Comp 524 - Assignment 0: Hello to Java and Eclipse Tools

Date Assigned: Tue August 24, 2021

Early Completion Date: Friday, August 27, 2020 (+5% extra credit)

Completion Date: Tuesday, August 31, 2020

The goal of this assignment is to familiarize you with some of the required and optional tools you need during this course.

The recommended optional tools are Eclipse, LocalChecks, and UNC Checkstyle (also called Hermes), for editing programs in the four languages, checking runtime behavior of programs, and checking source structure of programs, respectively.

Installing any kind of tool can create problems that cannot be solved by intuition or logic. So the moment you hit a brick wall, seek help and hopefully the instructors can solve the problem. The corollary of this is that follow the instructions carefully so you reduce the chance of such problems.

It is because of such problems the tools are divided into required and optional – you can do without the optional tools (implemented by us) if they create problems you do not or cannot solve. The optional tools log your interactions so we can improve them and our teaching. Being uncomfortable with such logging is another reason to not install them. In the past all students (including 301 students this summer) have managed to install all of these tools, but unanticipated problems can always occur with any tool, especially one built in a university.

The due date seems very conservative at this time and accounts for the fact that tools installation can be troublesome. Expect the next assignment to be given before the deadline for this assignment, with a deadline close to it.

Class Library

In the shared google folder:

Downloads/Comp524All.jar (will be updated throughout the semester.)

Reference Material (Instructor-Created)

Installing JDK, Eclipse, Project Libraries: PowerPoint PDF

In the shared google folder:

InstallationDocs/INSTALLING CLISP.pptx

InstallationDocs /installing sml and prolog.pptx

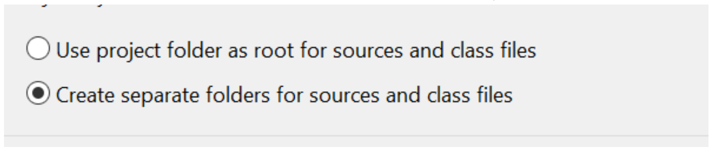
Task 0: Java/Eclipse Installation

1. Install JDK-11 (Described in Installing JDK, Eclipse, Project Libraries: PowerPoint PDF) on your computer. All assignments will use this JDK. Here is a link to it: https://www.oracle.com/java/technologies/javase-jdk11-downloads.html

2. (Optional but recommended): Install Eclipse (Described in Installing JDK, Eclipse, Project Libraries: PowerPoint PDF). ProDT (to be installed later) requires the following: IDE for Eclipse Committers, Version: Photon Release (4.8.0), Build id: 20180619-1200,Here is a link to Photon: https://www.eclipse.org/downloads/packages/release/photon/r

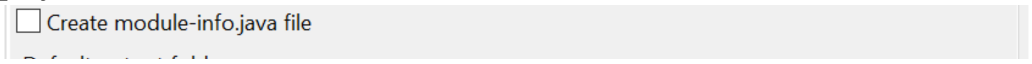
Task 1: Java Project and Class Library

Java Project Structure: Create a Java Eclipse project structure (a project directory with an **src** folder for source files and **bin** directory for class files.).



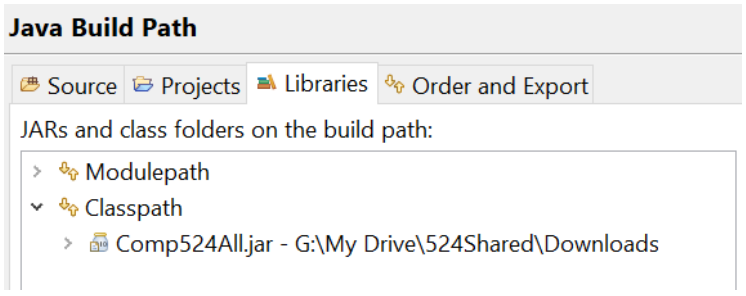
○ Use project folder as root for sources and class files
[✓ ● Create separate folders for sources and class files]

When creating a project, uncheck the box that asks for creation of the module-info file. If your project has been created with such a file, delete it.



☐ Create module-info.java file

Class library: Add the library Comp524All.jar (Available In the shared google folder: Downloads/Comp524All.jar). Instructions on how to add libraries to Eclipse projects are in (Installing JDK, Eclipse, Project Libraries: PowerPoint PDF). Add it to the classpath and not to the modulepath.



Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

```
> Modulepath
v Classpath
    > Comp524All.jar - G:\My Drive\524Shared\Downloads
```

Task 2: Class Registry

Create a single class (in an appropriate package) in the Java project that implements the following interface defined in the shared library:

gradingTools.shared.testcases.greeting.GreetingClassRegistry

For now, return the **null** value in the getter required by the interface. Eclipse will automatically generate such a method if you ask it to fix the error of unimplemented interface methods.

Our grader will look in your project for a class implementing this interface. **If such a class does not exist or more than one such class implements the interface, it will fail, and no other aspect of your assignment will be graded.** So it is important to get this simple requirement right.

Task 3: Localchecks Runner (optional)

Create a class with the following code (the name of the class does not matter). The commented lined are for later assignments, they have been provided as a template.

```
import grader.basics.execution.BasicProjectExecution;
import gradingTools.comp524f21.assignment0.F21Assignment0_1Suite;
import trace.grader.basics.GraderBasicsTraceUtility;
import util.trace.Tracer;
public class RunF21A0_1Tests {
	public static void main(String[] args) {
		// if you set this to false, grader steps will not be traced
		GraderBasicsTraceUtility.setTracerShowInfo(true);
		// if you set this to false, all grader steps will be traced,
		// not just the ones that failed
		GraderBasicsTraceUtility.setBufferTracedMessages(true);
		// Change this number if a test trace gets longer than 600 and is
clipped
		GraderBasicsTraceUtility.setMaxPrintedTraces(600);
		// Change this number if all traces together are longer than 2000
		GraderBasicsTraceUtility.setMaxTraces(2000);
		// Change this number if your process times out prematurely
		BasicProjectExecution.setProcessTimeOut(5);
		// You need to always call such a method
		F21Assignment0_1Suite.main(args);
	}
}
```

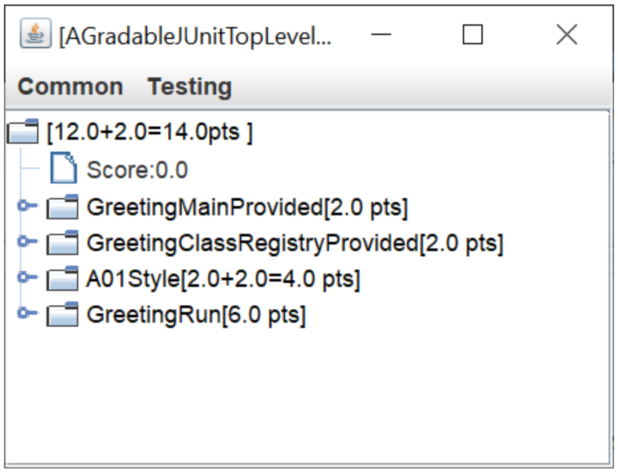
The main method of this class executes the main method of a test suite class we provide. We will call such a class **a LocalChecks runner**. You will create such a class for each assignment. Here you are running the main method in the following test suite:

gradingTools.comp524f21.assignment0_1.F20Assignment0_1Suite

In general, the test suite for Assignment <Number> in semester <Semester> will be in package

gradingtools.Comp524<Semester>.Assignment<Number>Suite.

