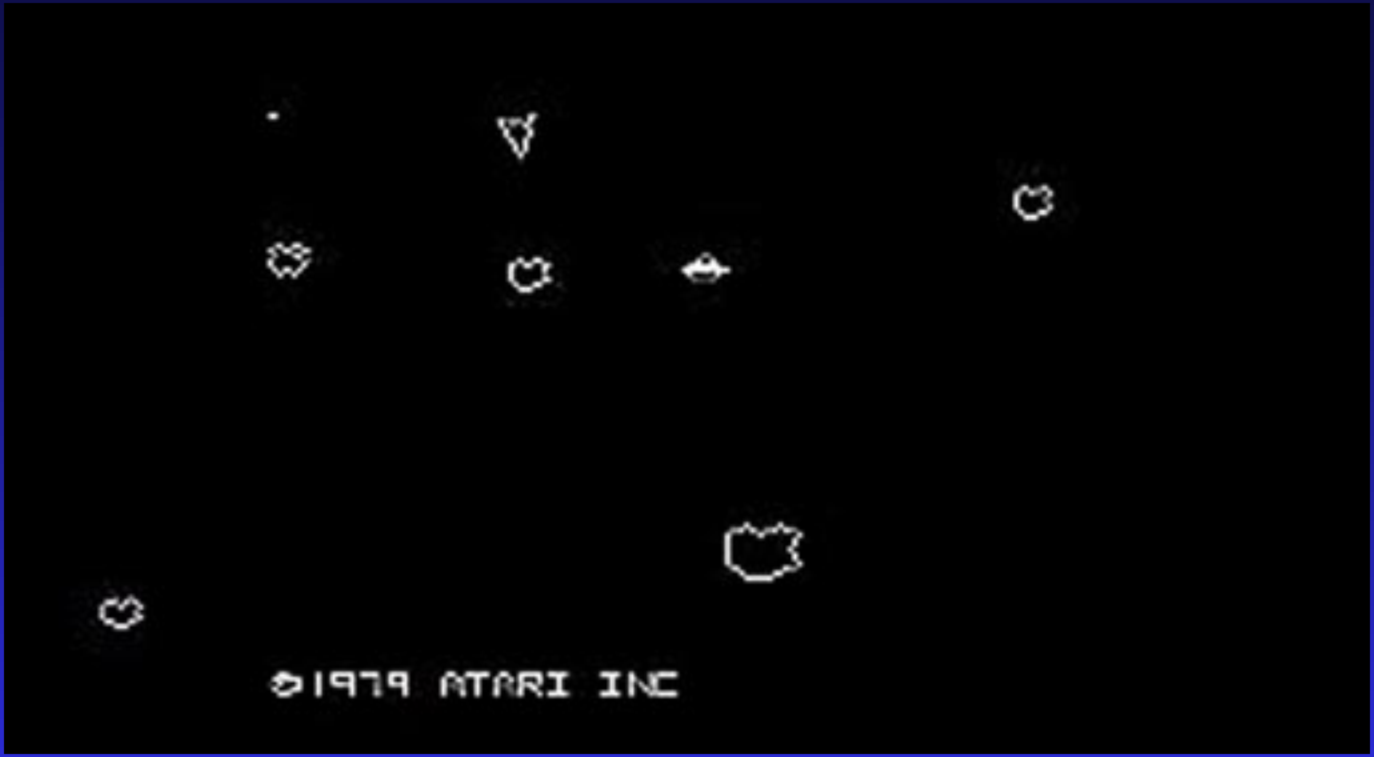
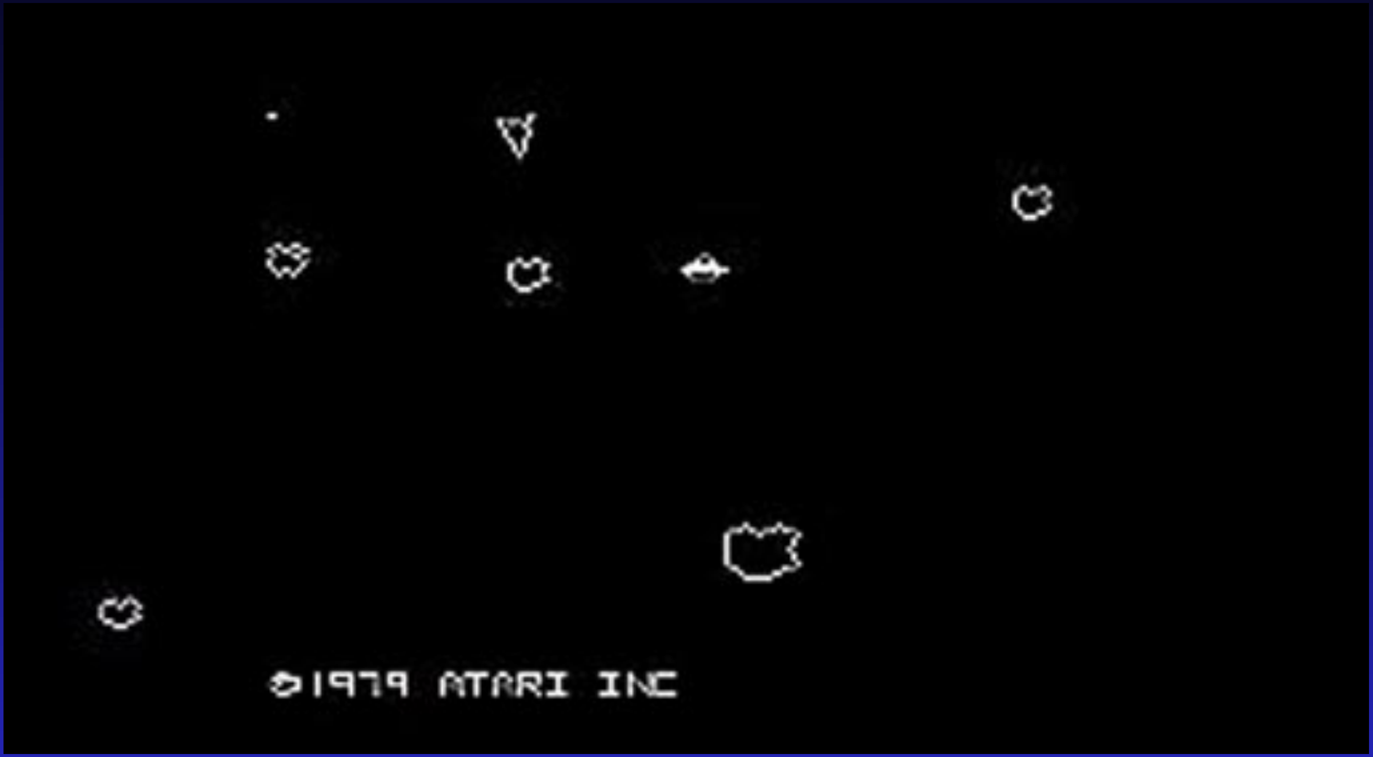
Unit_9b_AsteroidsProject Part 1



