I. MODULAR PROGRAMMING

week 14/15

- modular programming – dividing programs into smaller pieces (modules)
- this allows for:
  1. better problem solving by dividing a complex problem into smaller pieces
  2. use modules to organize/group code for reusability
  3. allows for more complete and rigorous testing
  4. use parameter passing mechanisms to **securely** share data (communicate) between code modules
- consider the concept of communicating personal information to a friend in a room, you could
  - speak to the entire room (least private or secure)
  - speak in a normal voice to your friend
  - pass your friend a note (most private or secure)
- in Java, these modules are called **methods**
  - a method is a grouping of code assigned to perform a specific task
  - a method can be a system defined method or a user defined method
  - we have seen and used system methods, such as Double.parseDouble or Math.pow
  - we have also seen and used non-static (instance) methods, such as the getter and setter methods
- in Java, there are also methods called **static** methods
  - static methods belong to the class itself, not a (new) instance of it
  - static methods do not use instance variables, rather data will come from parameters
  - static methods use the **static** keyword
  - to create a static method, it must be given a descriptive name
  - a static method may or may not have data values "passed" to it
  - a static method may return **zero or one** value

- a static method will consist of two basic parts:

1. method definition

- methods will be defined by their signatures, which will look like:

  access_modifier(s) return_type method_name (parameter list)

- access_modifiers will usually be keywords: public static
- return_type will specify what type of data the method returns (void will indicate no data will be returned)
- method_name will be the name used to reference (or use) the method
- parameter_list will consist of zero or more parameters, each prefixed with its specific type, separated by commas

- note that the return type and the parameter types DO NOT have to be the same types…while these are related, the types DO NOT have to be identical

- example definition:

```
public static double findMax (double num1, double num2)
{

        double max;

        if (num1 > num2)
                max = num1;
        else
                max = num2;

        return max;
}
```

- suggestion: in most cases, a **single** return statement should be used to return a value

2. **method invocation (or call or use of the definition)**

   - when calling the method definition, you don't use most of the keywords
   - to invoke, or call or use a method:

     value = method_name (*argument list*); // note value = is optional
   - note value returned and argument list are optional, based upon definition
   - example (from above)

     double max, firstNumber, secondNumber; // declare some variables

     max = findMax (firstNumber, secondNumber);

- **General Rules for Parameter Passing**

  1. Number of arguments must match the number of parameters
  2. Order must be maintained, e.g. arg1 ↔ param1
  3. Data types must match between matching parameters and arguments
  4. Names are only names and DO NOT have to match
  5. You can also use constants for argument values