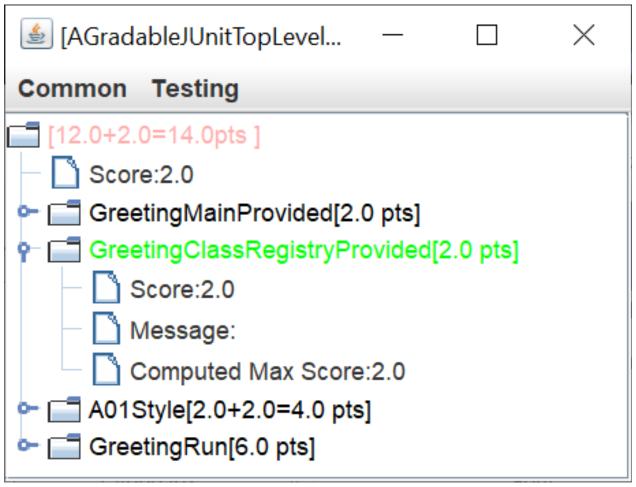
At this point you should see the following user-interface:



[12.0+2.0=14.0pts]
Score:0.0
GreetingMainProvided[2.0 pts]
GreetingClassRegistryProvided[2.0 pts]
A01Style[2.0+2.0=4.0 pts]
GreetingRun[6.0 pts]

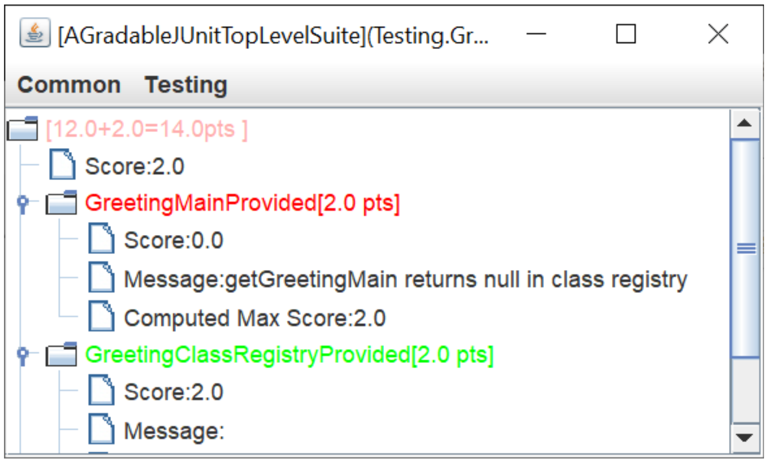
As you can see, each displayed check has a name that tries to explain its function. You can hover over the name of a check to see what it does. You can double click on any test to check it individually.

This is what should happen if you double click on **GreetingClassRegistryProvided**.



[12.0+2.0=14.0pts]
Score:2.0
GreetingMainProvided[2.0 pts]
GreetingClassRegistryProvided[2.0 pts]
Score:2.0
Message:
Computed Max Score:2.0
A01Style[2.0+2.0=4.0 pts]
GreetingRun[6.0 pts]

This is what happens if you double click on GreetingMainProvided:



```
[12.0+2.0=14.0pts ]
    Score:2.0
    GreetingMainProvided[2.0 pts]
        Score:0.0
        Message:getGreetingMain returns null in class registry
        Computed Max Score:2.0
    GreetingClassRegistryProvided[2.0 pts]
        Score:2.0
        Message:
```

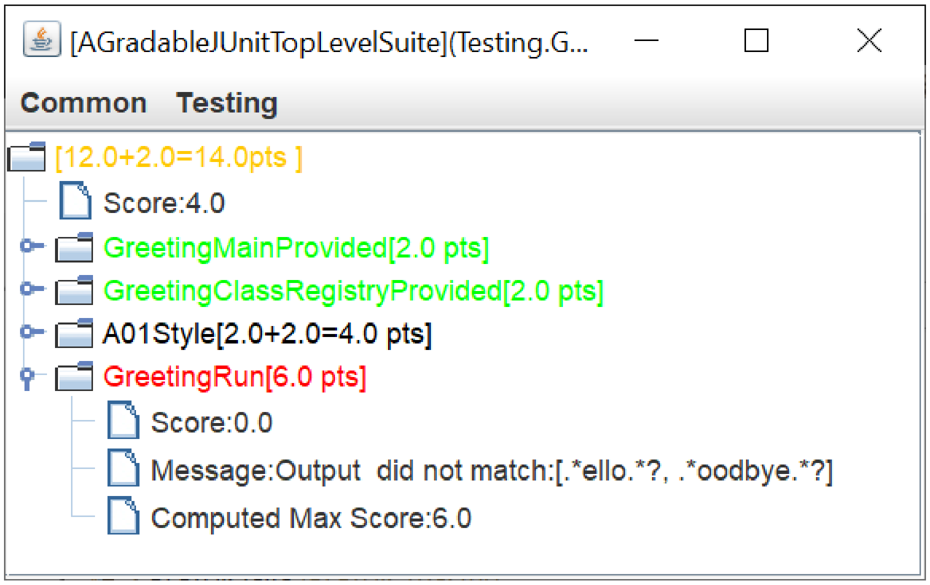
As you see here, a test turns green if it finds no problems, and a shade of red if it does find problems.

The error message is clear enough and is caused by returning a null main class in the registry, which in turn is a result of having a main class. Running tests before you have written code to satisfy them is called **test-first** or **test-driven** programming.

Create another class, and for now, give it the name, **cls**, which has been deliberately chosen to be a non-mnemonic name that does not follow Java conventions. Eclipse will complain; ignore this message for now. Do not put any method in it. Return its **class object** as the return value of the getter in the greeting class registry you defined above. The class object for a class named **C** is **C.class**. Thus, your class registry getter should now be:

```
public Class<?> getGreetingMain() {
        return cls.class;
}
```

Run your LocalChecks runner, and double click on GreetingRun, to get the following result:


[AGradableJUnitTopLevelSuite](Testing.G...

Common Testing

```
[12.0+2.0=14.0pts ]
    Score:4.0
    GreetingMainProvided[2.0 pts]
    GreetingClassRegistryProvided[2.0 pts]
    A01Style[2.0+2.0=4.0 pts]
    GreetingRun[6.0 pts]
        Score:0.0
        Message:Output  did not match:[.*ello.*?, .*oodbye.*?]
        Computed Max Score:6.0
```

This test depends on previous two tests, so they are executed automatically and as before turn green. The error message however is not as explicit because it describes what the final problem is, not the intermediate ones. To get an idea of the intermediate ones, you need to look at the console output, h

```
Error: Main method not found in class greeting.cls, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
GreetingRun did not pass completely:Output  did not match:[.*ello.*?,
.*oodbye.*?]%0.0
Steps traced since last test:
I***{AWT-EventQueue-0}(BasicProjectIntrospection) Looking for unique class of
existing type interface gradingTools.shared.testcases.greeting.GreetingClassRegistry
I***{AWT-EventQueue-0}(BasicProjectIntrospection) Found type:class
greeting.AGreetingRegistry
:.\bin
….
```

You see two error messages, the first one indicating the real problem, and the second one a consequence of the first one. The final one is the one that was displayed in the UI as the test failure.

