CS 100 Assignment 3: Methods and OOP
Summer 2001
Due in lecture, Friday July 13

We judge ourselves by what we feel capable of doing,
while others judge us by what we have already done. – Henry Wadsworth Longfellow

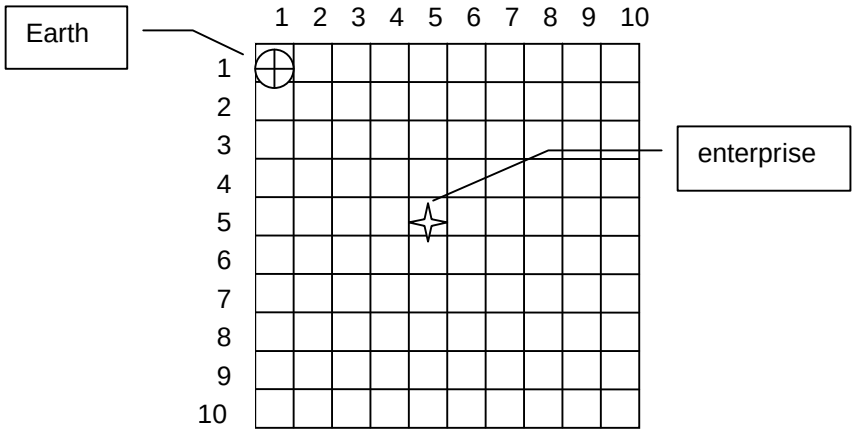
Objectives

Completing all tasks in this assignment will help you:
- implement a set of methods and observe how they work together
- consider what a "tester" program should do to help automate the process of determining whether a method has been implemented correctly
- take a problem and break it down into manageable chunks that can each be implemented as a separate method

First, skim the whole assignment. Then carefully read all instructions before starting. You must use a **fixed-width** font (e.g., Courier or Monaco) for your homework and you must **staple** your assignment together. If you are not sure whether you are using a fixed-width font, ask a consultant or TA to check it for you (sometime *before* the assignment is due!).

1. Implementing Methods (20 points correctness, 20 points style)

In this part of the assignment, you will help implement a Spaceship object. The Spaceship has already been partially constructed for you; you will be filling in the remaining methods. The Spaceship exists in a two-dimensional space that is quite limited: it is a 10x10 grid, which looks like this:



In SpaceshipTester.java, the newly-created Spaceship **enterprise** starts out at position (5,5). The Spaceship can
- **move()** horizontally (positive numbers go right) and vertically (positive numbers go down)
- **hyperjump()** to a new position on the grid
- **refuel()** (fill the fuel tank to capacity), if returns to Earth

The pilot can also get information from the ship:
- **getLocation()** to determine where the ship currently is
- **getFuel()** to see how much fuel it has in its tank (starts out with 30.0 tons)
- **displayStats()** to print out both location and fuel in a nice way

1. Download **Spaceship.java**, **SpaceshipTester.java**, and **Position.java** from the website. You should only make changes to Spaceship.java. Create a new project and

add all three files to it (you will not need SavitchIn.java). Set the target to **SpaceshipTester**.
2. Carefully read through Spaceship.java. Note which methods have already been implemented and which ones you will need to implement.
3. Add a comment block to the top of Spaceship.java with the following information:
   Name: (your name)
   ID: (your Cornell ID)
   Date: (current date)
   Assignment: 3, Spaceship.java
4. Now look at SpaceshipTester.java. If you compile and run the project right now, the output will look a little strange. The methods in Spaceship.java that you will implement are currently returning "dummy" values (so that the program will compile and run as is). You should **delete these `return` statements** and **replace them with your implementation** of each method. As you write each method, more of SpaceshipTester's output will be correct.
5. Print Spaceship.java and turn it in at lecture.

2. Testing Procedures (5 points correctness, 5 points style)

SpaceshipTester.java only tests some of the functionality of Spaceship.java. There are several other tests that can (and should!) be run, before deciding that Spaceship.java has been implemented correctly.

1. Create a new java file, **SpaceshipTester2.java**, and put the following information in it:
   Name: (your name)
   ID: (your Cornell ID)
   Date: (current date)
   Assignment: 3, Testing the Spaceship
2. Decide on a **useful test** you could run that does not appear in SpaceshipTester.java (you might want to test the methods with different inputs, or in different situations). Create a main() method in SpaceshipTester2.java and **write the code** to perform your test. (See SpaceshipTester.java for an idea of how to set things up.) Don't forget to comment it and otherwise use good programming style!
3. Print SpaceshipTester2.java and turn it in at lecture.

Bonus point 1: We would like to enable Spaceships to communicate with each other. For a bonus point, add a new method to the Spaceship class:

```
// BONUS POINT 1
// sendMessage: sends 'message' from the current Spaceship to 'recipient'.
//              However, the two ships must be no more than two units
//              of distance apart, for the message to be received.
//              If the ships are close enough to communicate,
//              print out a notification indicating what message was sent.
// Input: a Spaceship recipient and a String message
// Output: if the message is successfully received, return 0.
//         otherwise, return 1.
public int sendMessage(Spaceship recipient, String message)
{
   // Your implementation goes here.
}
```

You can test your method by uncommenting the relevant part of SpaceshipTester.java (near the end of the main() method). As always, you may wish to write your own testing code as well! For the bonus point, you only need to include this extra method in your Spaceship.java.

3. Method Design (5 points correctness, 5 points style)

We talked in lecture about how methods can improve a program's readability by eliminating redundancy.

1. Download **NeedsMethods.java** from the website and add the following in a comment block at the top of the file:
   Name: (your name)
   ID: (your Cornell ID)
   Date: (current date)
   Assignment: 3, NeedsMethods.java
2. This program has only one method, main(). However, there are parts of the program where similar blocks of code end up being repeated. Examine the program carefully, and **decide what method or methods** should be created to improve the program's style.
3. For each method, you will need to:
   - select a name.
   - decide how many parameters the method will have, and what type each one will be; select a name for each parameter.
   - decide whether the method should return a value; if so, decide what type it should be.
   - provide the full method implementation (you will probably be cutting and pasting, with some small modifications, from chunks of code in main() to do this). Note that because this program is not using objects, any methods you create should be declared **static** (also, they should all be **public**).
   - include comments before the method that describe **what the method does**, its **inputs**, and its **outputs**. For an example, see Spaceship.java.
4. **Add your implemented method(s)** to NeedsMethods.java. **Modify main()** so that it calls your methods instead. The program should behave **exactly** the same way, before and after your changes. In other words, you may not change the order in which actions are taken.
5. Print NeedsMethods.java and turn it in at lecture.

Bonus point 2: You may have noticed that the vowel counter only looks for lower-case vowels. Modify the program so that it reports the correct output for strings such as "2001: A Space Odyssey" and "All good things must come to an end." Put a comment before your changes that includes "**BONUS POINT 2**".

Bonus point 3: The word-counting method isn't flawless, either. Create a comment block just under your identifying information, write "**BONUS POINT 3**", and give **three examples** of different strings for which the word-counting method will report the wrong value. (Your examples should cause it to make different kinds of mistakes.) You do not need to modify the code to correctly handle these cases.