- We'll recreate the Asteroids arcade originally created in 1979!

Designing an Asteroids game



- What objects do we need to model?
- What do they have in common?

Designing an **`Floater`** class

- A *super* or *base* class for all objects that float in space
- Once we have a **`Floater`** class, we can **`extend`** it to make other *sub* or *derived* classes

1. spaceships
2. asteroids
3. bullets
4. UFOs (optional)

Designing an `Floater` class

- What do all objects that float in space do?

Designing an `Floater` class

- What do all objects that float in space do?
  1. Move
  2. Turn (Rotate)
  3. Get drawn or show

Designing an `Floater` class

- What do all objects that float in space do?
  1. Move
  2. Turn (Rotate)
  3. Get drawn or show
- What do all objects that float in space have?

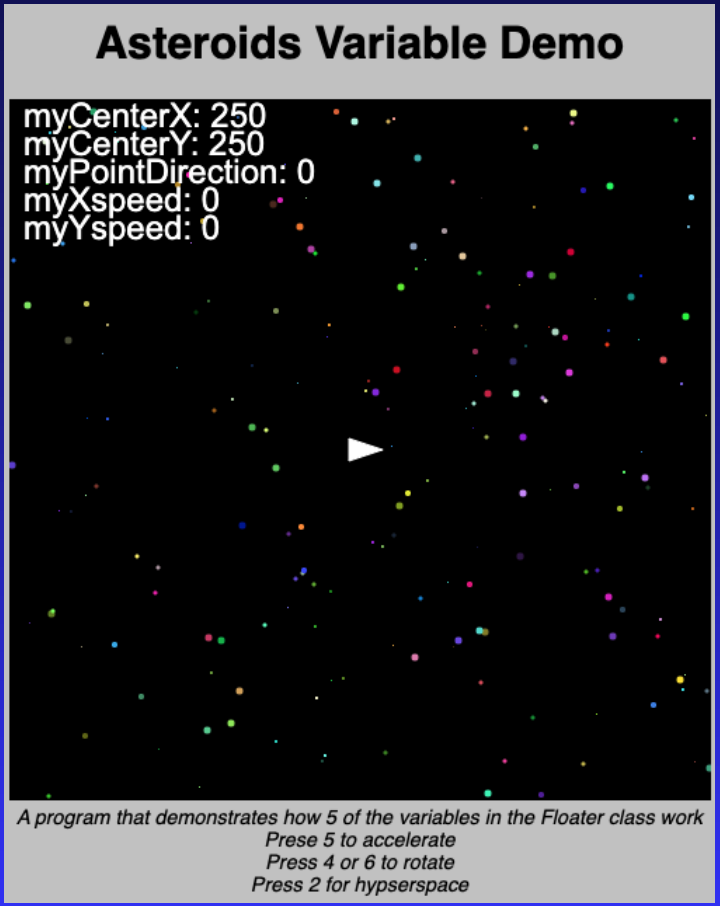
Designing an `Floater` class

- What do all objects that float in space do?
  1. Move
  2. Turn (Rotate)
  3. Get drawn or show
- What do all objects that float in space have?
  1. A number of corners
  2. X and Y position
  3. One direction (that they point)
  4. And a different direction (that they move)

