AP Computer Science
Homework Set 3
Class Methods

P3A. **(BlueJ)** Let's upgrade your `Song` class from P2A. You can make a copy of project **P2AProject** and rename it **P3AProject** in your Chapter 3 folder for this program.

Here are the specs for the upgrades:

a. Add a private instance variable "`lengthInSecs`" that stores the duration of the `Song` in seconds.

b. Upgrade the `Song`'s multi-argument constructor to properly initialize the new instance variable "`lengthInSecs`".

c. Upgrade the `Song`'s `toString()` method to include the length of the Song in the output of the `toString()` method.

d. Add "getter" and "setter" methods for the private instance variable "`lengthInSecs`". These methods will be called `getLengthInSec()` and `setLengthInSec( int newSeconds )`. In the Java vernacular, getters are also called "accessors" and setters are also called "mutators" and allow use to "change" and "view" individual instance variables.

e. Write a method `convertToMinSec()` with the following header:

```
public String convertToMinSec()
```

It will convert the length of the song into minutes and seconds and output a `String` representation of it. For example, if the name of the `Song` is `favSong` and its length is in seconds is 343, then the following line of code:

```
System.out.println( favSong.convertToMinSec() )
```

will print the following:

```
The length of the song is: 5 minutes and 43 seconds.
```

f. Upgrade the `Song`'s `toString()` method so that it calls the method `convertToMinSec()` as the final line in the `toString()` output.

g. Write a new `SongDriver` class to test the following:

i. initialize a new `Song` using the `Song`'s multi-argument constructor.

ii. print the result of calling the `Song`'s getter method `getLengthInSec()` using `System.out.println()`.

iii. change the length of the `Song` by calling `Song`'s setter method `setLengthInSec( int newSeconds )`,

iv. print the result of calling the `Song`'s getter method `getLengthInSec()` a second time using `System.out.println()`.

v. Call the `Song`'s `toString()` method to print all `Song` info.

vi. For practice, write a `getMinutes()` and `getSeconds()` and print the result of each in the driver.

P3B. (BlueJ)Let's upgrade the Student class from P2B to add the capability to store grades. You can make a copy of project P2BProject and rename it P3BProject for this program.

The specifications are listed below:

a. Add an instance variable of type array called "`grades`" that can hold five double variables for the student's grade point averages for each of his/her classes (English, Math, Science, Fine Arts, Social Sciences). The following scale will be used:

A = 4.00
A- = 3.70
B+ = 3.30
B = 3.00
B- = 2.70, etc.

b. Upgrade the zero and multi-argument constructors so that it also initializes the array of "grades" with 5 double values in the following order:

English
Math
Science
Fine Arts
Social Science

d. Add a public method, `calcGPA()`, that will calculate and return the average GPA for the student as a double value. This method should use a `for loop` to traverse the array `grades`, calculate the sum of the grades in the array and divide by the total number of grades to arrive at the GPA. Note that this method should work if the size of the array is changed (i.e., classes are added or dropped). Don't assume that the length of the array is "5".

e. Add a public getter and setter methods to set and get the `Math` GPA only. Typically, we would add getters and setters for each instance variable, but we'll just practice with one for now.

f. Upgrade the `toString()` method that will return a `String` that includes the student's name, and average GPA. Sample out is shown below:

```
Student: Mickey Mouse
Cumulative GPA: 3.97
```

Write a driver class to:

i. Create a Student `senior` using the multi-argument constructor with following mid-term grade information:

   First Name = Joe
   Last Name = Senior
   English = 3.0
   Math = 3.0
   Science = 3.5
   Fine Arts = 4.0
   Social Science = 4.0

ii. printout the student's name and average GPA using the `toString()` method,

iii. set the student's math grade to a `4.0` using the `setMath()` setter method.

iv. print the student's math grade using the `getMath()` getter method.

v. print the result of the "brain" method `calcGPA()`.

vi. printout the student's name and average GPA using the `toString()` method a second time after the grade change.

