet from the train at the start of the simulation and end the Railroad Crossing at 0 feet from the train by the end of the simulation. Throughout the simulation, the distance between the object and the train will decrease by 26400/149 feet increments.

2.3.2.5 setupGatesOpen(boolean val):

Variables:

1. **boolean val:** specifies the desired crossing gate position. Valid crossing gate positions for **val** are **false** when the crossing gates are in the closed position, and **true** when crossing gates are in the opened position.

Purpose: This function automatically populates the optical sensor's output over the course of **step** packets. The range of values outputted by the optical sensor is all set to the crossing gate

position **val**. Not calling this function defaults the crossing gate position to be closed throughout the simulation.

Example: Using **super(150)** and **setupGatesOpen(false)** allows the technician to set the crossing gate position to be closed throughout the simulation.

2.3.2.6 setupW1RPM(double start_val, double end_val):

Variables:

1. **double start_val:** specifies the desired starting RPM of wheel 1 at packet index 0. The **start_val** must be greater than or equal to 0.
2. **double end_val:** specifies the desired ending RPM of wheel 1 at packet index **(steps - 1)**. The **end_val** must be greater than or equal to 0.

Purpose: This function automatically populates the first RPM sensor's output over the course of **step** packets. The range of values outputted by the first RPM sensor starts with wheel 1 running at **start_val** RPM and ends with wheel 1 running at **end_val** RPM. Not calling this function defaults wheel 1's RPM to be 0 throughout the simulation.

Example: Using **super(150)** and **setupW1RPM(1000,1000)** allows the technician to start wheel 1 at 1,000 RPM at the start of the simulation and end wheel 1 at 1,000 RPM by the end of the simulation. Throughout the simulation, wheel 1's RPM will decrease by 0.

2.3.2.7 setupW2RPM(double start_val, double end_val):

Variables:

1. **double start_val:** specifies the desired starting RPM of wheel 2 at packet index 0. The **start_val** must be greater than or equal to 0.
2. **double end_val:** specifies the desired ending RPM of wheel 2 at packet index **(steps - 1)**. The **end_val** must be greater than or equal to 0.

Purpose: This function automatically populates the second RPM sensor's output over the course of **step** packets. The range of values outputted by the second RPM sensor starts with wheel 2

running at **start_val** RPM and ends with wheel 2 running at **end_val** RPM. Not calling this function defaults wheel 2's RPM to be 0 throughout the simulation.

Example: Using **super(150)** and **setupW2RPM(1000,1000)** allows the technician to start wheel 2 at 1,000 RPM at the start of the simulation and end wheel 2 at 1,000 RPM by the end of the simulation. Throughout the simulation, wheel 2's RPM will decrease by 0.

2.3.2.8 setupW3RPM(double start_val, double end_val):

Variables:

1. **double start_val:** specifies the desired starting RPM of wheel 3 at packet index 0. The **start_val** must be greater than or equal to 0.
2. **double end_val:** specifies the desired ending RPM of wheel 3 at packet index **(steps - 1)**. The **end_val** must be greater than or equal to 0.

Purpose: This function automatically populates the third RPM sensor's output over the course of **step** packets. The range of values outputted by the third RPM sensor starts with wheel 3 running at **start_val** RPM and ends with wheel 3 running at **end_val** RPM. Not calling this function defaults wheel 3's RPM to be 0 throughout the simulation.

Example: Using **super(150)** and **setupW3RPM(1000,1000)** allows the technician to start wheel 3 at 1,000 RPM at the start of the simulation and end wheel 3 at 1,000 RPM by the end of the simulation. Throughout the simulation, wheel 3's RPM will decrease by 0.

2.3.2.9 setupW4RPM(double start_val, double end_val):

Variables:

1. **double start_val:** specifies the desired starting RPM of wheel 4 at packet index 0. The **start_val** must be greater than or equal to 0.
2. **double end_val:** specifies the desired ending RPM of wheel 4 at packet index **(steps - 1)**. The **end_val** must be greater than or equal to 0.

Purpose: This function automatically populates the fourth RPM sensor's output over the course of **step** packets. The range of values outputted by the fourth RPM sensor starts with wheel 4 running at **start_val** RPM and ends with wheel 4 running at **end_val** RPM. Not calling this function defaults wheel 4's RPM to be 0 throughout the simulation.

Example: Using **super(150)** and **setupW4RPM(1000,1000)** allows the technician to start wheel 4 at 1,000 RPM at the start of the simulation and end wheel 4 at 1,000 RPM by the end of the simulation. Throughout the simulation, wheel 4's RPM will decrease by 0.

2.4 Section C

2.4.1 Purpose

The purpose of Section C is to provide the technician with the power to set each sensor output at specific packet indices inside the test file.

Note: Packet index refers to the current time that has elapsed since the start of the simulation. For example, a packet index of 0 means 0 seconds since the start of the simulation, while a packet index of 100 means 100 seconds since the start of the simulation.

2.4.2 Functions

2.4.2.1 setSpeed(double val, int index):

Variables:

1. **double val:** specifies the desired train speed in mph at packet index **index**. The **val** must be greater than or equal to 0.
2. **int index:** specifies the desired packet index to set **val**. The **index** must be greater than or equal to 0 and less than **steps**.

Purpose: This function sets the GPS unit's output at packet index **index** with the train speed at **val** mph.

Example: Using **super(150)** and **setSpeed(150,100)** allows the technician to set the train speed at 150 mph at packet index 100 of the simulation.