A sample Spaceship program

- You may find the program below helpful in understanding how the **`Floater`** class works

  AsteroidsVariableDemo



Asteroids Variable Demo

```
myCenterX: 250
myCenterY: 250
myPointDirection: 0
myXspeed: 0
myYspeed: 0
```

A program that demonstrates how 5 of the variables in the Floater class work
Prese 5 to accelerate
Press 4 or 6 to rotate
Press 2 for hypserspace

The 9 `protected` member variables in `Floater`

```
protected double myCenterX, myCenterY;
//holds center coordinates


protected double myXspeed, myYspeed;
//holds the speed of travel
//in the x and y directions



protected double myPointDirection;
//holds current direction the floater is pointing
//in degrees
```

`protected` member variables in `Floater`

```
protected int corners;
//the number of corners, a triangular
//floater has 3

protected int[] xCorners;
protected int[] yCorners;
//The coordinates of the corners, with center of
//object at (0,0) and myPointDirection=0 (right)

protected int myColor;
```

`extend`ing the **`Floater`** class

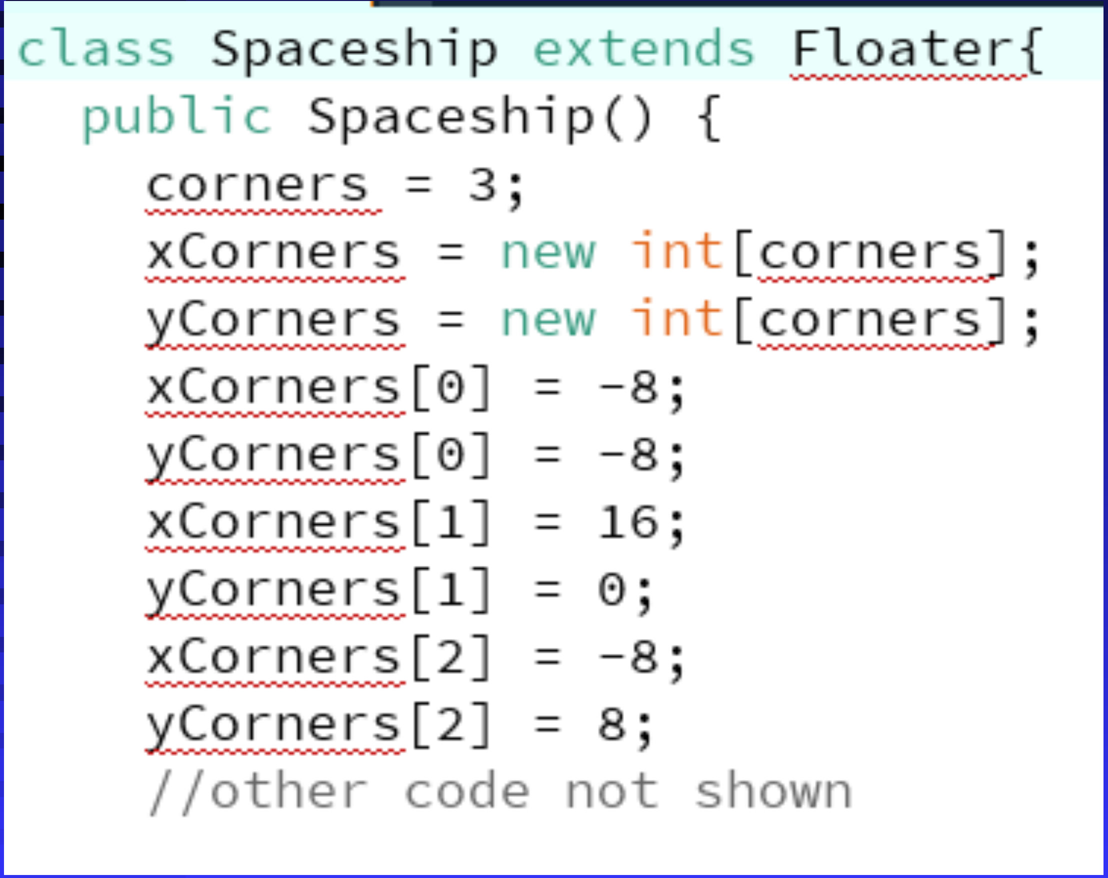
- What functions would you *need* to write to **`extend Floater`**?