P3C. (Greenfoot) Spaceship – Flight Statistics

In this program, you'll upgrade your `Spaceship` from P2C to include two new "brain methods" that will perform flight calculations.

a. Include instance variables for the x and y coordinates for the interstellar travel destination, `xDestination` and `yDestination`, respectively.

a. Trip Distance - Write a "brain" method `calcTripDistance()` that will calculate the distance between your current location in space and a destination location in space. Method `calcTripDistance()` should have the following header:

```
public double calcTripDistance()
```

Method `calcTripDistance()` will calculate and return the distance between the `Spaceship`'s current position and the x and y coordinates of the destination as shown below:

$$d = \sqrt{\left(y_{dest} - getY()\right)^2 + \left(x_{dest} - getX()\right)^2}$$

(where all distances are in km)

b. Trip Time – Write the "brain" method `calcTripTravelTime()` that will calculate the time needed to travel the distance based on the speed of your `Spaceship`. Method `calcTripTravelTime()` should have the following header:

```
public double calcTripTravelTime()
```

You can use the average speed equation to calculate the time as shown below:

$$speed = \frac{d}{t}$$

where speed is in km/hr, d = distance in km, and t = time in hours.

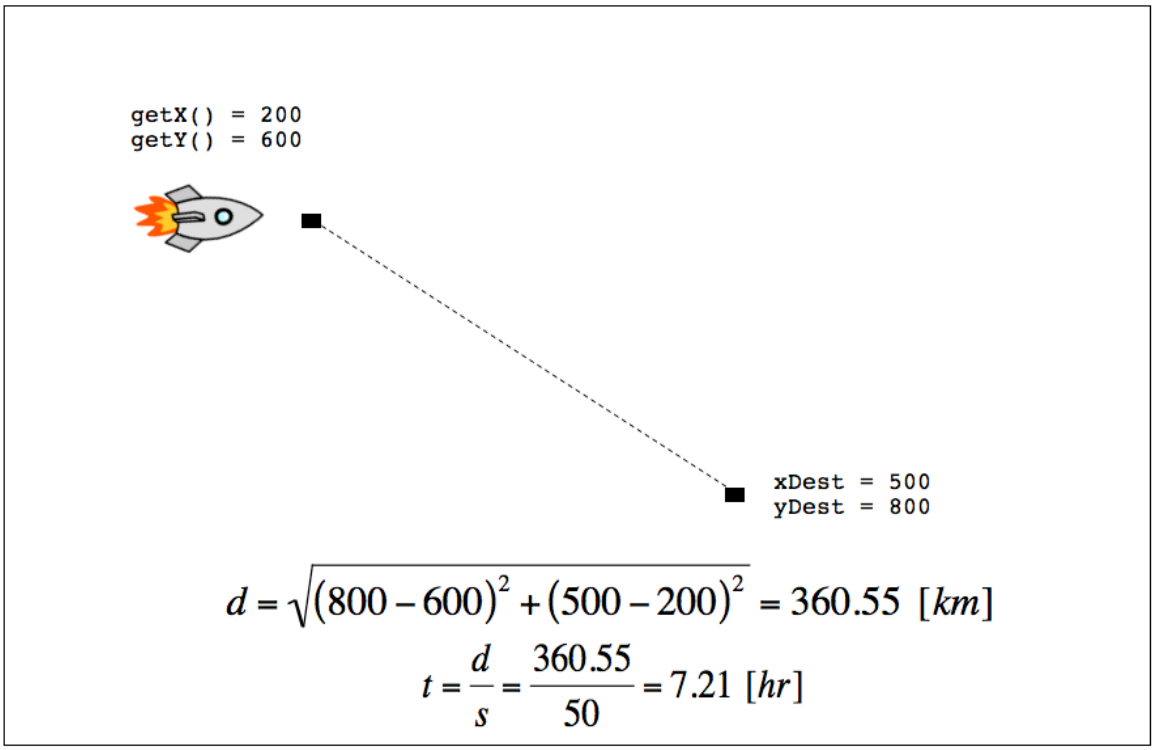
The distance "d" in the above equation should be the distance returned by the `calcTripDistance()` method.