2.4.2.2 setObjectDist(Double val, int index):

Variables:

1. **Double val:** specifies the desired distance between the object and the train in feet at packet index **index**. The **val** must be greater than or equal to 0 or **null** when there are no nearby objects.

2. **int index:** specifies the desired packet index to set **val**. The **index** must be greater than or equal to 0 and less than **steps**.

Purpose: This function sets the first Lidar sensor's output at packet index **index** with the object being **val** feet from the train.

Example: Using **super(150)** and **setObjectDist(31680,100)** allows the technician to set the object at 31,680 feet from the train at packet index 100 of the simulation.

2.4.2.3 setObjectDir(Direction val, int index):

Variables:

1. **Direction val:** specifies the desired direction that the object is currently heading at packet index **index**. Valid directions for **val** are **Direction.AWAY** when the object is moving away from the train, **Direction.STATIONARY** when the object is stationary, **Direction.TOWARDS** when the object is moving towards the train, and **null** when there are no nearby objects.

2. **int index:** specifies the desired packet index to set **val**. The **index** must be greater than or equal to 0 and less than **steps**.

Purpose: This function sets the proximity sensor's output at packet index **index** with the direction **val**.

Example: Using **super(150)** and **setObjectDir(DIRECTION.STATIONARY,100)** allows the technician to set the direction of the object to be stationary at packet index 100 of the simulation.

2.4.2.4 setCrossingDist(Double val, int index):

Variables:

1. **Double val:** specifies the desired distance between the Railroad Crossing and the train in feet at packet index **index**. The **val** must be greater than or equal to 0 or **null** when there are no nearby Railroad Crossings.

2. **int index:** specifies the desired packet index to set **val**. The **index** must be greater than or equal to 0 and less than **steps**.

Purpose: This function sets the second Lidar sensor's output at packet index **index** with the Railroad Crossing being **val** feet from the train.

Example: Using **super(150)** and **setCrossingDist(26400,100)** allows the technician to set the Railroad Crossing at 26,400 feet from the train at packet index 100 of the simulation.

2.4.2.5 setGatesOpen(boolean val, int index):

Variables:

1. **boolean val:** specifies the desired crossing gate position at packet index **index**. Valid crossing gate positions for **val** are **false** when the crossing gates are in the closed position, and **true** when crossing gates are in the opened position.

2. **int index:** specifies the desired packet index to set **val**. The **index** must be greater than or equal to 0 and less than **steps**.

Purpose: This function sets the optical sensor's output at packet index **index** with the crossing gate position **val**.

Example: Using **super(150)** and **setGatesOpen(false,100)** allows the technician to set the crossing gate position to be closed at packet index 100 of the simulation.

2.4.2.6 setW1RPM(double val, int index):

Variables:

1. **double val:** specifies the desired RPM of wheel 1 at packet index **index**. The **val** must be greater than or equal to 0.

2. **int index:** specifies the desired packet index to set **val**. The **index** must be greater than or equal to 0 and less than **steps**.

Purpose: This function sets the first RPM sensor's output at packet index **index** with wheel 1 running at **val** RPM.

Example: Using **super(150)** and **setW1RPM(1000,100)** allows the technician to set wheel 1 at 1,000 RPM at packet index 100 of the simulation.

2.4.2.7 setW2RPM(double val, int index):

Variables:

1. **double val:** specifies the desired RPM of wheel 2 at packet index **index**. The **val** must be greater than or equal to 0.

2. **int index:** specifies the desired packet index to set **val**. The **index** must be greater than or equal to 0 and less than **steps**.

Purpose: This function sets the second RPM sensor's output at packet index **index** with wheel 2 running at **val** RPM.

Example: Using **super(150)** and **setW2RPM(1000,100)** allows the technician to set wheel 2 at 1,000 RPM at packet index 100 of the simulation.

2.4.2.8 setW3RPM(double val, int index):

Variables:

1. **double val:** specifies the desired RPM of wheel 3 at packet index **index**. The **val** must be greater than or equal to 0.

2. **int index:** specifies the desired packet index to set **val**. The **index** must be greater than or equal to 0 and less than **steps**.

Purpose: This function sets the third RPM sensor's output at packet index **index** with wheel 3 running at **val** RPM.

Example: Using **super(150)** and **setW3RPM(1000,100)** allows the technician to set wheel 3 at 1,000 RPM at packet index 100 of the simulation.

2.4.2.9 setupW4RPM(double val, int index):

Variables:

1. **double val:** specifies the desired RPM of wheel 4 at packet index **index**. The **val** must be greater than or equal to 0.
2. **int index:** specifies the desired packet index to set **val**. The **index** must be greater than or equal to 0 and less than **steps**.

Purpose: This function sets the fourth RPM sensor's output at packet index **index** with wheel 4 running at **val** RPM.

Example: Using **super(150)** and **setW4RPM(1000,100)** allows the technician to set wheel 4 at 1,000 RPM at packet index 100 of the simulation.