`extending` the `Floater` class

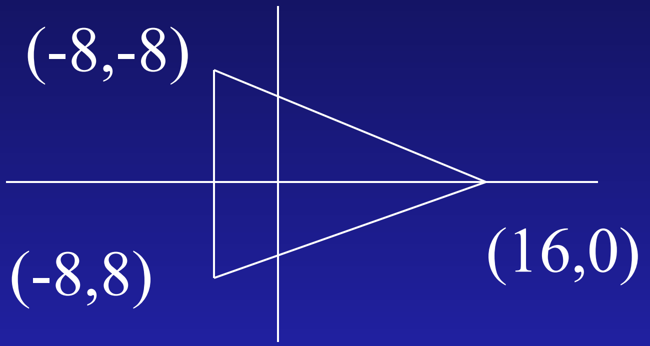
- What functions would you *need* to write to **`extend Floater?`**
- You would write a constructor
- You would also write “getter” and “setter” functions as necessary

Constructing a **Spaceship**

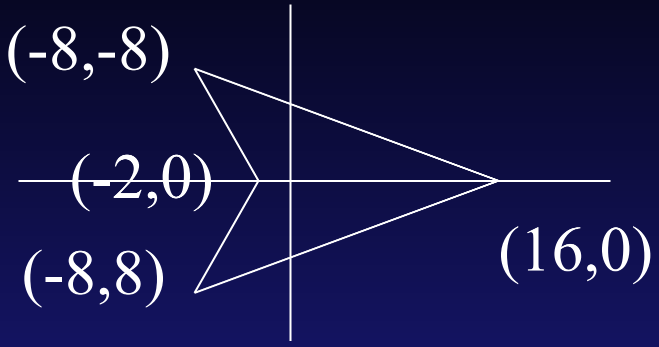
- Initialize the 9 inherited **protected** variables: **corners**, **xCorners**, **yCorners**, **myColor**, **myCenterX**, **myCenterY**, **myXspeed**, **myYspeed** and **myPointDirection**



```
class Spaceship extends Floater{
  public Spaceship() {
    corners = 3;
    xCorners = new int[corners];
    yCorners = new int[corners];
    xCorners[0] = -8;
    yCorners[0] = -8;
    xCorners[1] = 16;
    yCorners[1] = 0;
    xCorners[2] = -8;
    yCorners[2] = 8;
    //other code not shown
```

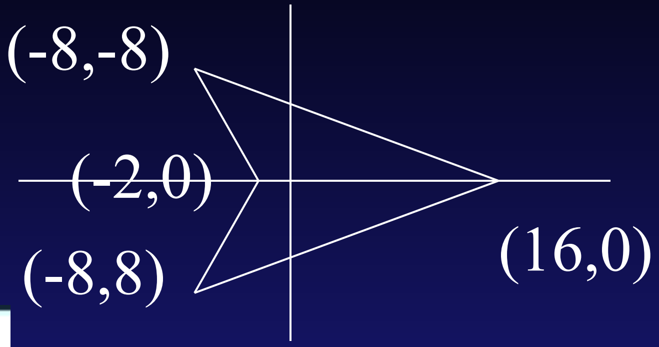


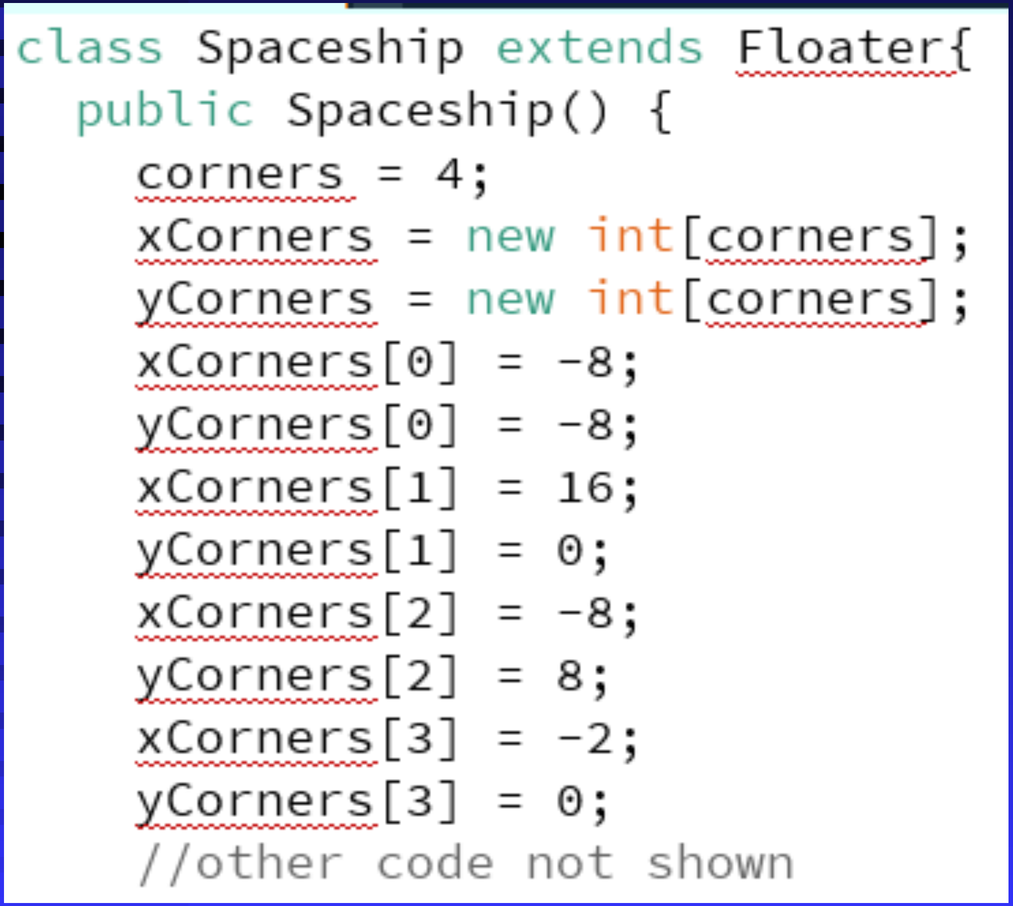
Constructing a *slightly fancier* `Spaceship`



