

COP5556 Assignment 1

Due: 11 Sept 2017 at 11:59pm

Implement a scanner for the programming language with the following lexical structure:

RawInputCharacter ::= any ASCII character

LineTerminator ::= LF | CR | CR LF

LF is the ASCII character also known as "newline", in java \n

CR is the ASCII character also known as "return", in Java, the char \r

CR immediately followed by LF counts as one line terminator, not two

InputCharacter ::= RawInputCharacter, but not CR or LF

Input ::= (WhiteSpace | Comment | Token)*

Token ::= Identifier | Keyword | Literal | Separator | Operator

WhiteSpace ::= SP | HT | FF | LineTerminator

SP is the ASCII character also known as "space"

HT is the ASCII character also known as "horizontal tab"

FF is the ASCII character also known as "form feed"

Comment ::= / / InputCharacter*

Identifier ::= IdentifierChars but not a Keyword or BooleanLiteral

IdentifierChars ::= IdentifierStart IdentifierPart*

IdentifierStart ::= A..Z | a..z | _ | \$

IdentifierPart ::= IdentifierStart | Digit

Literal ::= IntegerLiteral | BooleanLiteral | StringLiteral

IntegerLiteral ::= 0 | NonZeroDigit Digit*

NonZeroDigit ::= 1 .. 9

Digit ::= NonZeroDigit | 0

BooleanLiteral ::= true | false

StringLiteral ::= " StringCharacter* "

StringCharacter ::= InputCharacter but not " or \
| EscapeSequence

EscapeSequence ::=

\b | \t | \n | \f | \r | \" | \' | \\

Separators ::= (|) | [|] | ; | ,

Operators ::= = | > | < | ! | ? | : | == | != | <= | >= |
& | | | + | - | * | / | % | ** | -> | <- | @

Keywords ::= x | X | y | Y | r | R | a | A | Z | DEF_X | DEF_Y | SCREEN |
cart_x | cart_y | polar_a | polar_r | abs | sin | cos | atan | log | image | int |
boolean | url | file

- If an illegal character is encountered, your scanner should throw a `LexicalException`. The message should contain useful information about the error. The contents of the message will not be graded, but you will appreciate it later if it is helpful.
- If an integer literal is provided that is out of the range of a Java `int`, then your scanner should throw a `LexicalException`. The contents of the error message will not be graded, but you will appreciate it later if it is helpful.
- Use the provided `Scanner.java` and `ScannerTest.java` as starting points.

Turn in a jar file containing the source code `Scanner.java` and `ScannerTest.java`.

Your `ScannerTest` will not be graded, but may be looked at in case of academic honesty issues.

We will subject your scanner to our set of junit tests and your grade will be determined solely by how many tests are passed. Name your jar file in the following format:

firstname_lastname_ufid_hw1.jar

Additional requirements:

- This code must remain in package `cop5556fa17`(case sensitive): do not create additional packages.
- Names (of classes, method, variables, etc.) in starter code must not be changed. You may, of course, add additional variables, methods, enums, etc.
- Your code should not import any classes other than those from the standard Java distribution.

Comments and suggestions:

- The given `Scanner.java` and `ScannerTest.java` should compile correctly. When executed, only one test will pass, but all should pass in your completed scanner.
- **Work incrementally: add a single capability along with a junit test to exercise it.**
- Plan your approach. Pay attention to things that are basically the same--for example, if you can handle a semi-colon you can handle a comma, and all other characters that only appear by themselves in Tokens the same way. Similarly, for `=` (which may be the first char in an `= Token` or an `== Token`) and `!`, which may be the first token in a `! Token` or a `!= Token`.

- If you use `Integer.parseInt` to get the value of a numeric literal, it will throw a `NumberFormatException` if the value is too large. This is useful functionality, but the exception is not the same one as specified. You need to catch it and throw a `Scanner.LexicalException` instead with a useful message.
- `Character.isJavaIdentifierPart` mysteriously also returns true when it is given a value of 0, the sentinel value we are using to mark the end of the input, so you will probably need to check for this case.
- Potentially useful methods from the `java.lang.Character` class include `isDigit`, `isWhiteSpace`, `isJavaIdentifierStart`, `isJavaIdentifierPart`, along with `java.lang.Integer.parseInt`. In my solution, I used `java.util.HashMap` to map keywords to their `Token.kind`.