---

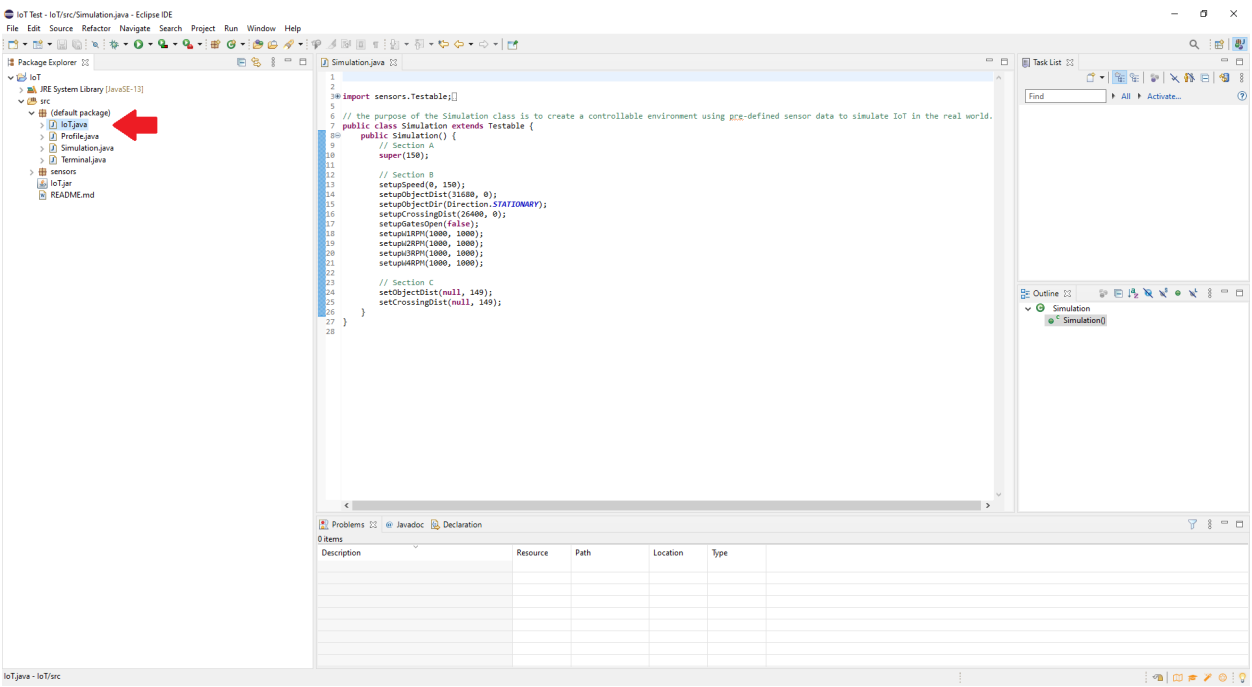
3) TEST FILE EXECUTION

3.1 Compilation

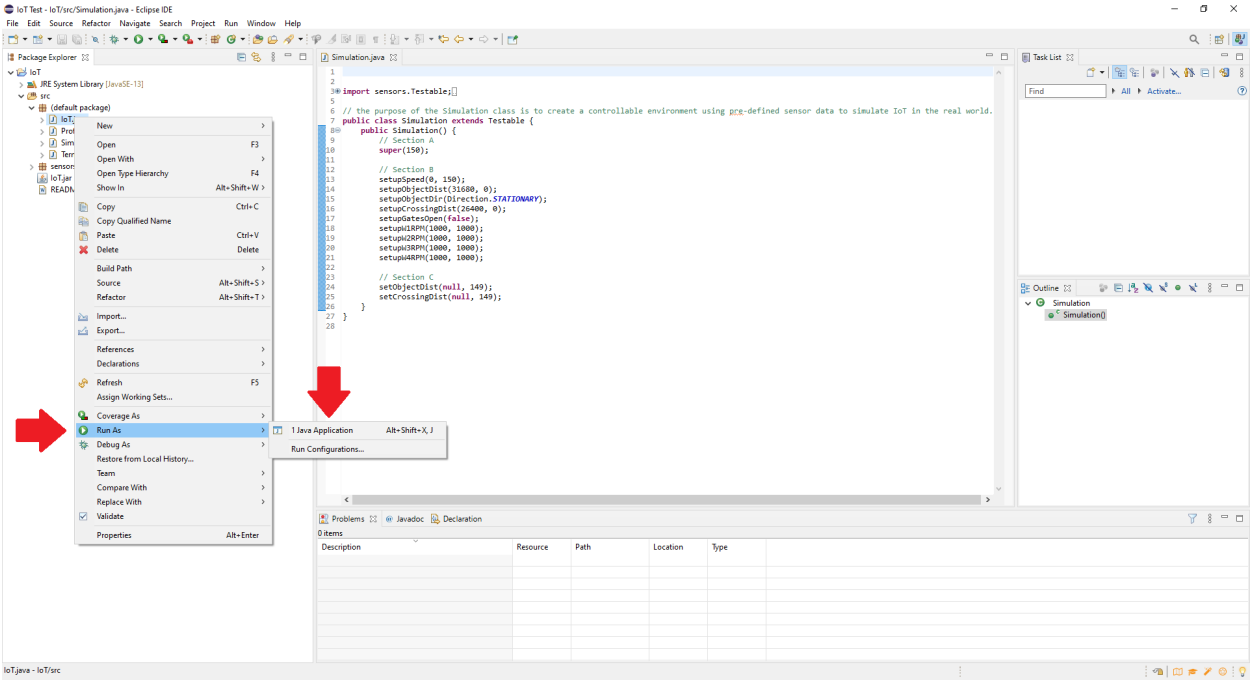
In this section, we will be compiling the IoT software along with the test file in the Eclipse IDE.