- How would the constructor of my slightly fancier `Spaceship` be different?

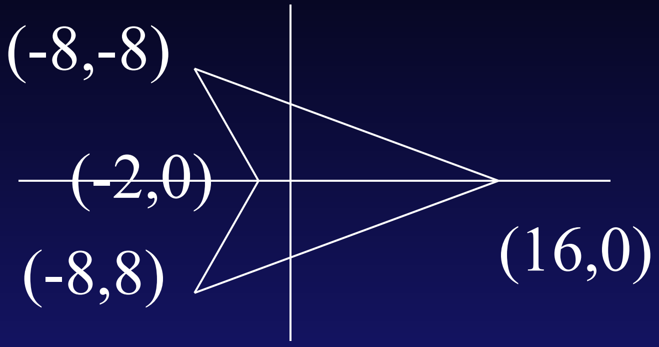
Constructing a *slightly fancier* **Spaceship**

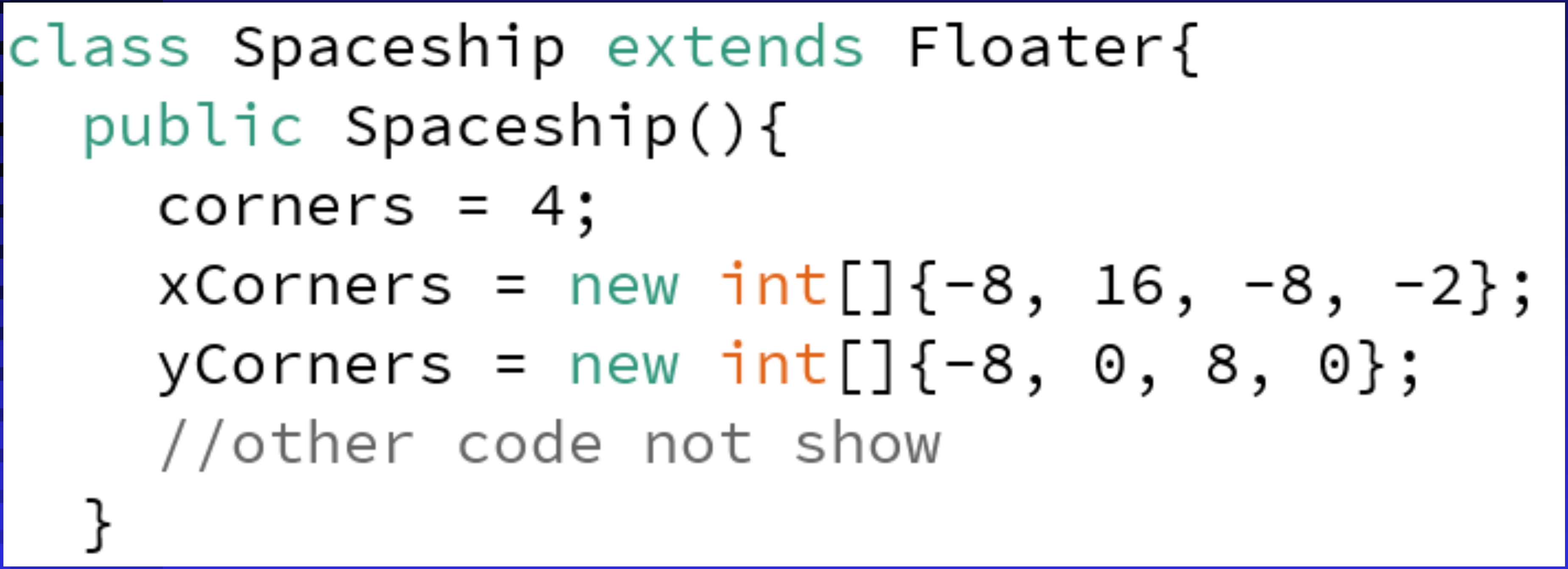




```
class Spaceship extends Floater{
  public Spaceship() {
    corners = 4;
    xCorners = new int[corners];
    yCorners = new int[corners];
    xCorners[0] = -8;
    yCorners[0] = -8;
    xCorners[1] = 16;
    yCorners[1] = 0;
    xCorners[2] = -8;
    yCorners[2] = 8;
    xCorners[3] = -2;
    yCorners[3] = 0;
    //other code not shown
```

