# Objective

To practice using and working with abstract classes, inheritance, arrays and ArrayLists.

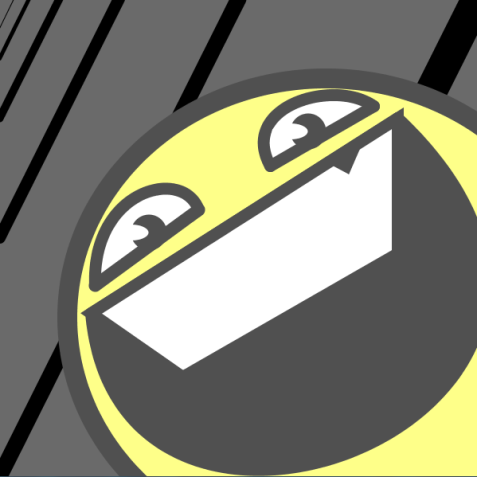
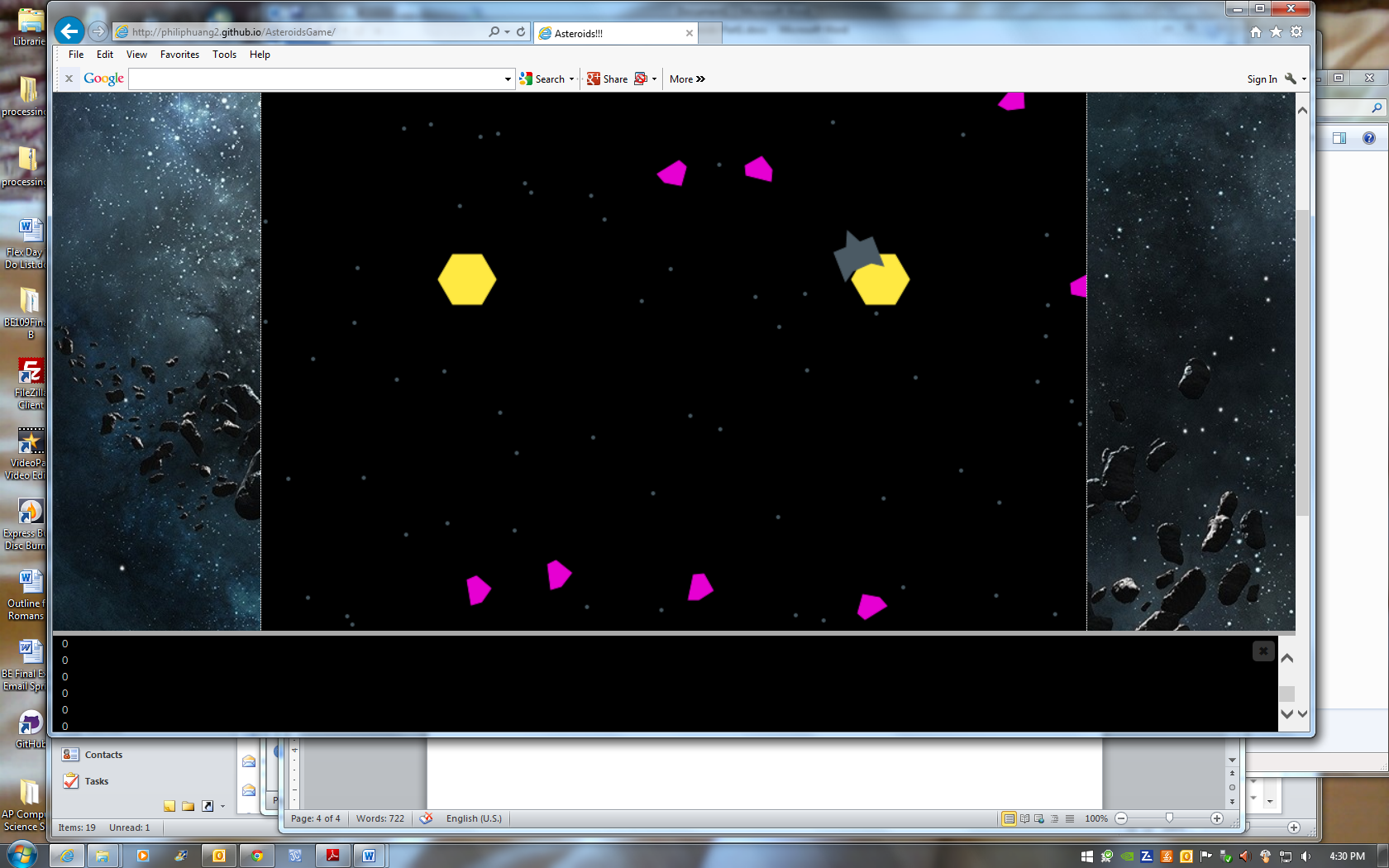
## Topics: abstract classes, inheritance, arrays, ArrayLists, traversing and removing from ArrayLists