1. Make sure to save all changes in **Simulation.java** and that there are no runtime errors in the file.

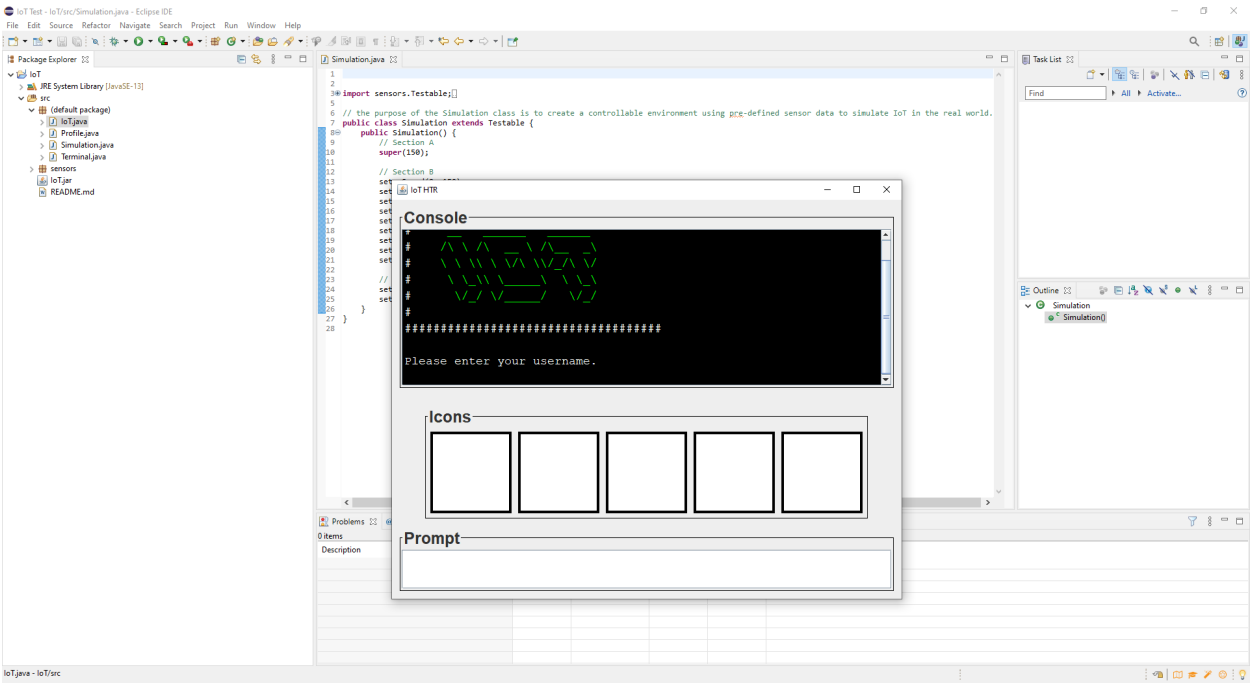
2. Look at the **Project Explorer** sidebar and find **IoT.java** under the package named **(default package)**.



3. Right click on **IoT.java** to display all the file options. In the file options, hovering over the **Run as** option should display all run configurations.Click on the option for **1 Java Application**.



4. If the IoT software compiles correctly, then a new executable window named **IoT HTR** will show up on your computer.



3.2 Debugging

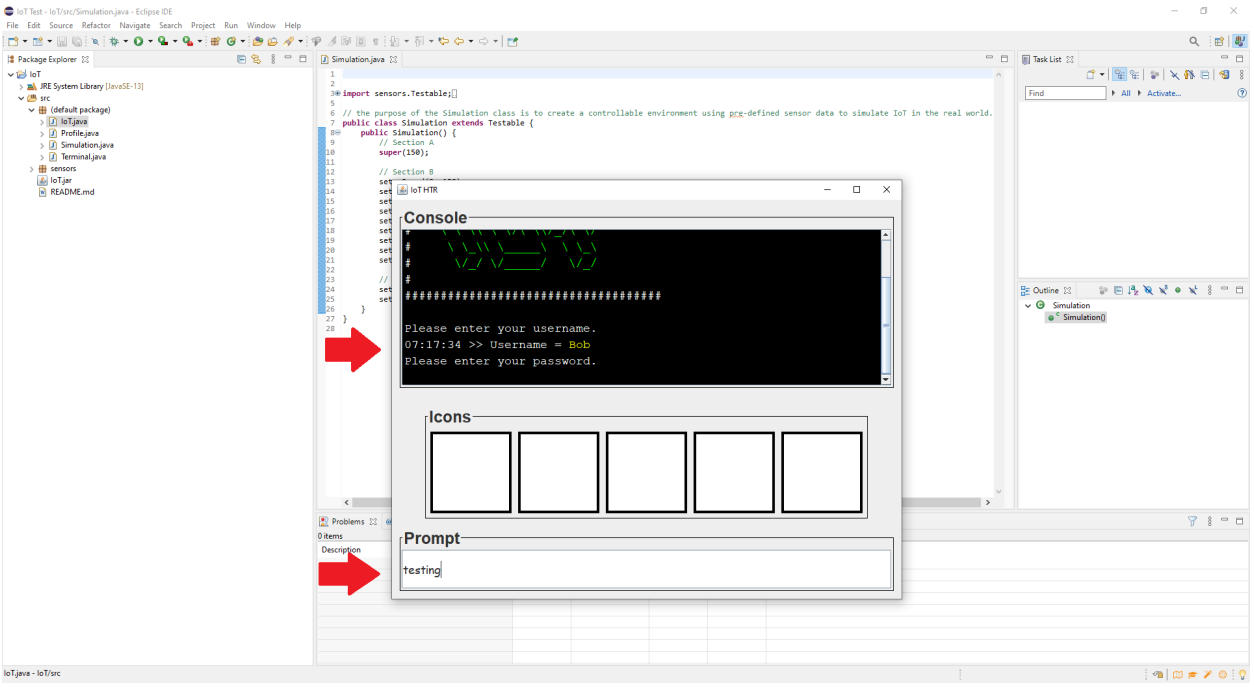
In this section, we will be outlining the basic features of the **IoT HTR** executable. For reference, we used the test file **Simulation.java** supplied in the GitLab repo.