c. Print the result of the `calcTripDistance()` and `calcTripTravelTime()` methods in `Spaceship`'s `act()` method using the following line of code:

```
getWorld().showText( "Distance to destination is: " + calcTripDistance() + " km",
                     getWorld().getWidth()/2, getWorld().getHeight() - 100 );
```

Use a similar line of code to print `calcTripTravelTime()`.

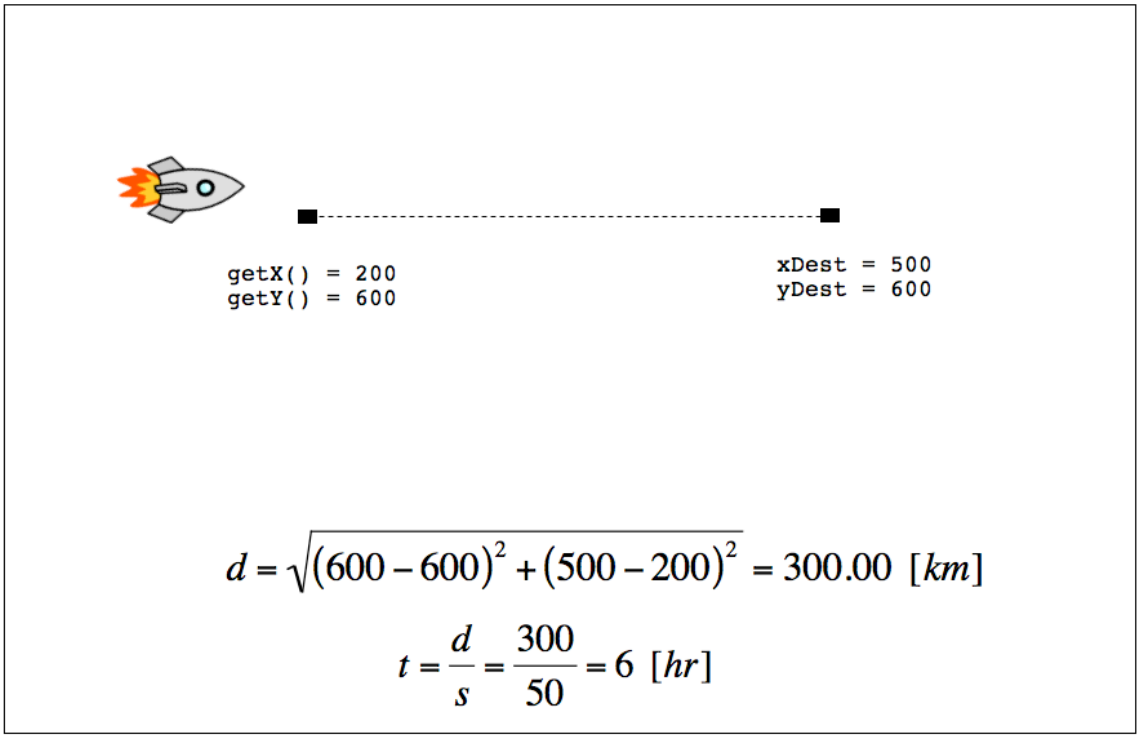
Example 1
(assuming `Spaceship` has speed of 50 [km/hr]