After the test failure message you see the following line:

```
Steps traced since last test:
```

This line is followed by a sequence of informational messages starting with the prefix I***, which indicate the steps the failed test took before the failure. For instance, the test GreetingRun first ran GreetingClassRegistry, on which it depends, which in turn looked for a class that implements the interface, GreetingClassRegistry.

```
I***{AWT-EventQueue-0}(BasicProjectIntrospection) Looking for unique class of
existing type interface gradingTools.shared.testcases.greeting.GreetingClassRegistry
```

Recall that your LocalTest runner has the line:

```
GraderBasicsTraceUtility.setBufferTracedMessages(true);
```

It is for this reason that messages associated with a test are suppressed and displayed only if the test fails. Set this value to false, and run each of the tests again, Now you see a message for each step displayed as it is taken, and the error messages are displayed after the display of the steps that lead to them:

```
I***{AWT-EventQueue-0}(BasicProcessRunner) Delaying input feed for ms:500
Error: Main method not found in class greeting.Hello, please define the main method
as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
I***{Out Stream Runnable}(BasicRunningProject) 1629636408764: Received output from
main: +++++++%$#@*___
I***{Output Sorter}(BasicRunningProject) main ended output
```

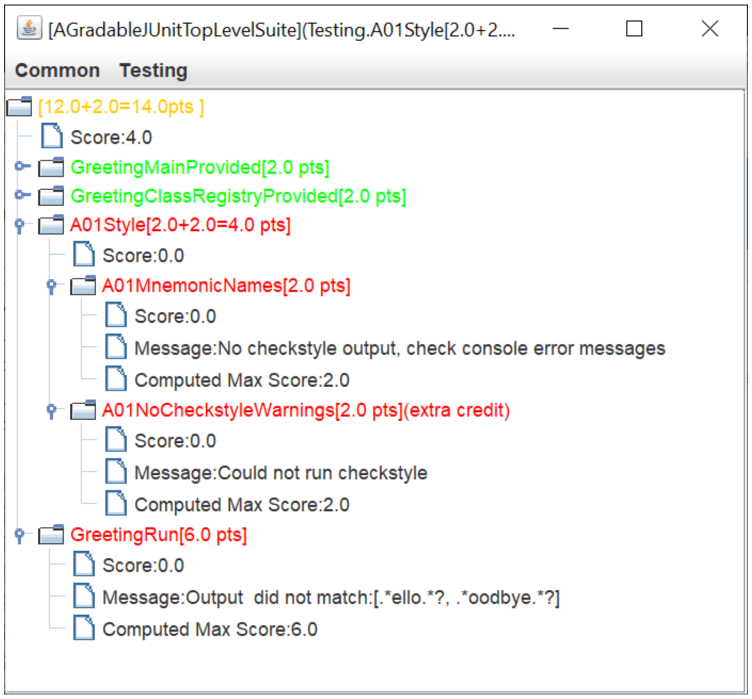
Unless you are interested in how successful tests work, you should buffer the messages. Otherwise your display will be cluttered. So set the buffering to true again.
Trace messages are shown because of the line:

```
GraderBasicsTraceUtility.setTracerShowInfo(true);
```

in your runner. If you set it to false, no trace messages will be shown. If you get an error you do not understand, you should turn it on. Unless, the buffered messages are bothering you, keep these on,

Our next job is to fix the problem of not having a main method. Ideally, such a violation of source-code requirements should have been given while editing code which requires a source code checker. Such a checker should also, ideally, catch violations of required code style such as not following Java class conventions and creation of non-mnemonic names. We wills leave these source code violations in our code to see how such a tool may work in the next section.

Before we do so, double click on A01 Style. You should get the following display:


```
[AGradableJUnitTopLevelSuite](Testing.A01Style[2.0+2....
Common   Testing
[12.0+2.0=14.0pts ]
    Score:4.0
    GreetingMainProvided[2.0 pts]
    GreetingClassRegistryProvided[2.0 pts]
    A01Style[2.0+2.0=4.0 pts]
        Score:0.0
        A01MnemonicNames[2.0 pts]
            Score:0.0
            Message:No checkstyle output, check console error messages
            Computed Max Score:2.0
        A01NoCheckstyleWarnings[2.0 pts](extra credit)
            Score:0.0
            Message:Could not run checkstyle
            Computed Max Score:2.0
    GreetingRun[6.0 pts]
        Score:0.0
        Message:Output  did not match:[.*ello.*?, .*oodbye.*?]
        Computed Max Score:6.0
```

A01Style is not an atomic test, but a suite, consisting of two atomic tests, A01MnemonicNames and A01NoCheckstyleWarnings. Double clicking on a suite runs all tests in it, which are usually related in some way. Both of these tests are red, and hence so is the Suite.

This time, one of the error messages explicitly asks us to look at the console output, which is more explicit:

```
Running junit test:A01MnemonicNames at Sun Aug 22 21:01:03 EDT 2021
Please run the checkstyle plugin on your project
Could not initialize checkstyle. Please make sure you have installed the plugin and
run checkstyle on the project
A01MnemonicNames did not pass completely:No checkstyle output, check console error
messages%0.0
```

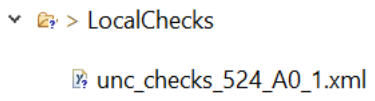
As you might now, a plugin is an addition to an existing program. This plugin has been created for the Eclipse programming environment. Adding a plugin to a programming environment is not as simple as adding a library jar, so this is going to be a much more cumbersome and error-prone optional step.

Task 3: UNC Checkstyle (optional)

Perhaps the most famous checker of Java source code is the extensible Checkstyle plugin extension to Eclipse. UNC Checkstyle is an extension of it. Checkstyle supports only application-independent checks such as following of Java class conventions. UNC Checkstyle extends the application-independent checks and also supports application-dependent checks through class **tags** as we will see below

The requirements of a Java file to be checked are defined in a checkstyle file associated with the Eclipse Java project in which the file is located. Each assignment will have a separate checkstyle requirement file available here
In the shared google folder:
F21/CheckStyleFiles (will be constantly updated)

The name of the file indicates the associated assignment. For example, the checkstyle file for this assignment is:
unc_checks_524_A0_1.xml
Download the file to your computer and move it to the Logs/LocalChecks directory in your project, which was created by running localchecks the first time. You may have to refresh the project folder (select project, right click, refresh) to make Logs appear


> LocalChecks
unc_checks_524_A0_1.xml

We will call it the **assignment check configuration**.