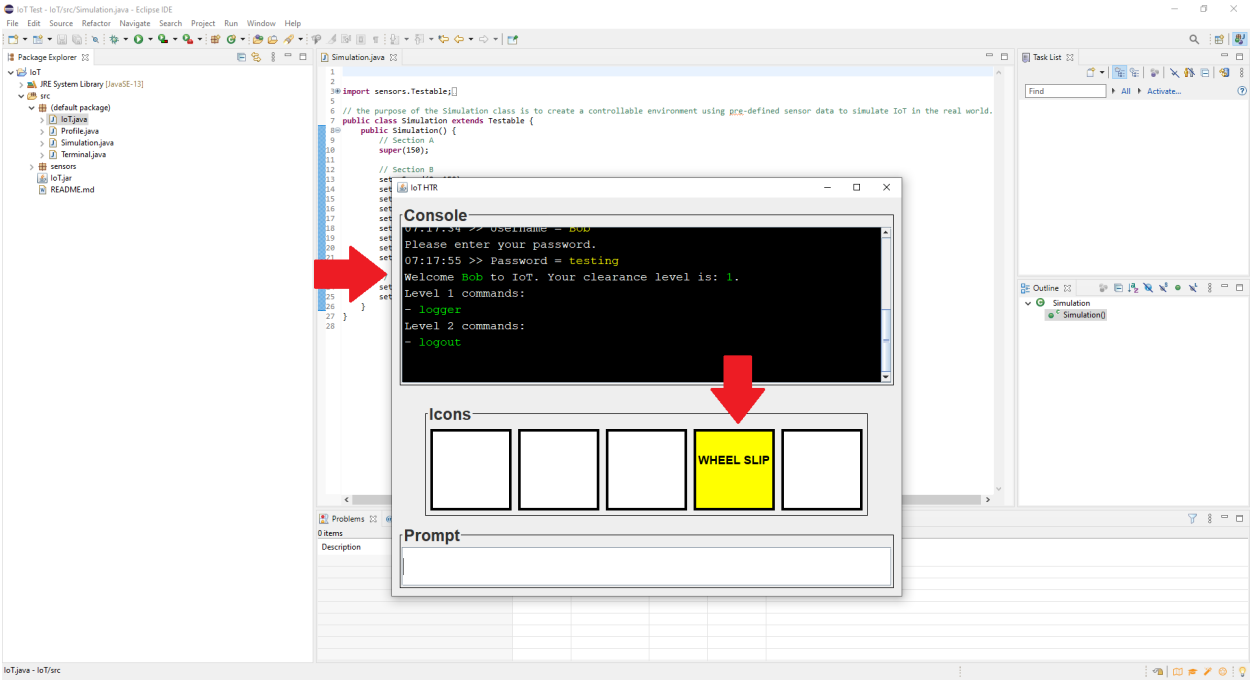
1. There are 3 panels on the executable: a read-only **Console** panel, a real-time **Icons** panel, and a text-entry **Prompt** panel.

2. Upon launching the executable, the **Console** panel should prompt you to enter your username and password. Choose between the following two valid user accounts (case sensitive):

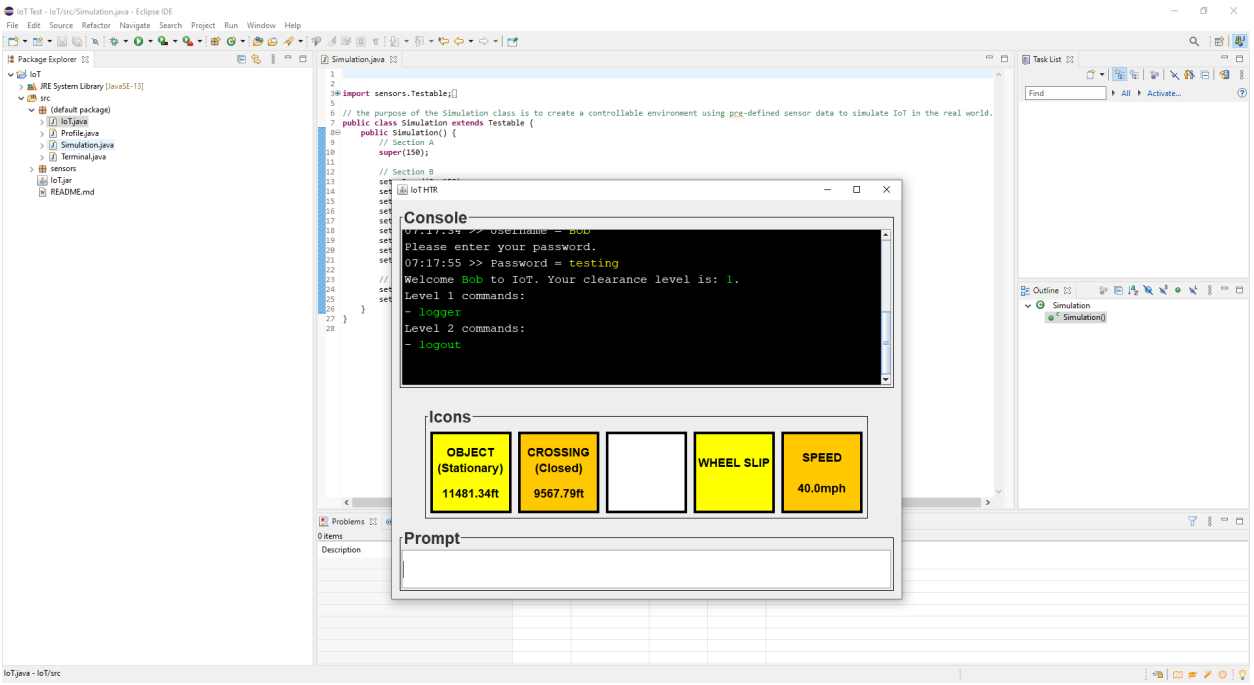
Train Engineer		**Train Operator**	
Username:	Bob	**Username:**	John
Password:	testing	**Password:**	12345
Clearance:	Level 1	**Clearance:**	Level 2