$$d = \sqrt{(800-600)^2 + (500-200)^2} = 360.55 \ [km]$$

$$t = \frac{d}{s} = \frac{360.55}{50} = 7.21 \ [hr]$$

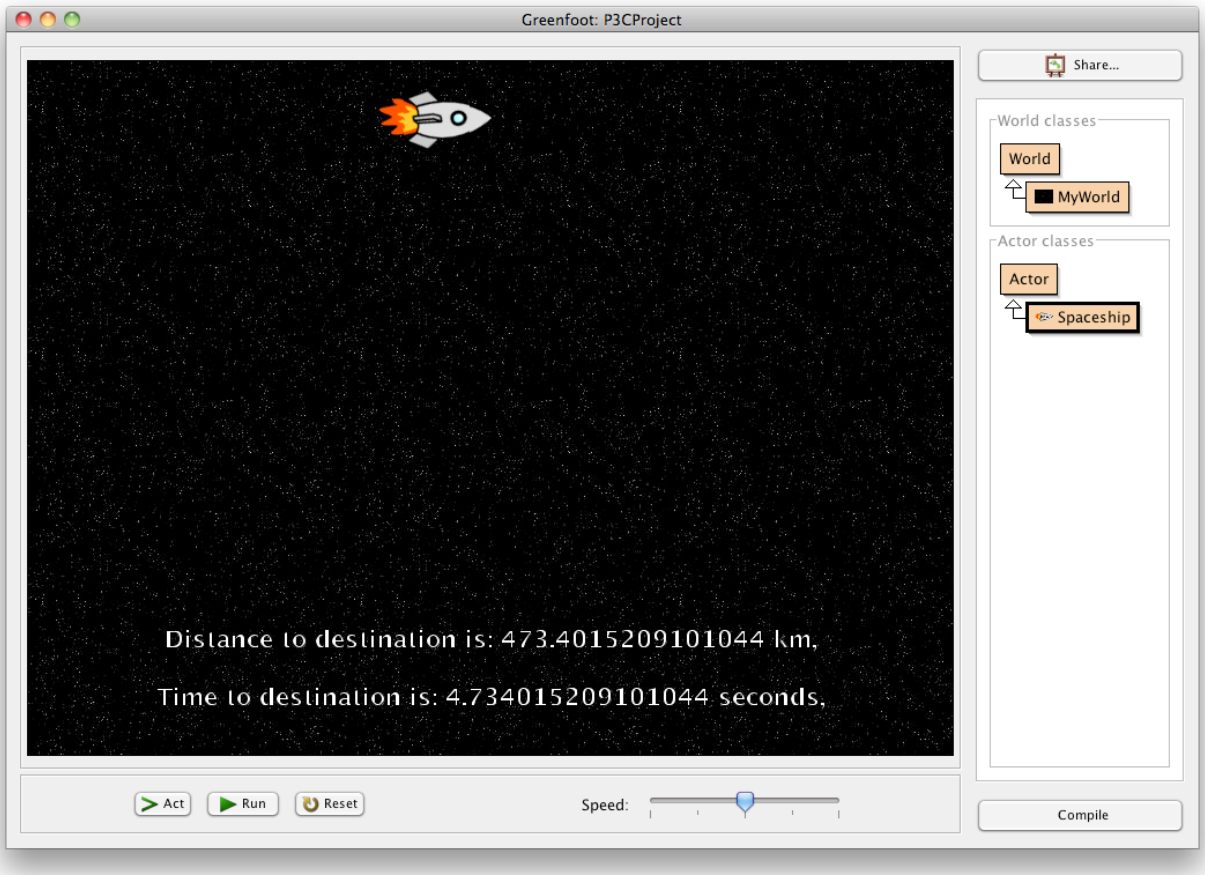
Example 2
(assuming `Spaceship` has speed of 50 [km/hr]



$$d = \sqrt{(600-600)^2 + (500-200)^2} = 300.00 \ [km]$$

$$t = \frac{d}{s} = \frac{300}{50} = 6 \ [hr]$$

Sample Output for P3C



P3D. (Greenfoot) Spaceship – Flight Training

As part of your training, you will demonstrate your piloting skills. The requirements for this training exercise are listed below. You can make a copy of project **P2CProject** and rename it **P3DProject** for this assignment.