The next step is to extend your Eclipse with the plugin. Slides on how to install it, the data it mines and logs, and how to connect a project with an assignment configuration file are accessible from:

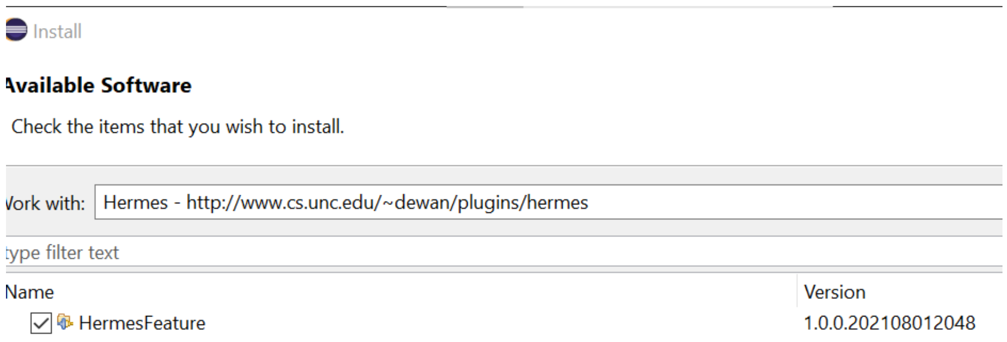
Checkstyle with UNC Checks : Install and Use	PowerPoint PDF

These instructions, in fact, take you to the following instructions for installing the difficulty plugin:

Difficulty Plugin	PowerPoint PDF

To reduce the installation effort, we have combined all relevant plugins into a single plugin, called the Hermes plugin which forms both the Checkstyle plugin and the Difficulty plugin. The plugin performs source checks, tries to detect programming difficulties, and also provides help within the Eclipse environment. We will focus here on the source checks aspect of it.

Study the instructions in the difficulty plugin slides on how to install and uninstall Hermes. Here is a figure of the main step you need to take:


Install

Available Software

Check the items that you wish to install.

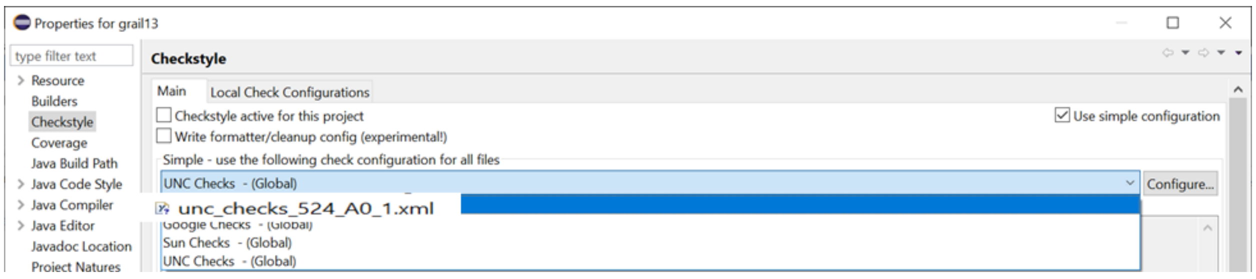
Work with: Hermes - http://www.cs.unc.edu/~dewan/plugins/hermes

type filter text

Name	Version
☑ HermesFeature	1.0.0.202108012048

Install the plugin and then follow the instructions in the Checkstyle slides on how to connect a project to a configuration file.

Connect the Java project you just created to the configuration file you downloaded.
The last step is shown in the figure below:


Properties for grail13

Checkstyle

Main | Local Check Configurations

☐ Checkstyle active for this project
☐ Write formatter/cleanup config (experimental!)
☑ Use simple configuration

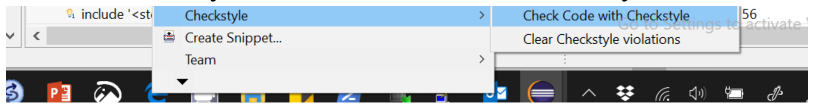
Simple - use the following check configuration for all files

UNC Checks - (Global)

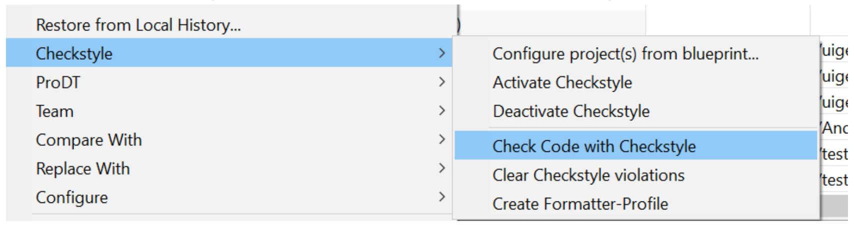
unc_checks_524_A0_1.xml

Google Checks - (Global)
Sun Checks - (Global)
UNC Checks - (Global)

If all has gone well with your installation, you should be able to right click on the file **cls.java** and run Checkstyle-->Check Code with Checkstyle.

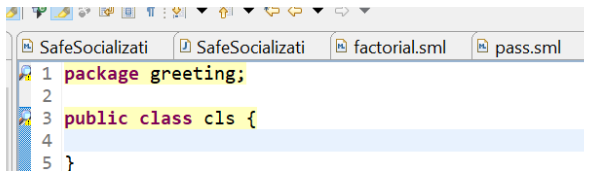


When you have more than one file to check in a project, you should right click on the project, and run Checkstyle   Check Code with Checkstyle.



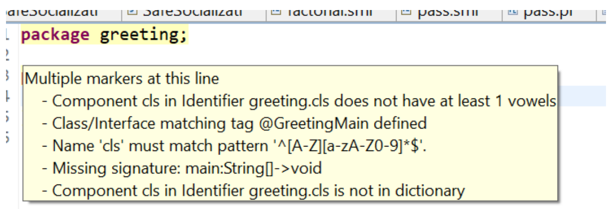
Restore from Local History...
Checkstyle
ProDT
Team
Compare With
Replace With
Configure
Configure project(s) from blueprint...
Activate Checkstyle
Deactivate Checkstyle
Check Code with Checkstyle
Clear Checkstyle violations
Create Formatter-Profile

You should see the source code annotated with some yellow highlights and magnifying glass marks in the margins.



```
package greeting;

public class cls {

}
```

You can hover over the magnifying glass marks to see the associated messages:



```
package greeting;
```

Multiple markers at this line
- Component cls in Identifier greeting.cls does not have at least 1 vowels
- Class/Interface matching tag @GreetingMain defined
- Name 'cls' must match pattern '^[A-Z][a-zA-Z0-9]*$'.
- Missing signature: main:String[]->void
- Component cls in Identifier greeting.cls is not in dictionary

You see that Checkstyle is complaining about the name of your class and also the fact that the main method is missing.

How did it know that a main method was required in this class? This information is in the configuration file:

```
<module name="ExpectedSignatures">
      <property name="severity" value="warning" />
      <property name="includeTypeTags" value="@GreetingMain" />
      <property name="expectedSignatures"
            value="
                  main:String[]->void,
            " />
</module>

<module name="ExpectedSignatures">
      <property name="severity" value="warning" />
      <property name="includeTypeTags" value="@GreetingMain" />
      <property name="expectedSignatures"
            value="
                  main:String[]->void,
            " />
</module>
```

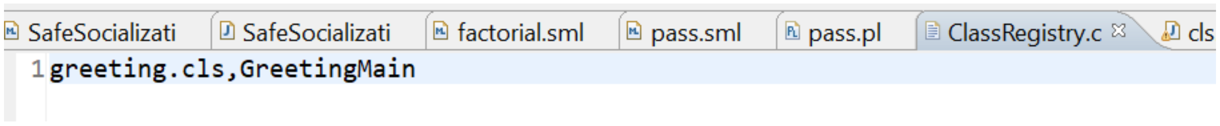
This is FYI, you do not have to understand this syntax. But it is important to know that somehow UNC Checkstyle needs to know the purpose of each file it is checking to provide file-specific checks.