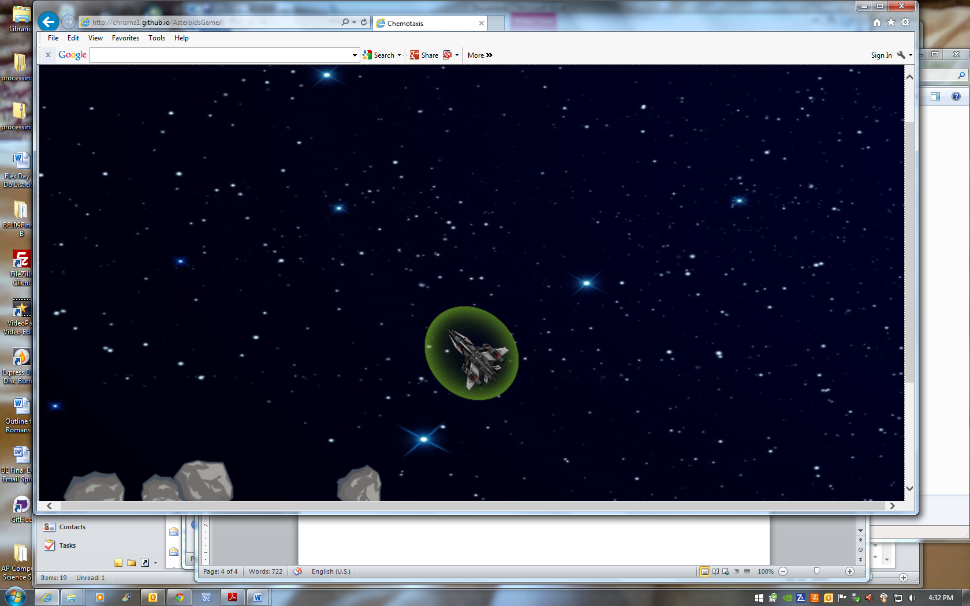
# Instructions

For this assignment we will replicate the old video game Asteroids. You will write a program that includes a space ship that is controlled by keys from the keyboard. You will need to write the following classes: SpaceShip, Star, Asteroid and Bullet. The SpaceShip class will extend the Floater class, which is an abstract class which represents all things that float in space (Asteroids, SpaceShips, Bullets, etc.) The Floater class has the show() abstract method. (note: some code is supplied in comments in the Floater class that might be useful for the subclass show() method. )

An abstract class is an “incomplete” class, which has abstract methods and complete methods. When you extend an abstract class you must write the methods that “completes” the class. Suggested (not required steps) for completing this assignment are listed in 4 parts below. You will be graded on the stage completed.

# Examples





# Hints

Sometimes

# Hints

Sometimes

# Suggested steps for completing Part 1: (all complete is 25% of your grade)

1. Uncomment the “extends Floater” on the SpaceShip class header line of code.
2. Complete the abstract Floater methods in the SpaceShip class.
3. Write the SpaceShip constructor: initialize the private data fields in the Floater class via calling the public set methods provided for you in the Floater class.
4. At the top of the program, declare a variable of type SpaceShip
5. Initialize the SpaceShip as a new instance of the class
6. In the draw( ) method, call the SpaceShip’s show( ) method
7. When you are comfortable with the appearance of SpaceShip, add a public void keyPressed( ) function to your program
8. Write code in the keyPressed( ) function that allows you to control the space ship with the keyboard. You must include the ability to rotate left, accelerate, and enter “hyperspace.” “Hyperspace” just uses the accelerate method to drastically increase the speed of the SpaceShip.
9. Add code to the draw( ) function to move( ) the SpaceShip.
10. Add a Star class that holds an x, y location of each Star. Each Star is NOT a Floater object because Stars will remain a stable background image. That is, they do not float! You will need a show( ), but no move( ) method for your Star class.
11. Create a constructor for you Star class which gives the Star a random x, y location on the screen.
12. Create an ArrayList of Star objects at the top of your file. Loop and create (instantiate) 100 stars or more in your setup( ) method. Then, loop and call each Star’s show( ) method from the draw( ) method.

