CS400 Week 1 Summary
Andy Kuemmel

0. TO DO list. These are things we recommend you do to prepare for success on your programming assignments and exams. This week we recommend that you:

a. Create a CS Lab Account, login to your CS Lab Account, do the linux examples in the Week 1 notes
b. Find your CS300 programs that concerned ADTs and Interfaces and review that code
c. join our Piazza site, look here later for a join code: **piazza.com/wisc/spring2019/cs400**

1. Linux. Learning this powerful operating system will give you access to the CS lab machines. Many students in upper level CS courses have asked that these skills be taught before they take these harder courses, which is why we are introducing you to linux now. Throughout the course we expect that you are practicing the things we show you in linux.

On page 4 of the notes, we practiced these commands (don't forget the `.` at the end of the first line)

The dot tells linux to place the copied file in the current directory

```
cp  /p/course/cs400-deppeler/public/html-s/code/Hello.java  .
ls
javac Hello.java
ls
java Hello <yourname>
```

2. Because we only had 2 lectures this week, we didn't quite get through all of our lecture material. On Monday, I will start on page 7, Test Driven Development. Here is a summary of the notes that you should copy down on page 6.

Testing:
Should be written in code, and preserved, not deleted or overwritten, as you build your program pieces.
Good Tests Have:
A name, a well-defined input sequence, expected output, actual output
Should clearly denote what is passing and what is failing
Comprehensive Testing:
Test all combinations of operations, over carefully chosen input sequences or data sets
Black-Box Testing: (What we focus on in program p1)
We know the ADT (the behavior) but do not see the source code
Test the functionality of the ADT operations over possible multiple input sequences
Compare the actual result to the expected result
This does not guarantee all code paths were visited (we can't see the code)
White-Box Testing:
We can see the code, and we write tests to ensure that all code paths were visited
We can test internal results, including private members
Unit Testing:
test small units such as each method, each class, or each package
try to write tests that are not dependent on other parts of the program if possible
confirm small units work before testing larger units

System (End-toEnd) Tests:

tests the interaction between units
look for correct output for a given input sequence
this test is the most difficult to do because of all the different parts and all the possible inputs

3. Clarifying the subtle differences between an ADT, a Java Interface, and a Data Structure

An Abstract Data Type is a data type described by predefined user operations without indicating how each operation is implemented. An ADT is not language-specific. An ADT is Abstract because it is only a model and not a concrete class. An ADT is often meant to work on any type, so we use Generics in Java to allow this. An ADT is often referred to as a "model of a data structure" because it does specify how the data structure will be implemented.

In Java, an interface is the mechanism by which an ADT is specified. The interface specifies the operations with method headers that show the type of the input and output.

A Data Structure can be thought of as the class that implements the interface. The instances of the Data Structure are concrete examples of the Abstract Data Type. There can be many ways to implement a given interface, and the Data Structure hides the implementation details through private members.

While the above explanation of Abstract Data Type is not consistent across all textbooks, I do feel that is helpful as we go forward. We will discuss an Abstract Data Type in our lecture notes, provide you its interface in Java, and then you will implement a class that meets the interface. For more information, check out the Abstract Data Type Wikipedia page.