3. In this case, we chose the Train Engineer. Enter the username **Bob** into the **Prompt** panel followed by the password **testing**. After successfully logging on, your login credentials and clearance level will be displayed onto the **Console** panel followed by an immediate yellow **WHEEL SLIP** warning on the **Icons** panel.

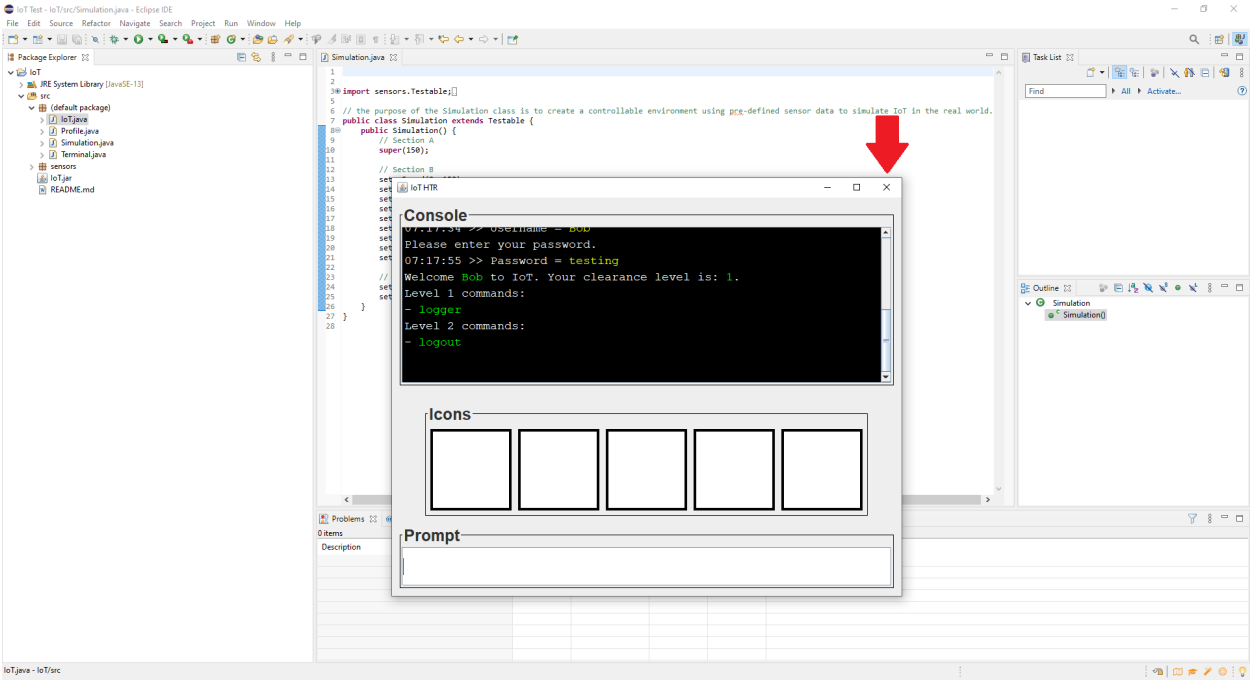




4. The simulation that you have created in the test file will now play out in real-time, with the appropriate icons displaying on the **Icons** panel.



5. To exit out of the executable, simply press the **Close** button.



3.3 Commands

In this section, we will be outlining the basic commands of the **IoT HTR** executable. These commands will only execute after a successful login.