Submission Checklist

- **Make sure that sources are included in the jar file.** Many IDEs (including Eclipse) do not do this by default.
 - [A quick reference for how to export a jar file from Eclipse](#)
 - If you are not using Eclipse, check [Creating a JAR file](#)
- To ensure that we will be able to compile and run your submission: upload your jar file to one of the ufl cise server, e.g. `storm.cise.ufl.edu`, uncompress it and run from the command line. Instructions:
 - Copy/upload your file to cise server. If your OS is windows, install some ssh client like putty for this step. Or you can use some ftp client(e.g. Filezilla) and skip this step. Suppose your cise id is *username*, the following instruction will upload the *HW1.jar* to your home folder on cise server:


```
scp /my/path/to/HW1.jar username@storm.cise.ufl.edu:~/
```
 - Uncompress file:


```
jar xf HW1.jar
```

 - If you packaged everything correctly, your uncompressed project directory structure will look like following:


```
cop5556fa17
|--Scanner.java
|--ScannerTest.java
|-- *all the other files*
|-- ...
```
 - Compile:


```
javac -cp ./usr/share/java/junit4.jar:/usr/share/java/hamcrest-core.jar
cop5556fa17/*.java
```
 - Run junit test from command line:

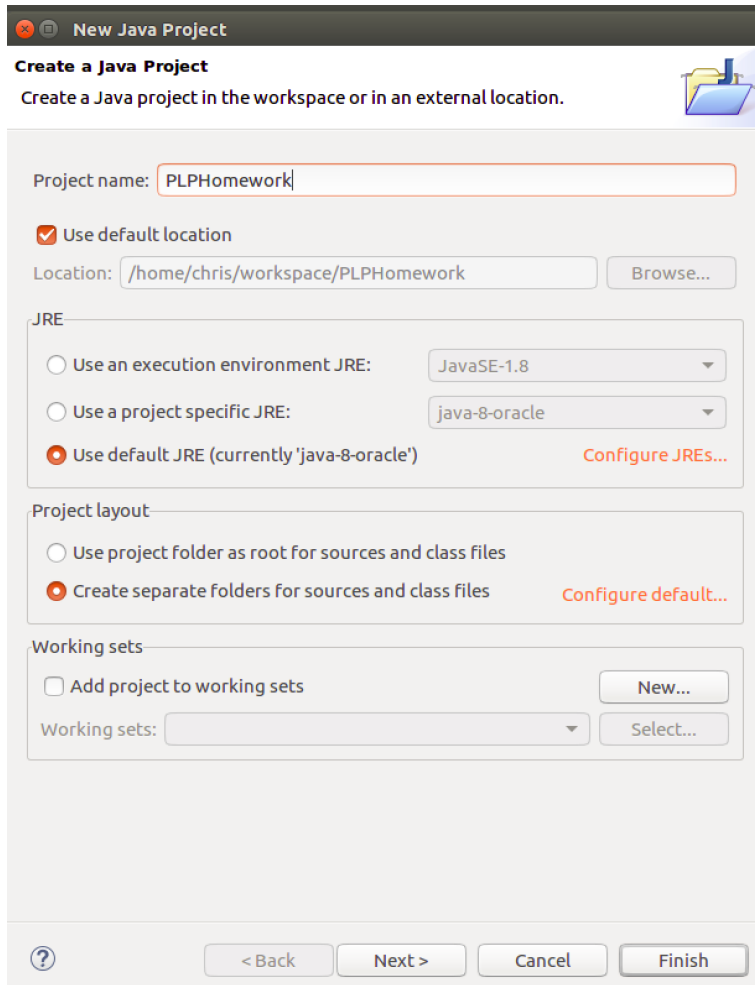

```
java -cp ./usr/share/java/junit4.jar:/usr/share/java/hamcrest-core.jar
org.junit.runner.JUnitCore cop5556fa17.ScannerTest
```
- **Make sure that your jar file has the same directory structure as the original one that you downloaded from Canvas(otherwise, you will fail grading and get 0 grade).**

