AP Computer Science
Homework Set 2
Class Design

P2A. Write a class `Song` that stores information about a song. In later programs we will use this `Song` class in our `MyPod` and online store.

Here are the specs for the `Song` class:

a. Class `Song` should include at least three instance variables that represent characteristics of a song (e.g., title, artist, and cost).
b. Class `Song` should include a three-argument constructor to initialize all instance variables.
c. Class `Song` should have a `toString()` method to display its instance variables in a user-friendly format.
d. Add an additional instance variable of your choice to the `Song` class. Modify the constructor and `toString()` method to accommodate the additional instance variable.
e. Add a zero-argument constructor the class to initialize the `Song` to "zero" or "null" values.

Write a separate "`SongDriver`" class to:

i. Create an instance of a `Song` called "`wakeUpSong`" and initialize its instance variable using the object's zero-argument constructor,
ii. Display the instance variables of the "`wakeUpSong`" using the object's `toString()` method,
iii. Create an instance of a `Song` called a "`goToSleepSong`" and initialize its instance variables using the object's multi-argument constructor, and
iv. Display the instance variables of the "`goToSleepSong`" using the object's `toString()` method.

P2B. Let's create a `Student` class that models a Loyola student. The `Student` class should have the following:

a. two instance variables for the student's first and last name, and at least TWO other instance variables chosen by you.
b. a zero-argument and multi-argument constructor to initialize all instance variables. Choose the most appropriate variable types (i.e., `int, double, String, boolean`) for each instance variable.
c. a `toString()` method to display the object's instance variables in a user-friendly format.

Write a separate "`StudentDriver`" class to:

i. create an instance of a `Student` called "`senior001`" using its zero-argument constructor,
ii. display the instance variables of the "`senior001`" using the object's `toString()` method,
iii. create an instance of a Student called a "`frosh001`" and initialize its instance variables using the object's multi-argument constructor, and
iv. display the instance variables of the "`frosh001`" using the object's `toString()` method.

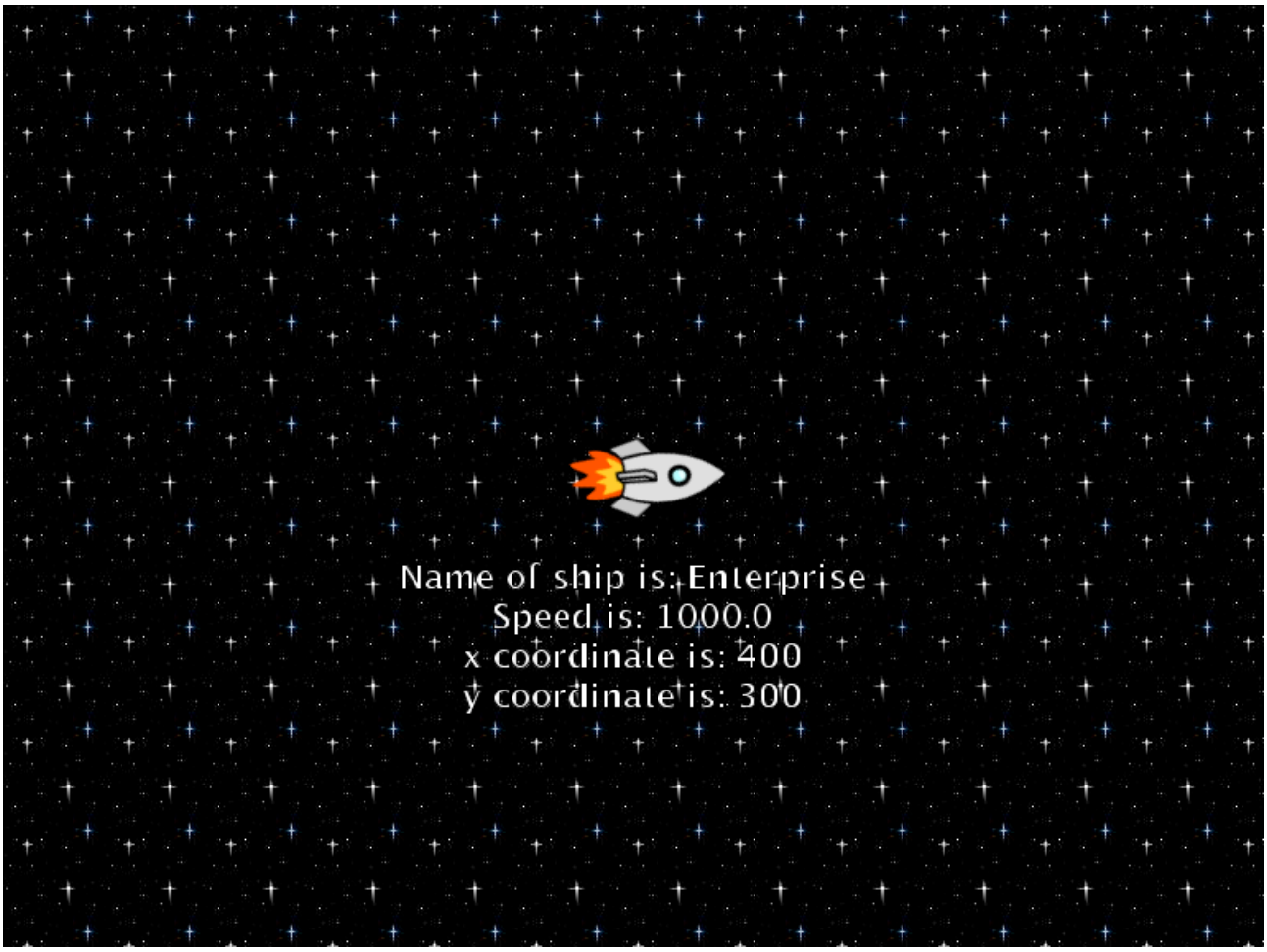
If you are looking for an extra challenge, try to printout the results of the program using `JOptionPane.showMessageDialog()`. See the LewTube video for this program set on the details of how to use this `JOptionPane` class method.

