CS206 Introduction to Data Structures

Lab 4

Command Line Arguments, Interfaces, and Exceptions

Thursday, Feb 13

More about directories: ls and cd are really useful. There are some things that make them more so.

~	denotes your home directory.	"`ls ~`" always gives a listing of the contents of your home directory
		"`cd ~`" will take you to your home directory
..	indicates the directory one up from your current directory	"`ls ..`" lists the contents of the directory one up from where you are
		"`cd ../aa`" move up one directory and then down into the directory `aa`.
.	represents the current directory	"`ls .`" is exactly the same as "`ls`". "`cp ../../A.java .`" copy into the current directory the file `A.java` from a directory two up

Exercise 1: Fill in the table below:

In the result column, describe what you would see, not specifics. For instance, "I would see the contents of my home directory". Assume that in the cs206 directory exists and there are at least two subdirectories, Lab1 and Lab2 of cs206.

Current location	Command	result
/home/YOU/cs206/Lab1	ls ..	
/home/YOU/cs206/Lab1	ls ~/Lab1	
/home/YOU/cs206/Lab1	ls ../..	
/home/YOU/cs206/Lab2	ls ~/../YOU	
/home/YOU/cs206/Lab1	cd ./..	

Exercise 2. Command Line Arguments and importing classes

This exercise very briefly introduces the topic of providing information to your program from the command line.

1. Create a new folder in Visual Studio Code — call it Lab4
2. Import the class Main.java from the file
   `/home/gtowell/Public206/lab04/Main.java`
   into the project by doing the following:
   1. Open a terminal window
   2. cd to the Lab4 directory you just created. Assuming you followed class conventions and the directions above execute
   3. `cd /home/YOU/206/Lab4`
   4. Copy the file
   5. `cp /home/gtowell/Public206/data/lab04/Main.java .`
      (Note the use of the "."; it is as described above!)
3. The file Main.java should appear in Visual Studio Code
4. Run the newly imported class from within VSC. What is the output?
5. Go back to the terminal window opened in step 2.
6. Run your program from the command line as follows:
   ```
   javac Main.java
   java Main
   ```
7. Run your program again but this time
   ```
   java Main a s d f g
   ```
8. What is the difference? Why?

9. Run you program one more time (replace YOU with your UNIX login name):

```
java Main /home/YOU/*
```

10. Compare your output to the following unix command which lists every file in a directory
11. `ls /home/YOU`
12. Are the lists of files the same? Explain?

Exercise 3: Circular Linked Lists

In this exercise you will import several classes that implement a CircularLinkedList.

A circular linked list is like a singly liked list except that it is a circle rather than a line. Hence, there is no well defined starting point (head) or ending point (tail). Rather every node is linked to the another node. Follow the circle long enough and you get back to where you started.

After importing, you will fully implement the size method that exists only as stub in the provided class. Note that this implementation does not have a size variable. (Do not add it.) Hence, the number of items in the list must be computed whenever the size method is called.

Here is the stub definition of size.

```
/****
    * @return the number of items in the list
    ****/
   @Override
   public int size()
   {
   return -1;
   }
```

Classes for this exercise are in `/home/gtowell/Public206/data/Lab04/`.
Into your Lab4 folder import: Rabbit.java, CircularLinkedList.java, LinkedListInterface.java and CLLMain.java using the procedure above. Note that in CircularLinkedList.java, the size method appears exactly as above.

After importing, implement the size function.

Finally, extend the main method of CLLMain.java to test your size.

When you have completed these tasks put your name on a page along this the total number of lines of code you added to CircularLikedList.java. If you did not compete these tasks, give a brief summary of how far along you got.