A different way

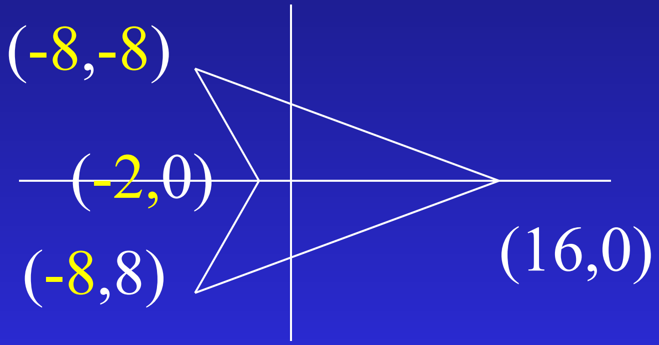




```
class Spaceship extends Floater{
  public Spaceship(){
    corners = 4;
    xCorners = new int[]{-8, 16, -8, -2};
    yCorners = new int[]{-8, 0, 8, 0};
    //other code not show
  }
```

Important: half of your coordinates should be negative!

- The **`Floater`** class was written with the assumption that (0,0) is the center of rotation and the ship is pointing to the RIGHT (0 °)
- I recommend sketching your design on graph paper



Hyperspace

- The assignment requires a hyperspace feature
- There is no requirement for any fancy visual effects, hyperspace just needs to *stop the ship*, and give it a new *random position and direction*

```
Spaceship bob = new Spaceship();
//other code (setup, draw) not shown

public void keyPressed(){
  if(key == 'h'){
     bob.??;
  }
}
```

Is this OK?

- Since we need to stop the ship, we'll need to set **`myXspeed`** and **`myYspeed`** to zero

```
Spaceship bob = new Spaceship();

public void keyPressed()
{
  if(key == 'h')
  {
     bob.myXspeed = 0;
     //other Java code not shown
  }
}
```

NO! **`myXspeed`** is **`protected`**

- So, what is the right way to set **`myXspeed`** to zero instead of this ?

```
Spaceship bob = new Spaceship();

public void keyPressed()
{
  if(key == 'h')
  {
     bob.myXspeed = 0;
     //other Java code not shown
  }
}
```

You could create a setter like **`setXspeed()`**

- Since it is a member of the **`Spaceship`** class it has access to **`myXspeed`**

```
class Spaceship extends Floater
{
   public SpaceShip() {
      //construtor code not shown
   }
   public void setXspeed(double x) {
      myXspeed = x;
    }
   //other functions not shown
}
```

Or you could create a **`hyperspace()`**

```
class Spaceship extends Floater
{
   public SpaceShip() {
      //construtor code not shown
   }
   public void hyperspace() {
      //code not shown
   }
   //other functions not shown
}
```