Mission Requirements:

a. Starting at the interstellar coordinate (0,0), your `Spaceship` will traverse the known universe from left to right (one pixel at a time) until you reach the right-most boundary of space, print the x and y coordinates of your `Spaceship` each time the `Spaceship`'s `act()` method is called. Add a boolean instance variable is`MissionAccomplished` (initially set to `false`), that you will set to true when your mission is accomplished.

   The following two methods can be called from your `Spaceship` object to determine the dimensions of your `Universe` (aka `World`):

```
getWorld().getWidth();
getWorld().getHeight();
```

   If you set the dimensions of your `Universe` to 800, 600 using:

```
super(800, 600, 1);
```

   Then,
   `getWorld().getWidth();` would return 800, and
   `getWorld().getHeight();` would return 600

b. After the right boundary of space is reached, turn right, move down 1 "pixel", turn right again and continue to traverse space right-to-left until the left-most boundary of the `Universe` is reached. When you reach the leftmost boundary of space, turn left, move down 1 "pixel", and turn left again. Continue this pattern until you have reached the last possible corner of the `Universe`.

c. Once you have reached the last possible corner of the `Universe`, complete the following:
   i. Teleport to the exact center of the `Universe` by using `Spaceship`'s method `setLocation( int newX, int newY )`. Recall that you can calculate the center of the universe by using the `getWorld().getWidth()` and `getWorld().getHeight()` methods shown above.
   ii. Set the value of `isMissionAccomplished` to `true`.
   iii. Display the message "Mission Complete!" using `getWorld().showText()`.
   iv. Use the `Greenfoot.stop()` method to end the program. If you don't use this method, Greenfoot will continue to run your Spaceship's `act()` method indefinitely.

P3E. (BlueJ) APLine (AP Exam Question 2010 #2)
Write the `APLine` class as described on the handout. You will be sent a BlueJ project that includes an `APLineDriver` and an "empty" `APLine` class that you will use to complete this program.

The following programs DO NOT need a separate class and driver files. They are stand-alone programs that are meant to review previous topics (and introduce a couple of new concepts)

P3F. (BlueJ) Write a program that creates three `String` objects. Each `String` should be the name of a university to which you are applying. We will now compare the universities with the `String` method `compareTo()`. Print out the result of comparing each school to another. For example, if `s1` and `s2` are defined as follows:

```
String school1 = new String ( "Stanford" );
String school2 = new String ( "UT Austin" );
String school3 = new String ( "UCLA" );
```

We can compare the schools as follows:
```
System.out.println( school1.compareTo( school2 );
```

Answer the questions below in a comment after all of your source code:
a. What is the output of the `compareTo()` method telling us?
b. Compare to schools that have the same first letter and a different second letter. What does `compareTo()` do in this case?

P3G. (BlueJ) Write a program that searches for a given integer in an array of integers. Specifically, write a program to perform the following tasks:

a. create and print an array consisting of the 10 integers shown below. Initialize these as shown; don't use `Math.random()`.

```
-9   2    3    4    7    9    10   23   45   67
```

b. use a for loop to traverse the array and return the index number of the location of the integer you wish to find. For example, in the array above, if the number to "find" is 9, the program should return the index number "5" in a user-friendly phrase.

c. If the number is NOT found in the array, return -1.

P3H. (BlueJ) Write a program that will sum the digits of a given integer. Specifically, the program will perform the following tasks:

a. Create a class called "`MyInteger`" that consists of the "Big 3":
   i. a single integer instance variable called `num`,
   ii a one-argument constructor that will initialize the value of `num`,
   iii. a `toString()` method that will print the value of the instance variable `num` in a user-friendly format.

b. Calculate the sum of the digits of the instance variable `num` in the "brain method" `calculateSum()`. The method `calculateSum()` shall have the following header:

```
public int calculateSum()
```

   For instance, if the value of `num` is initialized to the value of 123456, the result of `calculateSum` should be: 1 + 2 + 3 + 4 + 5 + 6 = 21.

c. Write the driver class `MyIntegerDriver` that will perform the following tasks:
   i. create an instance of the `MyInteger` class using its one-argument constructor,
   ii. print the value of the instance variable `num` using the class' `toString()` method, and
   iii. call and print the result of the method `calculateSum()`.