- Note that you can do this upload in the process so if you have problems, you will have time to figure out what is wrong.
- **No matter how your program runs on your own machine, if it fails to compile/run on the CISE server (storm or thunder) with the aforementioned instructions, your submission will get a zero grade, and there is no regrade. So double check before your submission.**
 - If you are non-CISE student who does not have a CISE account, google and consult the ithelp of CISE to get one account.
- If you use Eclipse, we suggest creating a project and importing the jar files (eg. HW1.jar) provided by each assignment into the project. After completing your work on the source files (keep all source files within the package cop5556fa17), you can export the package cop5556fa17 as a jar file for submission (remember to select the option of including source files in the jar package), so that it will have the same directory structure as the original jar file.
- **Important!!! It is YOUR RESPONSIBILITY to submit the assignment in time. If you keep failing submission on Canvas, contact TAs and ithelp for help. DO NOT jammed submitting at last 10 min before deadline, it's possible to fail submission at that time due to the server overloading. Start early, submit early. You can submit multiple times(not recommended), and we will only grade your last submission.**

A Quick Tutorial on How to Start Homework 1 in Eclipse:

1. Create a project (e.g. PLPHomework)

File->New->Java Project



The screenshot shows the 'New Java Project' dialog box in Eclipse. The title bar says 'New Java Project'. Below the title bar, it says 'Create a Java Project' and 'Create a Java project in the workspace or in an external location.' with a folder icon. The 'Project name' field contains 'PLPHomework'. The 'Use default location' checkbox is checked. The 'Location' field shows '/home/chris/workspace/PLPHomework' with a 'Browse...' button. The 'JRE' section has three radio buttons: 'Use an execution environment JRE:' (selected), 'Use a project specific JRE:', and 'Use default JRE (currently 'java-8-oracle')'. The first two have dropdown menus showing 'JavaSE-1.8' and 'java-8-oracle' respectively. There is a 'Configure JREs...' link. The 'Project layout' section has two radio buttons: 'Use project folder as root for sources and class files' and 'Create separate folders for sources and class files' (selected). There is a 'Configure default...' link. The 'Working sets' section has a checkbox 'Add project to working sets' and a 'New...' button. Below it, the 'Working sets:' dropdown is empty, and there is a 'Select...' button. At the bottom, there are buttons for '< Back', 'Next >', 'Cancel', and 'Finish'.

New Java Project

Create a Java Project
Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location:

JRE

☐ Use an execution environment JRE:

☐ Use a project specific JRE:

☒ Use default JRE (currently 'java-8-oracle') [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

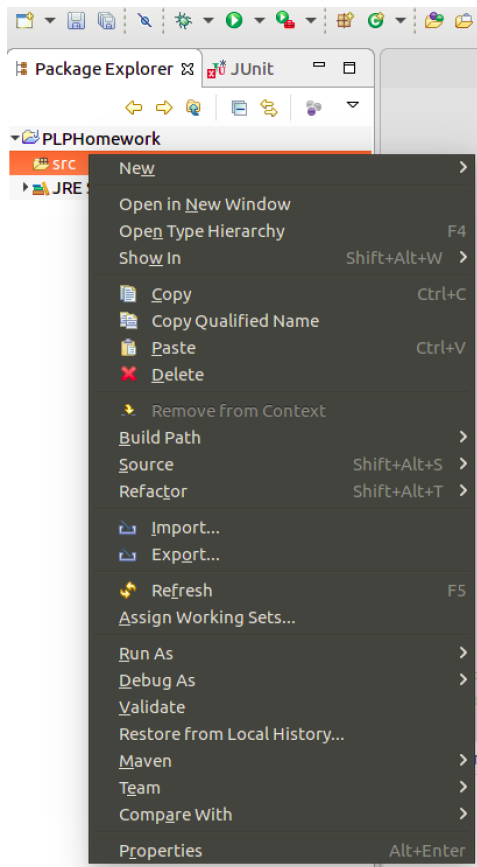
☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