Either way is fine

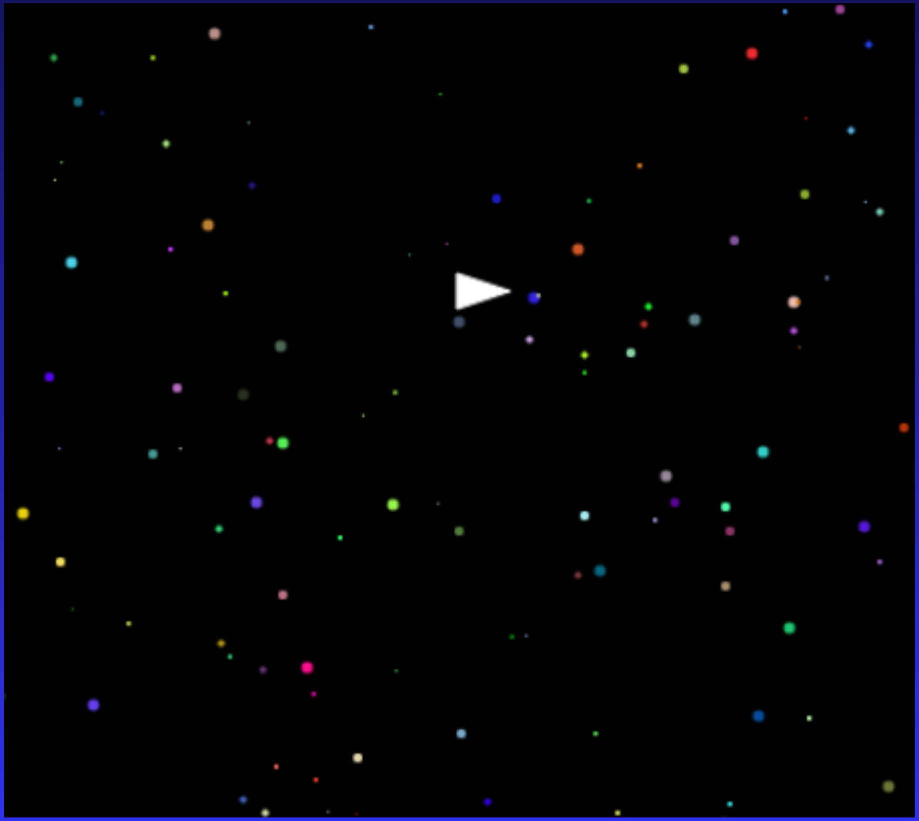
- Then use the setter function **`setXspeed()`** in **`keyPressed`**

```
Spaceship bob = new Spaceship();

public void keyPressed()
{
  if(key == 'h')
  {
     bob.setXspeed(0); //OK!
     //or
     bob.hyperspace(); //OK!
     //other Java code not shown
  }
}
```