GUI Programs – We will create a series of graphical user interface (GUI) based programs which will use Greenfoot's World and Actor classes. See the LewTube videos on how to create simple "subclasses" of the World and Actor classes for these graphical programs.

P2C. (Greenfoot) Design a class that models your own fusion powered `Spaceship` that will travel through intergalactic space. Here are the specs for your `Spaceship`:

a. Include instance variables to represent the `name` and `speed` of your `Spaceship`.
b. Include zero and multi-argument constructors for the `Spaceship`.
c. Modify `Spaceship's act()` method to call the built-in `move()` and `turn()` methods. Note that you can modify the operation of `move()` and `turn()` by placing integers (or variables that hold integers) in parenthesis. For example, `move(5)` will move the spaceship 5 pixels in the direction that it is facing and `turn(10)` will turn the spaceship. What do you think the "parameter" 10 means in turns of turning? Experiment with the parameter for the `turn()` method to see what that number means.
d. Use the Spaceship's built-in method `isAtEdge()` to determine if you at any one of the four sides or edges of the screen. If you are at the edge, turn at a random angle between 1 and 90.
e. Write a `toString()` method for your `Spaceship` to print out its `name`, `speed`, x coordinate, and y-coordinate. Note that the `Actor` class maintains status of its "x" and "y" coordinates which are accessible with its `getX()` and `getY()` methods. Since `Spaceship` is a subclass of `Actor`, it gets access to these methods for "free". See the sample output of the `toString()` method below.
e. Print the output of the `toString()` method in the `act()` using `getWorld().showText()` to see its results printed as your Spaceship moves! See example output on the next page.

Sample output



Complete the steps below to be a true SpaceCowboy:

i. Create a subclass of `World` called "`Galaxy`" (you pick the background).
ii. Create an instance of `Spaceship` and place it into your `Galaxy` using `Galaxy`'s `addObject()` method.
iii. Run your simulation!

Challenge Exercise (optional)

As your spaceship approaches the speed of light and travels off into the distance, it will appear to be physically smaller and smaller to the naked eye. Investigate the use "image scaling" methods of the Actor class cause your spaceship to become proportionately smaller and smaller as it travels through the unknown universe.

P2D. (Greenfoot) Create a subclass of `Actor` called `Airplane`. The class should have the following:

a. four instance variables; one each `int`, `double`, `boolean`, and `String`

b. a zero-argument constructor that initializes the four instance variables to known default values (unless there are more appropriate values, `ints` are usually set to 0, `doubles` to 0.0, `Strings` to `null`, and `booleans` to `false`).

c. a four-argument constructor that initializes the instance variables to values passed to the constructor.

d. a `toString()` method that returns a user-friendly output of the object's instance variables. Below is an example output of the `toString()` method for a `FlightInfo`:

```
Name: Air Canada 878,
 Speed = 528.67 mph,
Altitude = 35,000 feet
Non-stop flight: true
```

e. modify `Airplane`'s act method so that it is able to "wrap around" the screen in all possible directions when in flight: left/right, right/left, up/down, down/up, and all diagonal directions (i.e., PacMan movement). The methods `getX()`, `getY()`, `setX()`, and `setY()` might be helpful to accomplish this task.

To test the operation of your class, complete the following:

i. Create an instance of your object using its zero-argument constructor and place it into your `AirSpace` using the `AirSpace addObject()` method.

ii. Create a second instance of your object using its four-argument constructor and place it into your `Airspace` using the `Airspace addObject()` method.