P3I. (BlueJ) We are now going to learn about another type of array called `ArrayList` that can hold objects. One way `ArrayList`s are different from arrays is that you don't have to specify the number of elements in the `ArrayList` when you create it (as you do with regular arrays). This flexibility allows you to add and delete objects from the `ArrayList` as you wish and the `ArrayList` will take care of the resizing and re-indexing of elements itself.

a. Write a program that creates an `ArrayList` that can hold `String` objects. Populate the `ArrayList` with the names of the schools to which you are applying.

See LewTube videos for demos on how to create and use `ArrayLists` and all of its methods. The code below shows how to create an `ArrayList` of `Strings`:

```
// include these as the first lines of your program
//   to allow you to use the ArrayList object
import java.util.List;
import java.util.ArrayList;

List<String> colleges = new ArrayList<String>()
```

You can use the `ArrayList`'s "`add()`" method to add object to the `ArrayList` as shown below (include at least 5 schools in your `ArrayList`):

```
colleges.add( "UCLA" );
colleges.add( "Penn" );
```

b. Use a `for loop` to traverse the `ArrayList` and print the names of each of the schools you are applying to. We can use `ArrayList`'s .size() method to determine the number of element in the `ArrayList`. Include an if statement to test possible "`null`" entries.

c. Now use a `for-each` loop to traverse the `ArrayList` and print the names of each of the schools. See the LewTube video on how to use the `for-each` loop to traverse `ArrayLists`.

d. Now let's remove one of the schools that you are no longer applying to using the `.remove()` method. You can remove an element using the following syntax:

```
colleges.remove( 1 );
// this will remove element 1 and resize the ArrayList
```

e The `add()` method that we used in part (a) added elements to the end of the `ArrayList`. There is another version of the `add` method that allows you to add an element anywhere in the list. You can add an element anywhere in the list using the following syntax:

```
colleges.add( 1, "Santa Clara" );
// this will add element Santa Clara at element number 1 and
// resize the ArrayList
```

P3J. (BlueJ) Write a "Create new password" program that requires passwords to have a length of at least 6 characters and at least one non-alphanumeric character (+, -, *, /, or @.) Passwords that do not meet these requirements should be rejected. Use the String method `substring()` which returns `String` that is "part of" or "substring" of a given `String`. Consider the following example:

```
String password = new String( "abc*D_" );

System.out.println( password.substring( 3,4 ) );  // prints "*"
```

What do you think the arguments "3" and "4" represent?

Since the `substring` method returns a `String`, you can compare it to another `String` using `.equals()`. Below, we extract one letter from the password and determine if it is equal to "*".

```
String oneLetter = new String( password.substring( 3,4 ) );
oneLetter.equals( "*" );    // result is true
```

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 3 programs checked off)

Place a check next to those items that you have mastered

	Write getter and setter methods for instance variables in a class.
	Write "brain" methods for a class that perform algorithmic processing for a class.
	Use the `substring()` method of the `String` class to "extract" a letter or letters from a `String`.
	Use the `+=`, `-=`, `*=`, `/=`, and `%=` operators to accumulate values in calculations.
	Create an instance of class `ArrayList` of any given object type.
	Populate an `ArrayList` with objects using the `ArrayList`'s `add()` method.
	Get number of elements in an `ArrayList` using the `ArrayList`'s "`size()`" method.
	Traverse an `ArrayList` using a `for` loop.
	Traverse an `ArrayList` using a `for-each` loop.
	Traverse an array using a `for` loop.
	Traverse an array using a `for-each` loop.