This are two UNC Checkstyle checks called **ExpectedSignatures.** The first says that a warning should be given if a class with the tag **@GreetingMain** does not have a method with the following signature:

```
	main:String[]->void
```

The second says an informational message should be given if such a class does have the signature above.

How did the plugin know that class **cls** has the tag **GreetingMain?** It is integrated with localchecks. When you ran the local check **GreetingClassRegistryProvided**, it generated a file class **ClassRegistry.csv** in your project whose contents have map each class name to the associated tag:



```
greeting.cls,GreetingMain
```

This file is read by the plugin to map your classes to the tags.

This means if you change class names, you should rerun the **class registry check** - the local check whose name contains the string **ClassRegistry**.

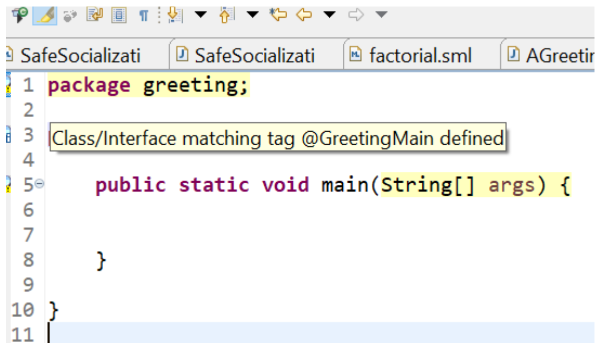
Rename **cls** to follow the Java convention of starting Class names with upper case letters. Also make its mnemonic by calling it, say, Hello, **(**Right click on the class name in the left explorer window and enter ALT-SHIFT-R). Run the class registry check. The class registry file should now be updated:

```
greeting.Hello,GreetingMain
```

Run Checkstyle again on the file. Now you see both purple and yellow marks:

```
package greeting;

public class Hello {

	public static void main(String[] args) {


	}

}
```

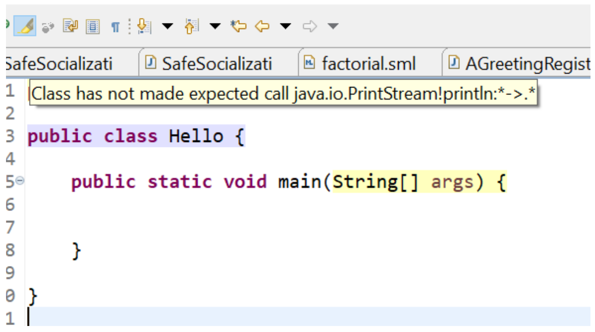
The purple marks are information messages, they are not problems. Hovering over the one shown above tells us that Checkstyle has mapped this class to the tag **@GreetingMain**. This is good, we have fulfilled at least the need for a main class.



SafeSocializati | SafeSocializati | factorial.sml | AGreetir

```
package greeting;

Class/Interface matching tag @GreetingMain defined

    public static void main(String[] args) {

    }

}
```

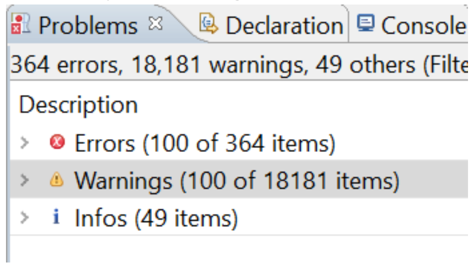
We no longer get an error saying that the main method is missing. But we do get the message about a missing println() in this method:



SafeSocializati | SafeSocializati | factorial.sml | AGreetingRegist

```
Class has not made expected call java.io.PrintStream!println:*->.*

public class Hello {

    public static void main(String[] args) {

    }

}
```

Checkstyle messages can also be seen in the Problems tab.



364 errors, 18,181 warnings, 49 others (Filte

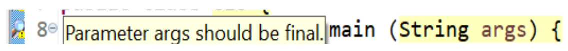
Description
- Errors (100 of 364 items)
- Warnings (100 of 18181 items)
- Infos (49 items)

This tab reports problems reported both by the Java compiler and also (UNC) Checkstyle. Add the following two prints the **Hello** class created in the optional steps above.

```
package greeting;

public class Hello {
	public static void main(String[] args) {
		System.out.println("Hello World");
		System.out.println("Goodbye World");
	}

}
```

The remaining purple highlights tell us that this class has correctly been tagged, In general informational messages are good!

Not all messages are provided by UNC Checks. Here is one provided by the un-extended Checkstyle

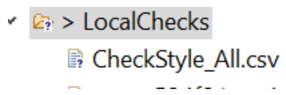


```
8⊖ Parameter args should be final. main (String args) {
```

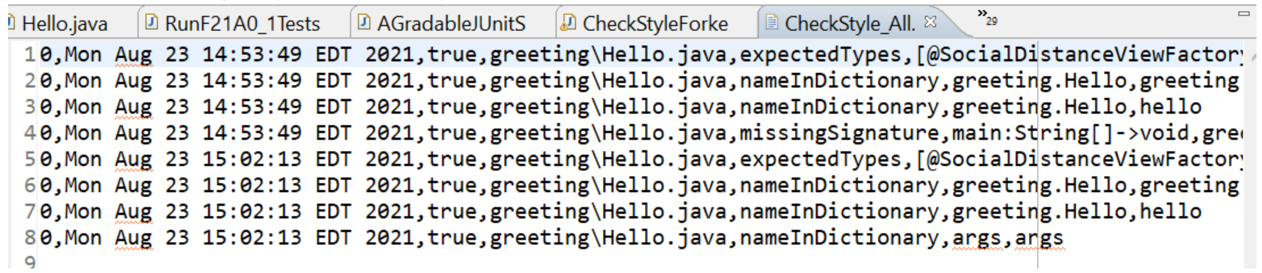
We do not change the value of parameter, args, in the main method, so this message makes sense. But for now let us ignore this message and see if we can get away with it in terms of grades. To do so, we need to go back to using LocalChecks. We can now run the LocalChecks tests that depend on Checkstyle being installed.

Task 4: Scoring Checkstyle Results (Optional)

Refresh LocalChecks. You should see the following file.



This file has a log of your checkstyle runs and their results:

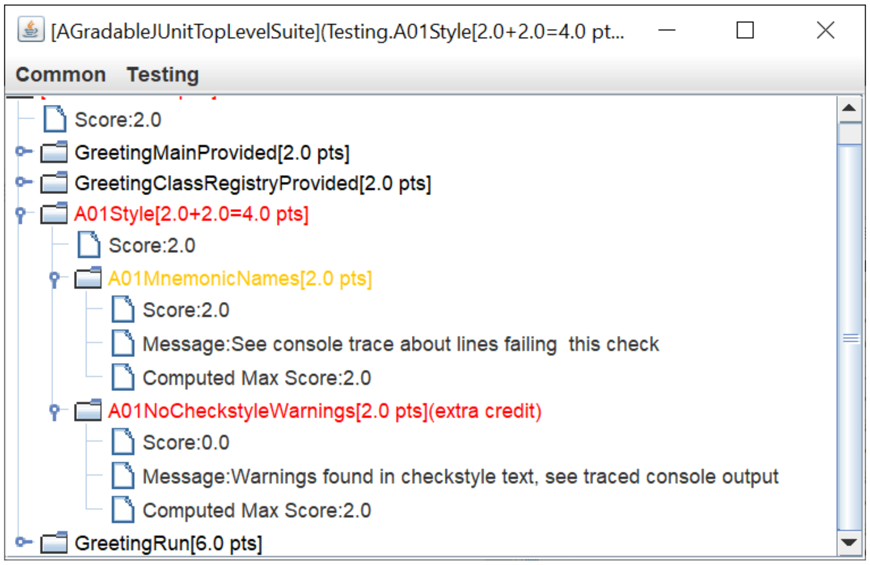


```
0,Mon Aug 23 14:53:49 EDT 2021,true,greeting\Hello.java,expectedTypes,[@SocialDistanceViewFactor
0,Mon Aug 23 14:53:49 EDT 2021,true,greeting\Hello.java,nameInDictionary,greeting.Hello,greeting
0,Mon Aug 23 14:53:49 EDT 2021,true,greeting\Hello.java,nameInDictionary,greeting.Hello,hello
0,Mon Aug 23 14:53:49 EDT 2021,true,greeting\Hello.java,missingSignature,main:String[]->void,gre
0,Mon Aug 23 15:02:13 EDT 2021,true,greeting\Hello.java,expectedTypes,[@SocialDistanceViewFactor
0,Mon Aug 23 15:02:13 EDT 2021,true,greeting\Hello.java,nameInDictionary,greeting.Hello,greeting
0,Mon Aug 23 15:02:13 EDT 2021,true,greeting\Hello.java,nameInDictionary,greeting.Hello,hello
0,Mon Aug 23 15:02:13 EDT 2021,true,greeting\Hello.java,nameInDictionary,args,args
```

For instance, this file shows that in the run at 14:53 I had a missing signature message but in the run at 3:02 I did not.

The contents of the file are of concern only to the curious. The existence of the file is an indication that LocalChecks can now use Checkstyle to allocate points based on source code constraints.

So let me double click on A01Style again. Here is what I see on my computer:


[AGradableJUnitTopLevelSuite](Testing.A01Style[2.0+2.0=4.0 pt...
Common
Testing

```
Score:2.0
GreetingMainProvided[2.0 pts]
GreetingClassRegistryProvided[2.0 pts]
A01Style[2.0+2.0=4.0 pts]
    Score:2.0
    A01MnemonicNames[2.0 pts]
        Score:2.0
        Message:See console trace about lines failing  this check
        Computed Max Score:2.0
    A01NoCheckstyleWarnings[2.0 pts](extra credit)
        Score:0.0
        Message:Warnings found in checkstyle text, see traced console output
        Computed Max Score:2.0
GreetingRun[6.0 pts]
```

A01Mneomonics have me full points but is yellow to indicate it is not entirely happy. The reason is that it checks the whole project source, not just the Hello.java file - it does not know what set of files in the source code are relevant to the submission, My A01 solution is part of a a single project in which I also have solutions to other problems and other code. It found the following problematic name:

```
A01MnemonicNames did not pass completely:See console trace about lines failing  this
check%0.9984423676012462
Steps traced since last test:
I***{AWT-EventQueue-0}(CheckStyleWarningsRatioTestCase) 1 lines failing check
I***{AWT-EventQueue-0}(CheckStyleWarningsRatioTestCase) [WARN]
D:\dewan_backup\Java\PLProjs\PLProjsJava\.\src\style_bad\PLProjsPostProcessingCustomM
ain.java:8: Component projs in Identifier style_bad.PLProjsPostProcessingCustomMain
is not in dictionary [MnemonicName]
I***{AWT-EventQueue-0}(TestCaseResult) ### anonymous: 0.9984423676012462
```

Yet it gave me full points because 99.8 percent of my names had components in dictionary.

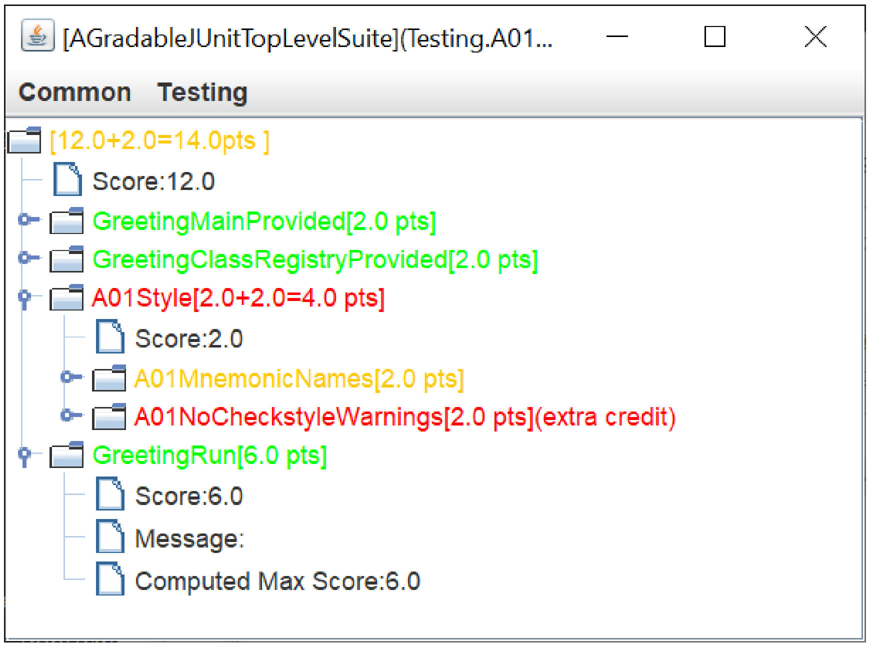
I was not so lucky with the other test case, A01CheckStyleNoWarnings. As its name suggests, it is looking for no warnings from Checkstyle, and has no understanding of what the warnings are. The console output has all the warnings:

```
A01NoCheckstyleWarnings did not pass completely:Warnings found in checkstyle text,
see traced console output%0.0
Steps traced since last test:
Warnings found in following checkstyle text
I***{AWT-EventQueue-0}(CheckStyleWarningsRatioTestCase) [WARN]
D:\dewan_backup\Java\PLProjs\PLProjsJava\.\src\greeting\Hello.java:5:33: Parameter
args should be final. [FinalParameters]
```

Did we get away with fighting Checkstyle. Yes and No. Yes, because this test is extra credit. No, because we lost the extra credit. With all extra credit features, we let you make the decision regarding whether they are worth implementing.

