CS 100 Assignment 4: OOP, Class Design, Graphics
Summer 2001
Due in lecture, Thursday July 19

For a man to achieve all that is demanded of him,
he must regard himself as greater than he is.
– Johann Wolfgang von Goethe

Objectives

Completing all tasks in this assignment will help you:
- Define some terms commonly used in OOP
- Design and implement your own class
- Experiment with some basic graphics methods

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.

Submission for this assignment's java files will be done ONLINE. The text file in part 2 and the screenshot in part 3 should still be printed out and turned in at lecture. To submit your files, go to

http://submit.cs.cornell.edu/cs100/scripts/submit.pl

This link will also be available from the Assignments page of our website. For submission purposes, your username will be your Netid. Your password, however, is **not** the password you use for Bear Access. Instead you will be sent a different password via email (to your Netid@cornell.edu) which you should use for submitting work.

1. Spaceships and Planets (40 points correctness, 40 points style)

In this part of the assignment, you will build on the Spaceship class we've already seen. You should download the new and improved Spaceship.java. It has instance variables properly marked private, and some other updates and modifications. Note that Position.java has also been updated slightly, so make sure you have the new one. The proper files should have **ASSIGNMENT 4** in their headers.

Here are the methods you implemented in Assignment 3:
- **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
- **getFuel()** to see how much fuel it has in its tank (starts out with 30.0 tons)
- **isValidPosition()** returns true if the input is a "valid" point in our two-dimensional space-grid

We're now going to add a new class, **Planet**, and update the Spaceship class. Planets occupy a fixed location in our space and cannot be moved, like Spaceships can. If a Spaceship occupies the same location as a Planet, it may choose to land there. Once landed, the Spaceship cannot move() (or hyperjump()) until it launches back into space.

1. Download **Spaceship.java**, **SpaceTester.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 **SpaceTester**.

2. 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: 4, Spaceship.java

3. For this assignment, you'll need to design and implement a Planet class. Planets each have a name associated with them and a position in space. First, create a new java file called Planet.java (and add a similar comment block to the top of it).

- Decide what **instance variables** you'll need to represent a Planet and add them to the Planet class. Remember, instance variables should be **private**.
- Decide what **constructors** you'll need. (You should have a default constructor and at least one other constructor.)
- Decide what **accessor** methods you'll need (to get the values of the instance variables).
- Decide what **update** methods you'll need (to change the values of those variables in "safe" ways).
- Implement a **toString()** and an **equals()** method for the Planet class.
- Add any other methods you think might be useful.

4. Now carefully read through Spaceship.java. **Note the changes that have been made from the previous Spaceship.java.** There are some methods you will need to implement, and others which you will need to update.
   - Implement a **default constructor** for the Spaceship class. Remember that constructors should assign values to all of the instance variables.
   - Implement a **constructor** for Spaceships that start out on planets. It should take in a Planet and set the ship's x and y coordinates (to the same as those of the Planet it is on). As usual, the ship gets a full fuel tank.
   - Implement a **constructor** for Spaceships that takes in a String name for the Spaceship as well as x and y coordinates.
   - Update **move()** and **hyperjump()** so that Spaceships can only move (or jump) if they are not currently landed on a Planet. See Spaceship.java's comments for information on what you should return if these methods fail because the Spaceship is on a Planet. Note that the return values for the other error conditions should also be changed!
   - Implement a **land()** method that takes in a Planet and lands the Spaceship there. This method must check to see if the Spaceship is at the same location as the Planet. If not, it cannot land there, and should return an error value (see Spaceship.java comments). In addition, landing requires 0.1 tons of fuel.
   - Implement a **launch()** method that takes in no inputs and leaves the Planet it is currently on. If the Spaceship is not landed on a Planet, return an error value. (see Spaceship.java comments). Launching requires 0.5 tons of fuel.

5. Test your newly implemented methods using SpaceTester.java. Then create your own testing file, SpaceTester2.java, that includes additional tests (SpaceTester.java is deliberately not comprehensive!) It is **your responsibility** to ensure that you've tested your code appropriately. You can think of SpaceTester.java as a set of "alpha" tests of your code. We will be running our own set of additional "beta" tests on your submitted code. It is to your advantage to try to anticipate what you think we might test, and run those tests before handing in your homework.

6. Submit Spaceship.java and Planet.java at the submission website (see above). You do not need to submit SpaceTester2.java.

2. Object-oriented Programming (10 points)

1. Create a new text file and put the following information in it:
   Name: (your name)
   ID: (your Cornell ID)
   Date: (current date)
   Assignment: 4, OOP Questions

2. Answer the following questions in your file:
   1. What is the difference between a **class** and an **object**?
   2. What does it mean if a variable or method is declared **private**?
   3. What does it mean if a variable or method is declared **static**?
   4. Explain what a method **precondition** is.
   5. What are two motivations for using **encapsulation**?

3. Print your text file and turn it in at lecture.

3. Graphics (5 points correctness, 5 points style)

After the lecture on Graphics and reading Savitch Chapter 15, you'll know how to draw basic shapes using Swing and AWT, the graphics libraries provided with Java. In this part of the assignment, you'll get a chance to flex your creative muscles and show us what you think a spaceship should really look like.

1. Download **SpaceshipImage.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: 4, SpaceshipImage.java

2. In this program, the main() method has already been written for you. Your job is to **implement the paint() method** so that when the program is run, the resulting graphics window displays a spaceship. Here are some guidelines:

   - The background of the image should be **black** (see the constructor).
   - You must specify an appropriate **title** for the window (see the constructor).
   - You should call at least **five different drawing methods** in your program.
   - For help getting started, see the Madeleine example in Savitch, p. 954-5. However, you do not need to go overboard with declaring 30 named constants, as Savitch did. Use constants where appropriate (for values you reuse often in your drawing commands), and select useful, informative names for them.
   - For a list of useful drawing methods, see Savitch p. 957-8, 962. If you wish to label parts of your Spaceship, see the **drawString()** method (p. 980).
   - You may define your own colors, if you like (see p. 970 – 973 in Savitch).

3. Submit SpaceshipImage.java at the submission website. Print out a **screenshot** of the graphics window that shows your spaceship and turn it in at lecture (write your name and CUid on the screenshot).

Bonus point:
Make a copy of SpaceshipImage.java in a new file called **SpaceshipActionImage.java**. In this file, add a button to your image which, if clicked, displays exhaust flames (if your ship has an exhaust) or another indication that the spaceship is moving. This button should be labeled **"Go!"**. In addition, add a second button labeled **"Stop!"** which makes the exhaust flames disappear. For hints, see section 15.1 in Savitch (the SadMadeleine example).

To have your bonus work graded, you must submit **both** SpaceshipImage.java and SpaceshipActionImage.java, as **separate files**. If you submit the bonus point version as SpaceshipImage.java, we will not grade the bonus part. If you only submit SpaceshipActionImage.java (forgetting SpaceshipImage.java), we will not be able to grade part 3 of your assignment.