P2E. (Greenfoot) In this program you'll design a `Robot` class that will model an e-commerce warehouse robot as shown in the video in class. The `Robot` class should include the following:

a. include instance variables for the speed, battery level, and direction for your `Robot`.
b. include a three-argument constructor to initialize these three instance variables.
c. a `toString()` method that returns a user-friendly output (or status) of the `Robot`'s instance variables.
d. modify your robot's `act()` method to accomplish the following:
    i. direct your `Robot` to move (in any direction you choose) at the speed specified by your robot's instance variable "`speed`", that is, `move( speed )`, not `move( 4 )`. We do this so that if you change the robot's speed later, it will move at the new specified speed, not to the constant value of "4" as in "`move( 4 )`".
    ii. decrement the battery level of your robot by one unit after every three calls to the `move()` method. If the battery level reaches 0, the `Robot` is out of fuel and should no longer move.
e. print the robot's battery level in its `act()` method using `System.out.println()`. Using `System.out.println()` will allow you to see the progression of the battery level in a separate window which can aid in the debugging process.

Now let's create a warehouse that your `Robot` will travel around in:

i. create a subclass of `World` called "`Warehouse`" (you pick the background)
ii. create an instance of `Robot` and place it into your `Warehouse` using `Warehouse's addObject()` method. Place your `Robot` in upper left hand corner of the `Warehouse`. Construct your `Robot` with a speed of 4 and battery level of 100, and a direction of your choosing for the initial direction of the `Robot`.

The following programs DO NOT need a separate class and driver files (like P2A or P2B) and are meant to be completed in BlueJ. They are stand-alone programs that are meant to review previous topics and introduce some new concepts.

P2F. Write a program that creates an array that can hold 9 double values that represent baseball batting averages for a starting baseball lineup. Once you have created the array, complete the following tasks.

a. Use a for loop to populate array with double values in the range of 0.00 to 0.500.
b. Use a second for loop to print the values in the array on one line with each value separated by two spaces.
c. Mistake in batting averages! It was found that each batting average was 0.02 less than it was supposed to be. Use a third for loop to traverse the array and add 0.02 to each batting average in the array. Be sure to actually change the value in the array as opposed to printing out the value and adding 0.02 to array value.
d. Use a fourth for loop to print the correctly adjusted batting averages on one line with each value separated by two spaces
e. Use a fifth for loop to print the array of batting averages in reverse order on one line with each value separated by two spaces.
f. Finally, find and print the maximum batting average in the array.

P2G. College application time! Write a program that creates an array of `Strings`. Fill the array with the names of at least four universities that you will be applying to. Then perform the following tasks:

a. Traverse the array using a for loop and printout the length of each university's name.

   You can get the length of a String using its `length()` method as shown below. The example below assumes that `universities` is an array consisting of `Strings`.

```
System.out.println( universities[0].length() );
```

b. Use a second for loop to traverse the array to find and store the index of the university with the fewest number of letters.

c. Print the name of the university that has the shortest length.

d. You've decided not to apply to one of the schools on your list. Remove that school from the list by setting that element in the array to "null" (you can do this in main). This can be done as shown below:

```
universities[3] = new String( "NYU" );
universities[3] = null;     // null "removes" NYU from the array
```

e. Print your list of schools again using a third for loop.

f. Print the length of each of the universities names using a fourth for loop.

g. Attempting to call the `.length()` method of a `String` (or any method for that matter), for a non-existent object can give a `NullPointerException`. Include an `if` statement in your fourth for loop to prevent a `NullPointerException` from occurring.

P2H. Let's upgrade the "Login" program from HW Set 1 so that it will allow the user to input a username/password combination for a total of three attempts. If incorrect on the fourth attempt, the program should print out the following message:

```
"You have exceeded your three attempts.
    Please try again next period."
```

Hint: Use a loop along with a counter to count the number of unsuccessful attempts. If a successful password has been entered, the program should end (i.e., the program should not ask the user for another password or remain in an endless loop).

Learning Objective Checklist

(please print and complete after you have had all HW Set 2 programs checked off)

Place a check next to those items that you have mastered

	Write a class that includes the "Big 3" (instance variables, constructors, `toString()` method).
	Write a zero-argument constructor for a class that initializes all instance variables to "0" for `int` values, 0.0 for `double` values, `false` for `booleans`, and `null` for `String` objects.
	Write a multi-argument constructor for a class that initializes all instance variables to values provided as parameters in the constructor
	Write the Java code that uses the "`new`" operator to instantiate an object in the "driver" class using the zero-argument constructor.
	Write the Java code that uses the "`new`" operator to instantiate an object in the "driver" class using the multi-argument constructor.
	Use the `+=`, `-=`, `*=`, `/=`, and `%=` operators to accumulate values in calculations.
	Create a Greenfoot project that includes a student-designed class with the "Big 3".
	Write an `act()` method for a student-designed class in Greenfoot that includes `move()` and `turn()` methods.
	Run a Greenfoot simulation that exhibits movement and provides text-based and graphical-based output for a student-designed class.

Name: _____

Date: _____