# Suggested steps for completing Part 2: (all completed is 25% of your grade)

Now that we have a functioning space ship, we’ll add some asteroids to our game. We’ll write an asteroids class that extends Floater.

1. Write an Asteroids class that extends SpaceShip (an Asteroid behaves like a floating SpaceShip and will need to have access to the show( ) method of the SpaceShip class. You will need to write a constructor.
2. Add an int field variable of the Asteroid class. It will hold the speed of rotation for each asteroid. Make sure that this is initialized to have an equal probability of being positive or negative. Also make sure to declare appropriately.
3. “Override” the move( ) method of the SpaceShip class in the Asteroid class that also rotates each Asteroid by its own speed. You may then call the move( ) method of the SpaceShip class to move the Asteroid. Careful that you call the super class’s move method and not move(), which is infinite recursion.
4. Add a single asteroid to your project. Start by calling the Asteroid’s show( ) method. Make sure you can see it and are satisfied with its shape before going to the next step.
5. Add the code that moves and rotates your Asteroid.
6. Modify the project so that you have an array of Asteroids.

# Suggested steps for completing Part 3: (all completed is 25% of your grade)

An array probably isn’t the best way to keep track of a bunch of asteroids. Arrays have a fixed size. You can’t easily add or remove asteroids from an array. A better choice might be an ArrayList. The ArrayList class has a number of useful member methods:

* boolean add(Object x)
* void add(int index, Object element)
* Object get(int index)
* Object remove(int index)
* Object set(int index, Object x)
* int size()

1. Modify your asteroids game to use an ArrayList instead of an array of asteroids.
2. Now we’ll modify the program so that when our space ship strikes an asteroid, the asteroid is removed from the ArrayList. Everytime an asteroid moves, find the distance between that asteroid and the ship. Use processing’s dist( ) function to find the distance between the asteroid and the ship. If the distance is less than 20 remove the asteroid from the ArrayList. Otherwise, move and rotate the asteroid normally.

# Suggested steps for completing Part 4: (all completed is 25% of your grade)

To finish the Asteroids game, we need to write a new class that represents Bullets. We will store the Bullets in an ArrayList much like we did with the Asteroids. Once we can shoot the bullets and destroy Asteroids, we will have a working game. Your Asteroids game doesn’t have to look like mine or any other. Feel free to modify it in any way you wish, provided you have: 3 Floater subclasses, a Star class, a SpaceShip controlled by the keyboard (as specified earlier) and arrays/ArrayLists of Stars, Bullets, and Asteroids. The SpaceShip, Asteroids and Bullets must move in your sketch.

1. Write a Bullet class that extends Floater.
2. Write a constructor that takes one ship argument. Bullet (SpaceShip theShip) This constructor will:
   1. Initialize myCenterX, myCenterY of the bullet to be the same as the ship.
   2. Initialize myPointDirection of the bullet to be the same as myPointDirection of the ship
   3. Convert myPointDirection to radians with the following code:

double dRadians = myPointDirection \* (Math.PI/180);

* 1. Initialize myDirectionX as 5 \* Math.cos(dRadians) + the myDirectionX of the ship
  2. Initialize myDirectionY as 5 \* Math.sin(dRadians) + the myDirectionY of the ship

1. Write the show( ) method in the Bullet class so that you use circular bullets.
2. Add one bullet to your program and draw it on the screen. Make sure you can see it before continuing.
3. Now, move the bullet.
4. Create an ArrayList of Bullets. The list should be empty to start with. Everytime you press the key to “shoot” (in my case the ‘ ‘ key), add a new Bullet to the ArrayList. Modify the program with loops that draw and move all the bullets in the ArrayList.
5. One way to check for collisions between the bullets and the Asteroids is to write a loop within a loop. Everytime you move one asteroid you will need:
   1. A loop that goes through all the bullets to see if there is a collision between the bullet and the asteroid.
   2. If there is a collision remove both the asteroid and the bullet from their ArrayLists
6. Alternatively, to check for collisions, you might be able to use processing’s get() to check for collisions with colors of the sketch.

# Hints

Most of the mathematical code is written for you in the: accelerate, rotate and commented out show method code in the AsteroidsStartCode.pde file. Also, review your notes or textbook on how to add objects to an ArrayList and traverse an ArrayList of objects.

Some useful Processing keyCode values are: LEFT, RIGHT, UP, DOWN.

# Challenge

If you have extra time and want some challenge, add an animation of “rockets” that appear from the back of the ship when you accelerate. You can also add some unusual shapes for Asteroids/Ship/Bullets that are not circles, etc.