Introduction to data structures, algorithms, and object-oriented programming

DAY 7
Java has classes that are crosses between an array and a list.

ArrayLists are like arrays in that:

1. Items can be accessed directly by index number
2. When an ArrayList is declared, its data type is specified.

ArrayLists are like lists in that:

1. The size is dynamic, can be expanded and shortened. In other words, it only uses as much memory as there are items in the list.

The ArrayList is declared as follows:

```java
ArrayList<Integer> aLine = new ArrayList<Integer>();
```

Since ArrayLists can contain only data of object type, to make one that holds integers requires use of the Integer class.
for-each loop

Used to traverse array or collection of elements. The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.

Syntax:   for( data_type variable : array ){}

```java
class ForEachExample1{
    public static void main(String args[]){
        int arr[]={12,13,14,44};

        for(int i:arr){
            System.out.println(i);
        }
    }
}
```
for-each loop

Traversal of an ArrayList to add 1 to each element:

class ForEachExample1{
    public static void main(String args[]){
        ArrayList<Integer> intArrLst = new ArrayList<Integer>();
        for (int i = 0; i < 100; i++) {
            intArrLst.add(i);
        }
        for(Integer i : intArrLst){
            inArrLst.set(i, i+1);
        }
    }
}
import junit.framework.TestCase;
import static org.junit.Assert.*;
import org.junit.Test;

public class TryFib extends TestCase {

    /* Test case for fib(1) */
    public void testrecfib1() {
        int actual = RecFibonacci.getFib(1);
        int expected = 1;
        //Using assertEquals to check-expect
        assertEquals(expected, actual);
    }
}

Communicating with a Method

• Getting information into a method:
  – Explicit input parameters.
  – Class and instance variables of the object on which the method is invoked can be used inside the method

• Getting information out of a method:
  – Changes to "state" of object: class and instance variables:
    • Of input parameters that are reference types to the method.
    • Of the object on which the method is invoked.
The book calls a method whose job is to compute and return some value a *function*. The return-type is used to specify the type of value that is returned by the function.

Static vs non-static methods:

In a running program, a *static method* is a member of the class that contains it. It is called on the class name.

A *non-static method* can be called only on objects of the class type, and such methods are members of the object. To make it clear, when calling this type of method, you can call it on "this"
A program can contain many methods, but only one main method.

All methods are contained with a class block and no method can be written inside another (although methods can call each other). Methods can contain any kind of statement (except package and import statements.)

General form of method headers:

```java
modifiers return-type methodName ( ParType parName) {
    statements
}
```
Parameter lists

Parameters are part of the interface of a method. They contain information that is used inside the method after the method is called.

Parameters are not given values until the method is called and arguments are passed into the method.

Unlike variables created outside a method, the variables contained within the parameter list must each have its type specified and each TypeName varName pair must be separated by commas.
Calling Methods

The syntax for calling a method that exists inside the same class as the method that is calling it *from a non-static context* is as follows:

```
methodName(argument(s));
```

The syntax for calling a *static method* in the same or another class is:

```
ClassName.methodName(argument(s));
```

If the method returns a value, you should declare a variable to hold that value, use it as an argument to a method, or use it as part of an expression.
General Rule of Java Program Structure

Methods are never written inside methods (unlike the local special form in Racket). But methods can call other methods.

Methods can have any number of parameters, including 0.

The first line of a method is called the method header or signature, e.g.

```
modifiers return-type subroutine-name ( parameter-list )
```
The method and the parameter list is called the *method signature*. As we discussed in the last class, we can OVERLOAD methods by writing more than 1 methods with the same name in a class, but each must have a unique parameter-list.

\[
\text{subroutine-name ( parameter-list )}
\]

One of the most common places where overloading exists is with constructors. A single class may have many different constructors, but each must have a different parameter list.
Member Variables

Local variables are those declared inside a method.

These variables are not assigned a default value and must be initialized before being used in an expression.

Global variables are declared inside a class and are called member variables or fields.

Member variables can be static, meaning they are used in expressions following the class name.

Member variables are assigned a default value.
Returning values from a Method

The author of our text calls methods that return a value *functions*. These methods must have a return type that is not void.

You need to explicitly return a value of the type given in the method header line.

```
return expression;
```

Only one value can be returned in a return statement. Executing a return statement breaks out of the method and returns control to the statement that called the method.

In a method that returns void, control can be returned to the calling method by using return with no expression:

```
return;
```
Method Contracts

Prior to writing each method, you should write a comment, or contract, to explain what the method does and any assumptions about parameters.

Parameters can be of any type, including arrays.

```java
String[] names = {Gerry, Roger, Helen, Ann};
countLetters(names);

static int countLetters(String[] noms) {
    int count = 0;
    for (int i = 0; i < noms.length; i++) {
        <add code here to count letters>
        <add statement here to return count>
    }
}
```
Reverse a String

Since Strings are numbered like arrays, and because the String class has a function length(), it is easy to use a for loop to build and return the String in reverse.

(in class exercise)

Convert the method to a palindrome checker.
Write a JUnit tester class

After writing the reverseString class, write a tester class to test various String inputs.