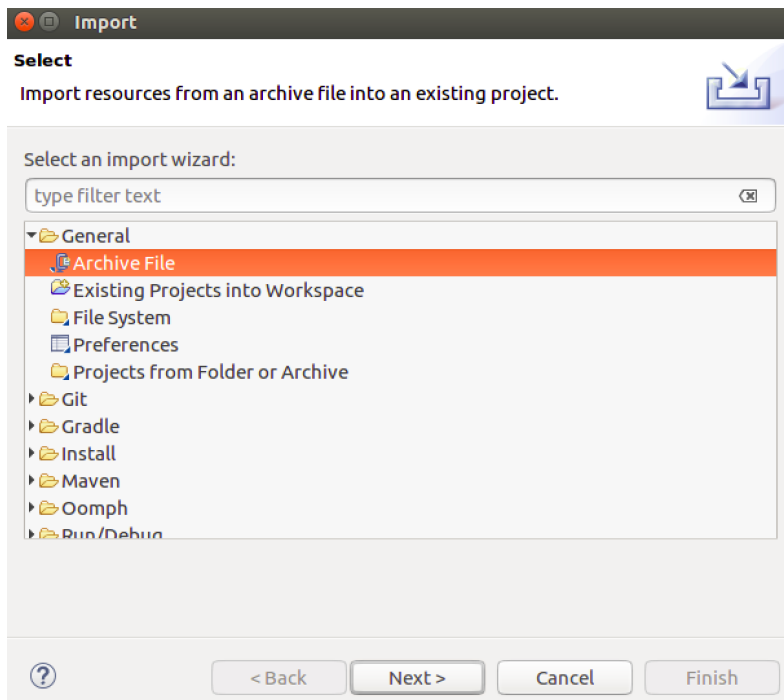
☐ Add project to working sets

Working sets:

2. After project created, right click on the src folder in the left sidebar, choose Import t...

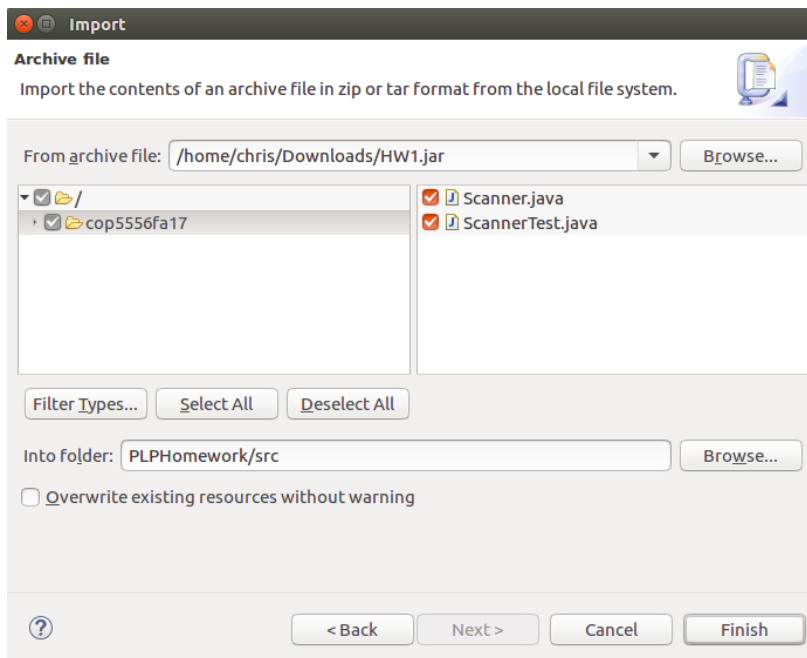


Select General->Archive File



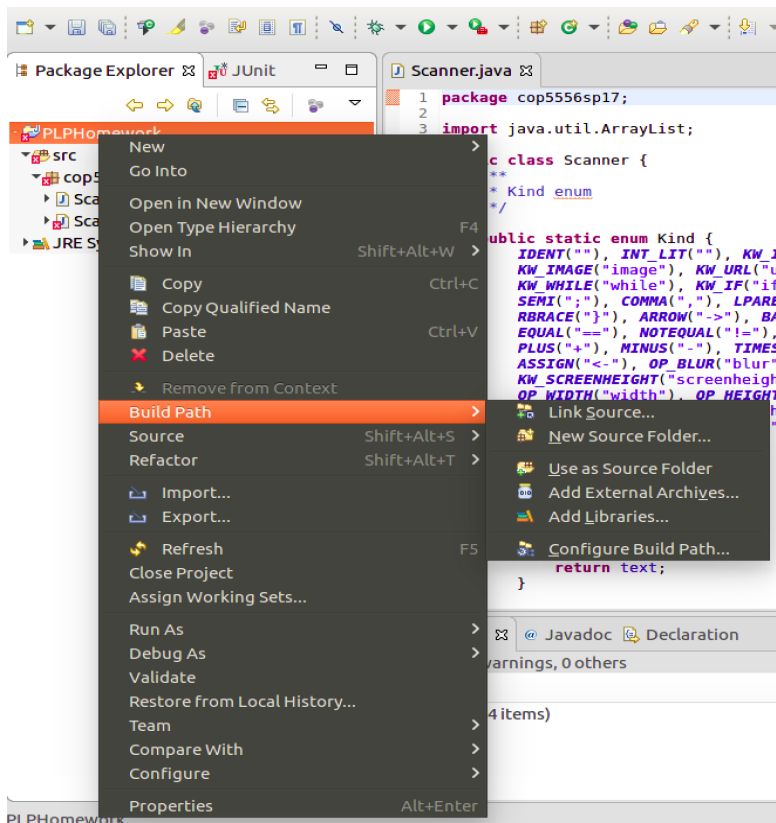
Browse and choose your downloaded HW1.jar, make sure both Scanner.java and

ScannerTest.java have been checked

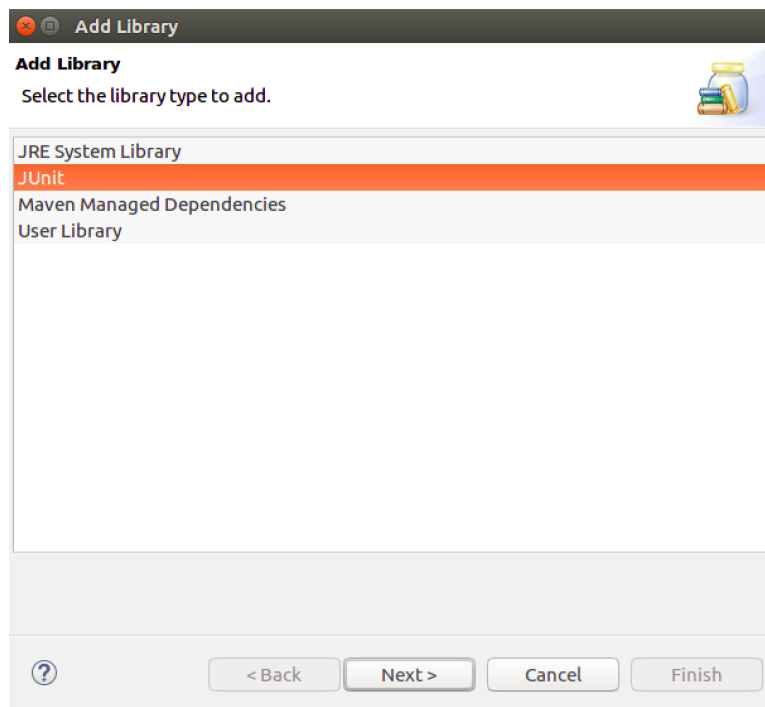


3. Add Junit Library to Build Path.

Right Click on project, select Build Path->Add Libraries...



In the list, choose JUnit



4. To Run the unit tests