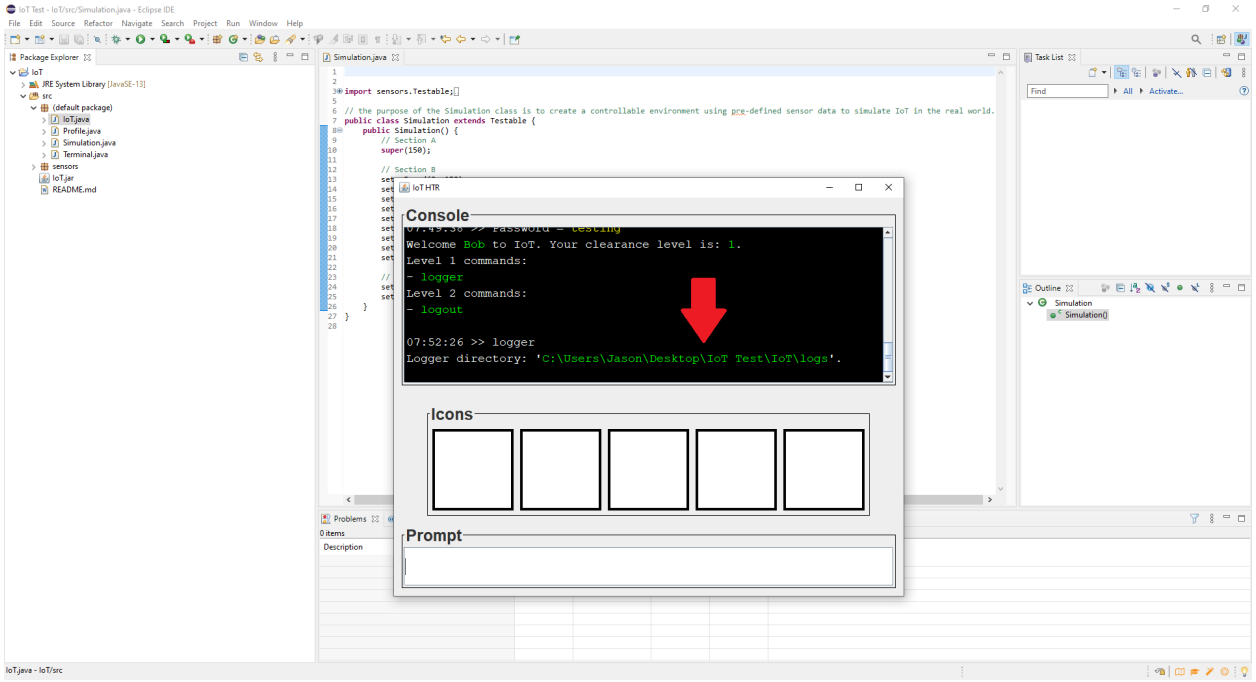
3.3.1 Logout

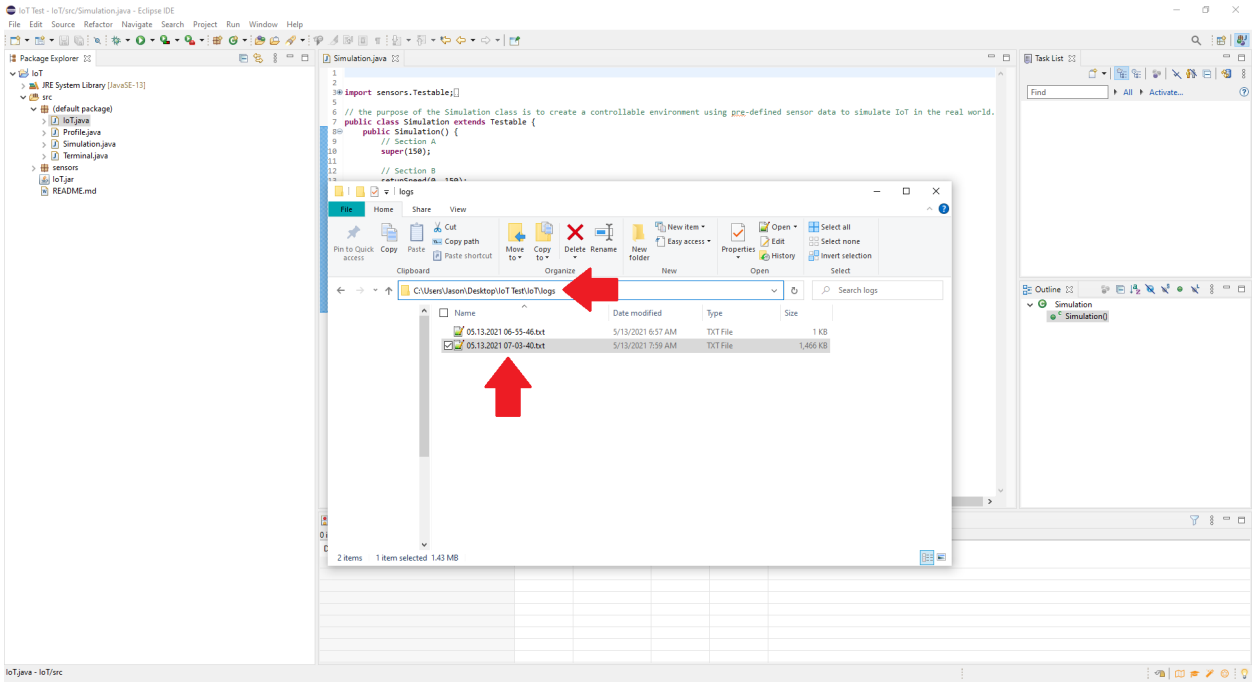
This command requires **Level 1** or **Level 2** clearance. To initiate a logout, enter the command **logout** in the **Prompt** panel. The current user account should then be logged out with a login reprompt on the **Console** panel. Additionally, the simulation will reset, allowing for another rerun of the simulation.