Task 5: Completing LocalChecks Tests (Optional)

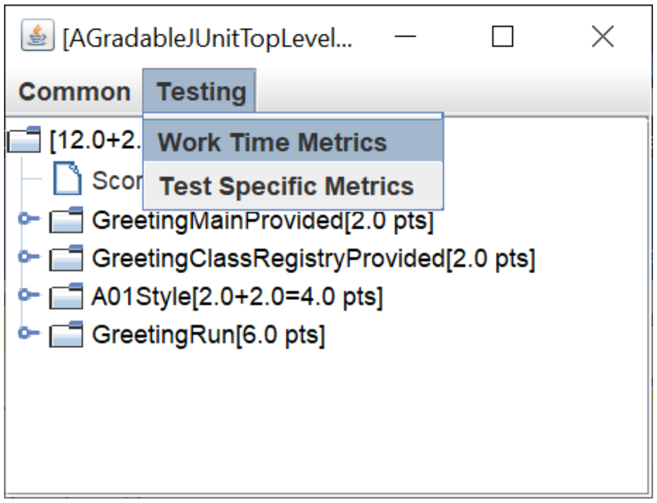
Now that we have completed the Hello class, we can run the GreetingRun test also to complete the assignment.



```
[12.0+2.0=14.0pts ]
  Score:12.0
  GreetingMainProvided[2.0 pts]
  GreetingClassRegistryProvided[2.0 pts]
  A01Style[2.0+2.0=4.0 pts]
    Score:2.0
    A01MnemonicNames[2.0 pts]
    A01NoCheckstyleWarnings[2.0 pts](extra credit)
  GreetingRun[6.0 pts]
    Score:6.0
    Message:
    Computed Max Score:6.0
```

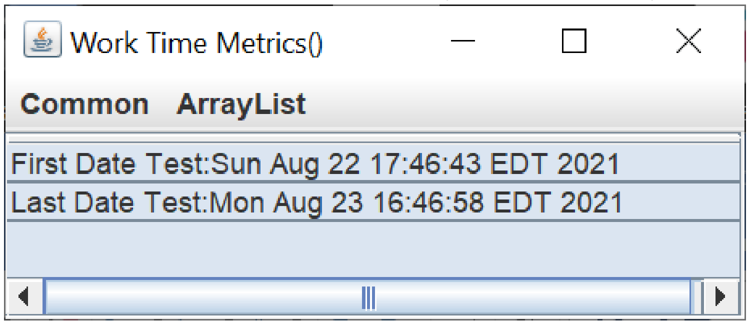
Task 6: Statistics (Optional)

If you were using the plugin, you probably received notifications from it telling you how much time you spent in Eclipse each hour. You can also get statistics from the LocalChecks user-interface. If you execute the Testing   Work Time command:



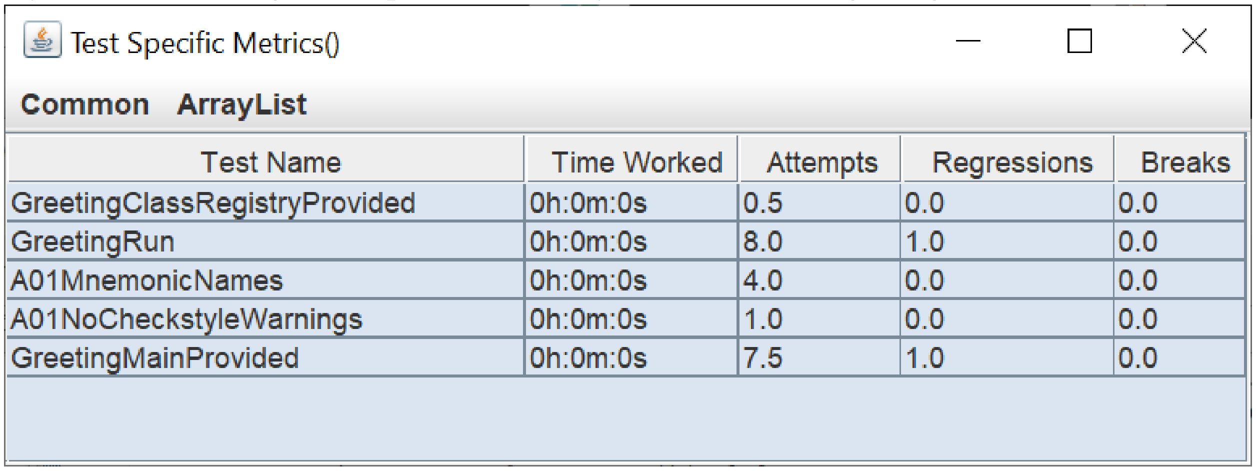
GreetingMainProvided[2.0 pts]
GreetingClassRegistryProvided[2.0 pts]
A01Style[2.0+2.0=4.0 pts]
GreetingRun[6.0 pts]

You will see the first and last time you executes any test:



First Date Test:Sun Aug 22 17:46:43 EDT 2021
Last Date Test:Mon Aug 23 16:46:58 EDT 2021

If you execute Testing   Test Specific Metrics, you will see data regarding each test:



Test Name	Time Worked	Attempts	Regressions	Breaks
GreetingClassRegistryProvided	0h:0m:0s	0.5	0.0	0.0
GreetingRun	0h:0m:0s	8.0	1.0	0.0
A01MnemonicNames	0h:0m:0s	4.0	0.0	0.0
A01NoCheckstyleWarnings	0h:0m:0s	1.0	0.0	0.0
GreetingMainProvided	0h:0m:0s	7.5	1.0	0.0

A test is attempted if it is executed and its current score is different from the score returned by the last time it was executed, A test regresses if its score reduces. If N tests are attempted in a test run, which can happen if for instance you run a suite or run a dependent test, then it is number of attempts increased by 1/N. Time worked and breaks is work in progress right now as LocalChecks logs are not yet combined with plugin logs.

We certainly would benefit from knowing these statistics aggregated for the class. In case you too are interested (which can happen if you think you went abnormally fast or slow), then youc an use these commands.

This is all work in progress and these commands will probbaly get more sophisticated as the semester progresses. If you use them, please let us know how they can be improved.

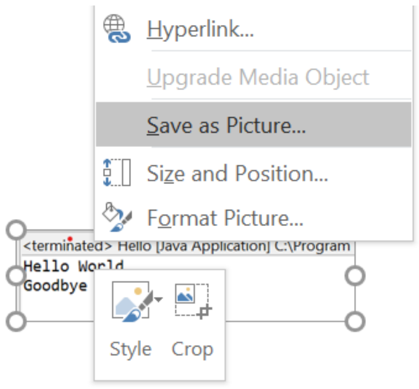
Task 7: Main Method Requirements

If you did not any of the optional tasks above.

Create a main class that prints the strings containing the text “ello” and “oodbye” in two different lines. Return the class object for this main class in the getter of your class registry. The class object for class C is C.class.

Task 8: Screenshot

Run the main class. Create a screenshot (not a text file) of the console after you run the main class and save it in the Java project folder. I captured my picture using Microsoft Snip and Sketch, which does not seem to have a save as command but does copy the picture to the clipboard. I saved it by pasting it in a PPT slide, selectin it, and using the right menu to save the image as a picture:

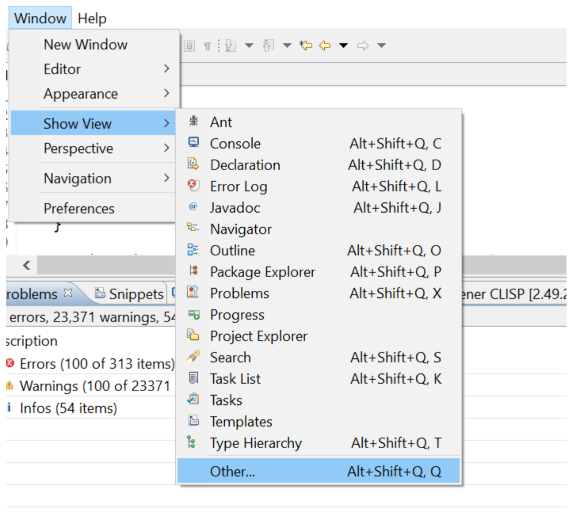


```
<terminated> Hello [Java Application] C:\Program
Hello World
Goodbye
```

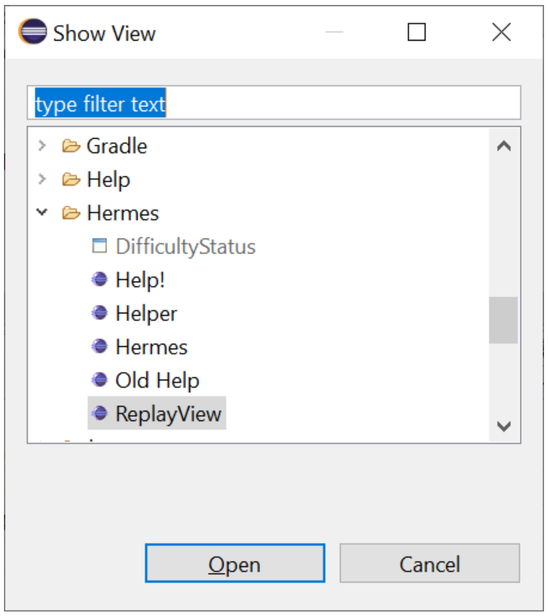
The last remaining step is to submit the assignment to Sakai and Gradscope. There are two ways to do it.

Task 9: Export Zip Using Plugin (Optional)

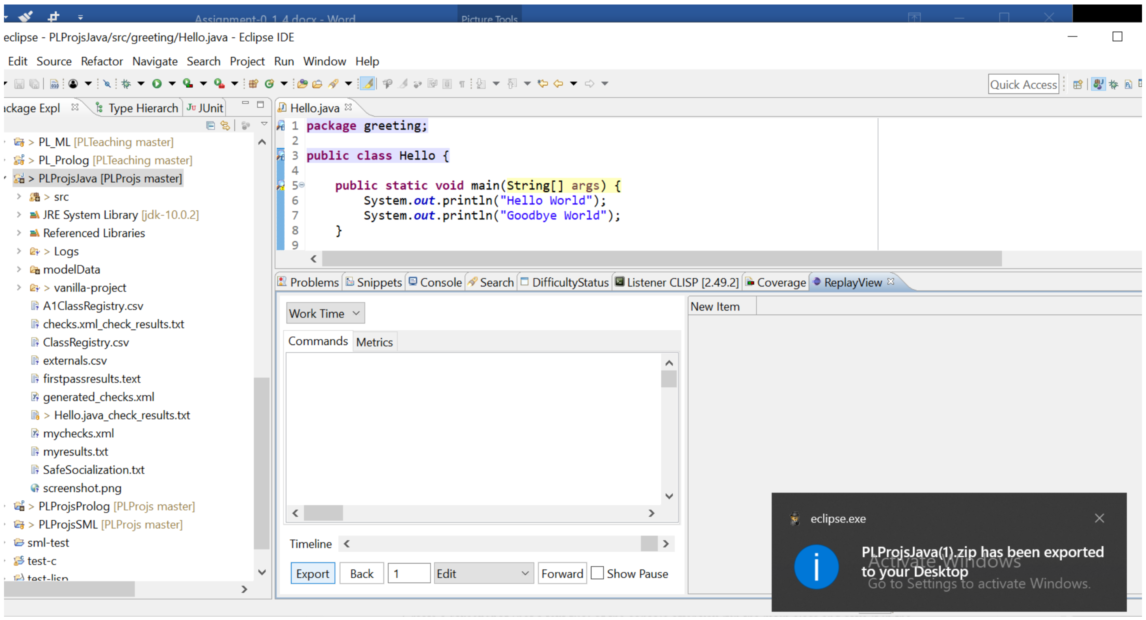
If you have installed the Hermes/Checkstyle plugin, then execute the Eclipse Window   Show View   Other command.



Next select Hermes   Replay View



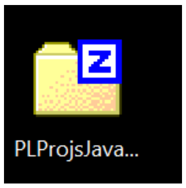
Select the project to be exported and hit the button Export (lower right) in the replay view:



```
package greeting;

public class Hello {

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

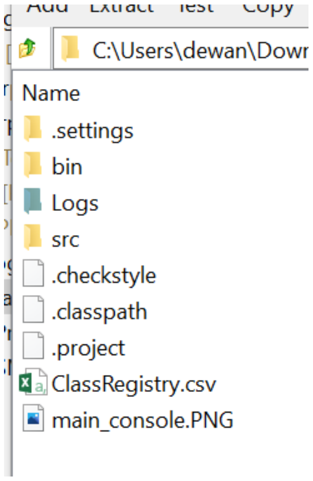
eclipse - PLProjsJava/src/greeting/Hello.java - Eclipse IDE
Edit Source Refactor Navigate Search Project Run Window Help
PL_ML [PLTeaching master]
PL_Prolog [PLTeaching master]
PLProjsJava [PLProjs master]
src
JRE System Library [jdk-10.0.2]
Referenced Libraries
Logs
modelData
vanilla-project
A1ClassRegistry.csv
checks.xml_check_results.txt
ClassRegistry.csv
externals.csv
firstpassresults.text
generated_checks.xml
Hello.java_check_results.txt
mychecks.xml
myresults.txt
SafeSocialization.txt
screenshot.png
PLProjsProlog [PLProjs master]
PLProjsSML [PLProjs master]
sml-test
test-c
Problems Snippets Console Search DifficultyStatus Listener CLISP [2.49.2] Coverage ReplayView
Work Time
Commands Metrics
New Item
Timeline
Export Back 1 Edit Forward Show Pause
eclipse.exe
PLProjsJava(1).zip has been exported to your Desktop

Your zip file is saved on your desktop.



Task 10: Manually Create Zip

Create a zip of the entire project folder including (src, bin, logs, and the screen dump file. In your zip, the project folder name does not matter and we can handle zips with and without a spurious nesting of the top folder.   Here is a view of a correctly formatted zip file.



Name
.settings
bin
Logs
src
.checkstyle
.classpath
.project
ClassRegistry.csv
main_console.PNG

Do not reorganize the folder before submission. For instance, some students have submitted a folder containing the Logs folder and another folder that contains the src and bin folder. Just submit the folder as is as shown above. While we can deal with only an src folder, it is more efficient if you submit the entire project folder.

Task 11: Gradescope and Sakai Submissions

Submit the automatically exported or manually zipped project to Sakai and Gradescope using the appropriate name: Assignment 0-1 and G_Assignment0_1.

The late penalty or early credit we compute for each submission will be based on when it was submitted. Thus, it possible to get early credit on the Gradescope submission and late penalty on the Sakai submission, Usually we will grade only the Gradescope submission. But we have less control over the Gradescope grader and cannot run your assignment manually (if needed) over Gradescope. Hence, the need for both submissions.

Potential Gotchas and Solutions

(Please add to this list)

Problem: When trying to complete task 2 I keep getting the error message: "The type gradingTools.shared.testcases.greeting.GreetingClassRegistry is not accessible". I could use some help fixing this.

Solution: Makes sure the Comp524All.jar file is in the module path and not the class path and delete any module-info file in your project.