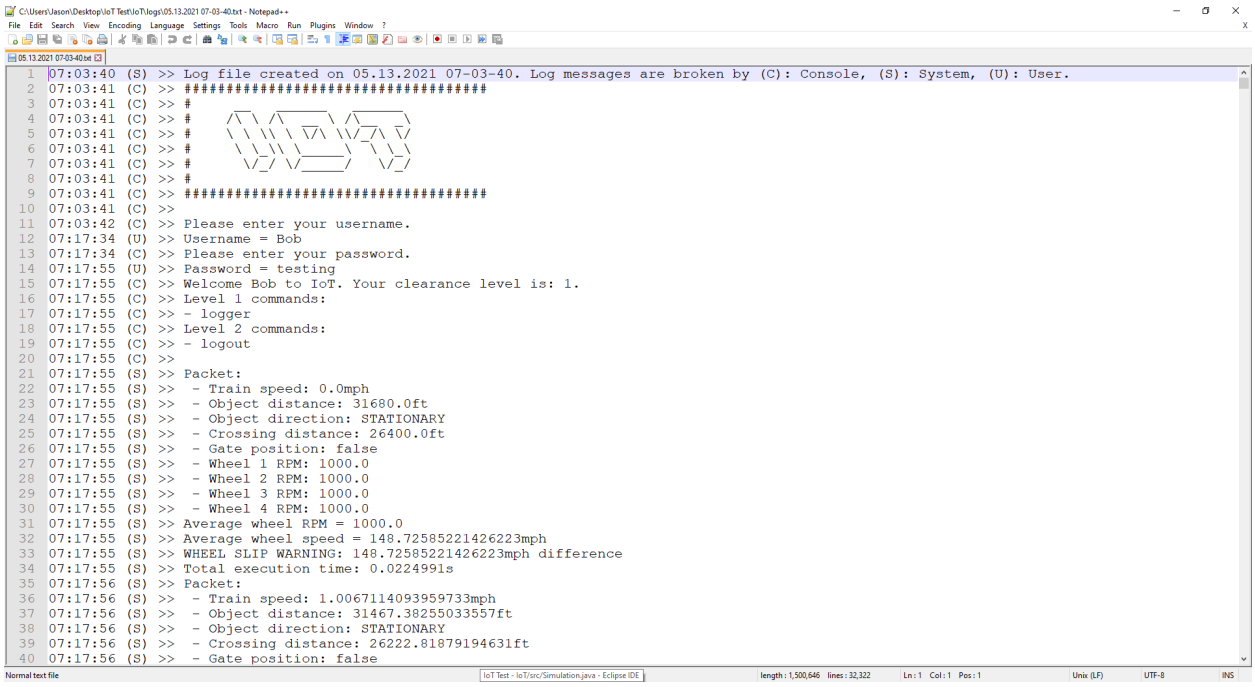
3.3.2 Logger

This command requires **Level 1** clearance. To request the log file directory, enter the command **logger** in the **Prompt** panel. An absolute file path to the directory containing all log files should be printed onto the **Console** panel.

Note: If the executable is still open and you try to access the current log file, the log file will appear to be empty. However, after exiting the executable, all log data will be saved to the current log file.


IoT HTR
Console
Welcome Bob to IoT. Your clearance level is: 1.
Level 1 commands:
- logger
Level 2 commands:
- logout
07:52:26 >> logger
Logger directory: 'C:\Users\Jason\Desktop\IoT Test\IoT\logs'.
Icons
Prompt


logs
C:\Users\Jason\Desktop\IoT Test\IoT\logs
05.13.2021 06-55-46.txt
05.13.2021 07-03-40.txt
2 items
1 item selected 1.43 MB


C:\Users\Jason\Desktop\IoT Test\IoT\logs\05.13.2021 07-03-40.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
05.13.2021 07-03-40.txt
07:03:40 (S) >> Log file created on 05.13.2021 07-03-40. Log messages are broken by (C): Console, (S): System, (U): User.
07:03:41 (C) >> ##
07:03:41 (C) >> #
07:03:41 (C) >> #
07:03:41 (C) >> #
07:03:41 (C) >> #
07:03:41 (C) >> #
07:03:41 (C) >> #
07:03:41 (C) >> ##
07:03:41 (C) >>
07:03:42 (C) >> Please enter your username.
07:17:34 (U) >> Username = Bob
07:17:34 (C) >> Please enter your password.
07:17:55 (U) >> Password = testing
07:17:55 (C) >> Welcome Bob to IoT. Your clearance level is: 1.
07:17:55 (C) >> Level 1 commands:
07:17:55 (C) >> - logger
07:17:55 (C) >> Level 2 commands:
07:17:55 (C) >> - logout
07:17:55 (C) >>
07:17:55 (S) >> Packet:
07:17:55 (S) >> - Train speed: 0.0mph
07:17:55 (S) >> - Object distance: 31680.0ft
07:17:55 (S) >> - Object direction: STATIONARY
07:17:55 (S) >> - Crossing distance: 26400.0ft
07:17:55 (S) >> - Gate position: false
07:17:55 (S) >> - Wheel 1 RPM: 1000.0
07:17:55 (S) >> - Wheel 2 RPM: 1000.0
07:17:55 (S) >> - Wheel 3 RPM: 1000.0
07:17:55 (S) >> - Wheel 4 RPM: 1000.0
07:17:55 (S) >> Average wheel RPM = 1000.0
07:17:55 (S) >> Average wheel speed = 148.72585221426223mph
07:17:55 (S) >> WHEEL SLIP WARNING: 148.72585221426223mph difference
07:17:55 (S) >> Total execution time: 0.0224991s
07:17:56 (S) >> Packet:
07:17:56 (S) >> - Train speed: 1.0067114093959733mph
07:17:56 (S) >> - Object distance: 31467.38255033557ft
07:17:56 (S) >> - Object direction: STATIONARY
07:17:56 (S) >> - Crossing distance: 26222.81879194631ft
07:17:56 (S) >> - Gate position: false
Normal text file
IoT Test - IoT/src/Simulation.java - Eclipse IDE
length : 1,500,646 lines : 32,322
Ln : 1 Col : 1 Pos : 1
Unix (LF)
UTF-8
INS