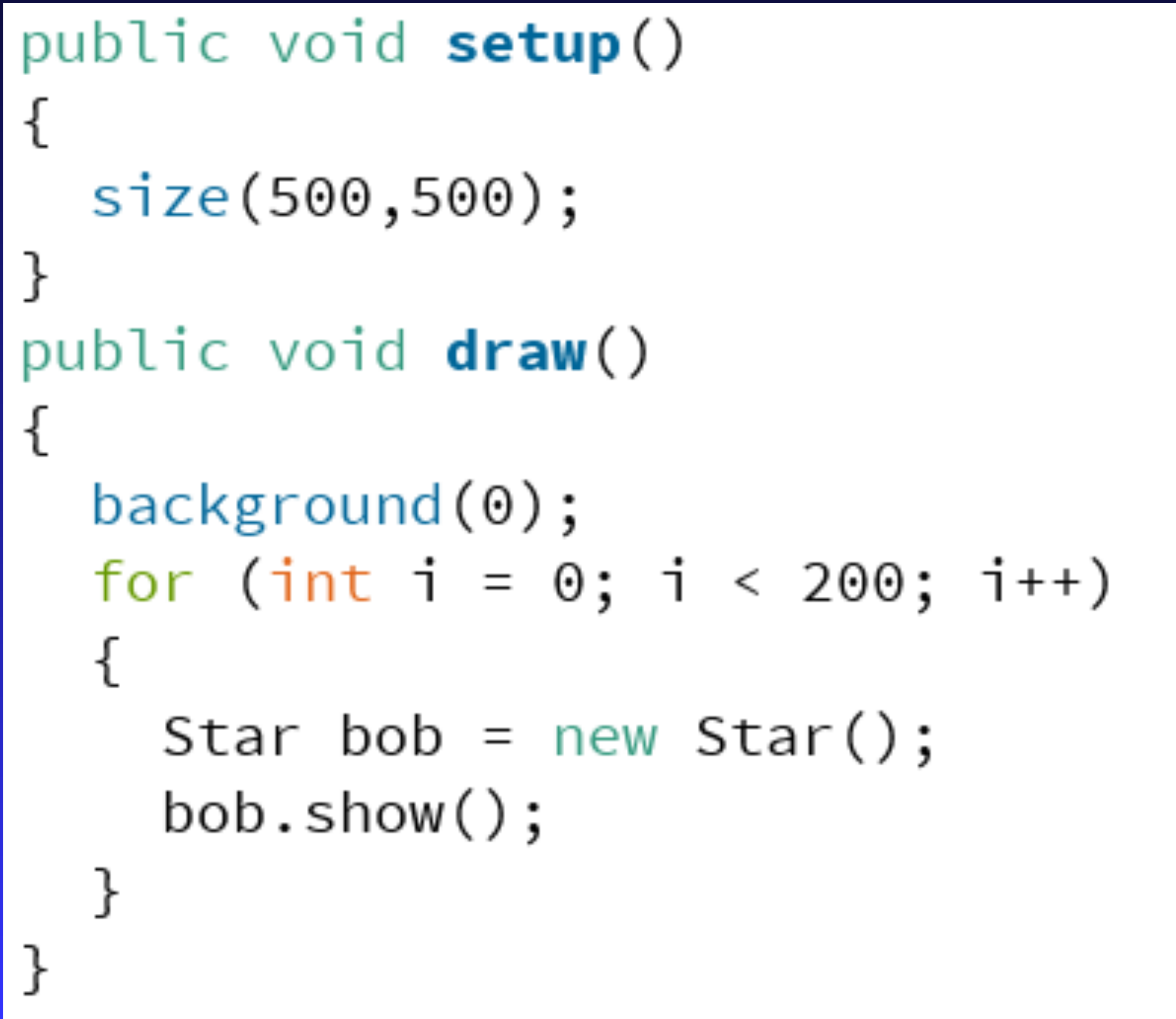
Asteroids: `Star` class

- Your program should have a **`Star`** class that creates the background of the game



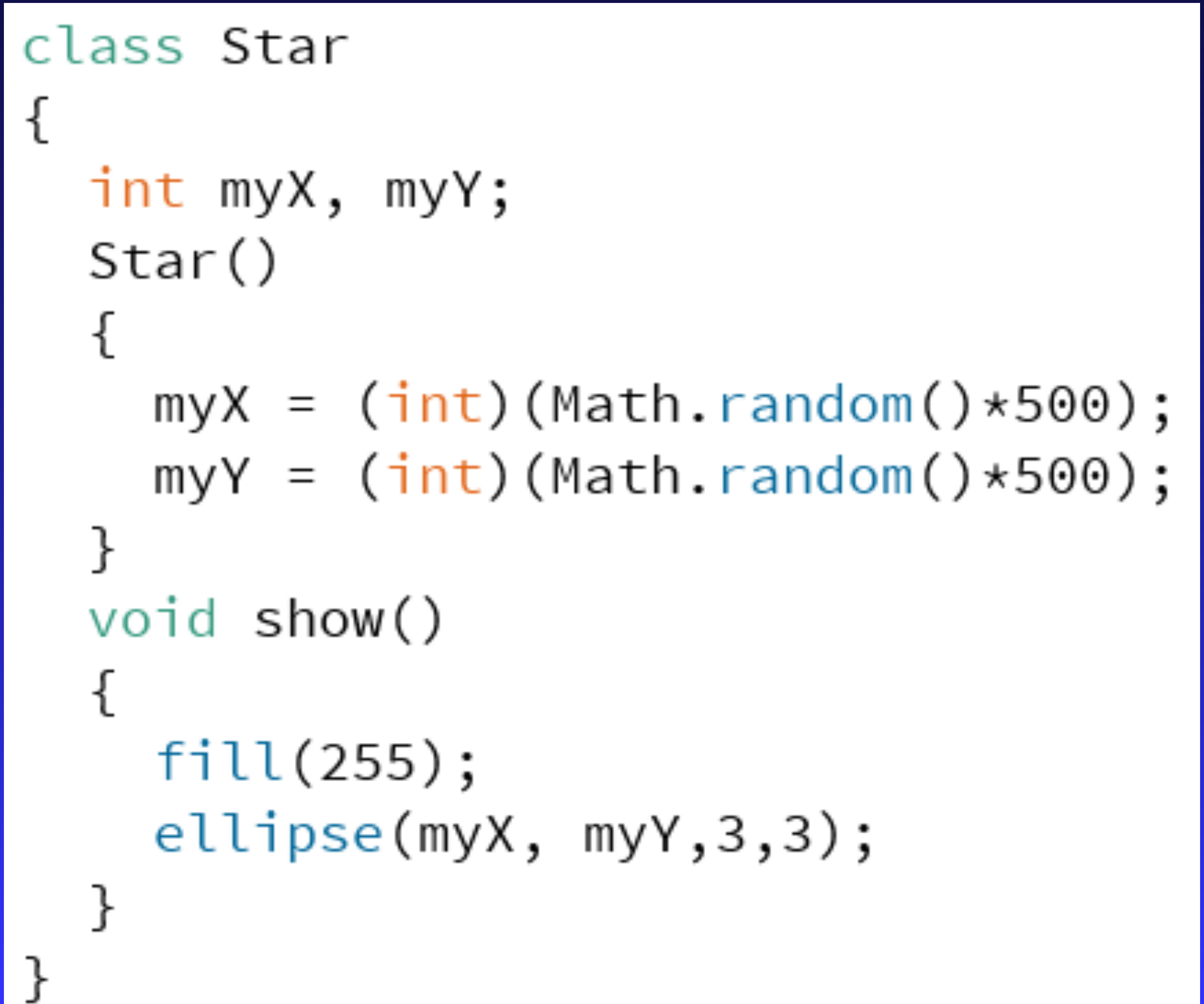
Creating an Array of Stars

- There is a common mistake in this code, do you see it?



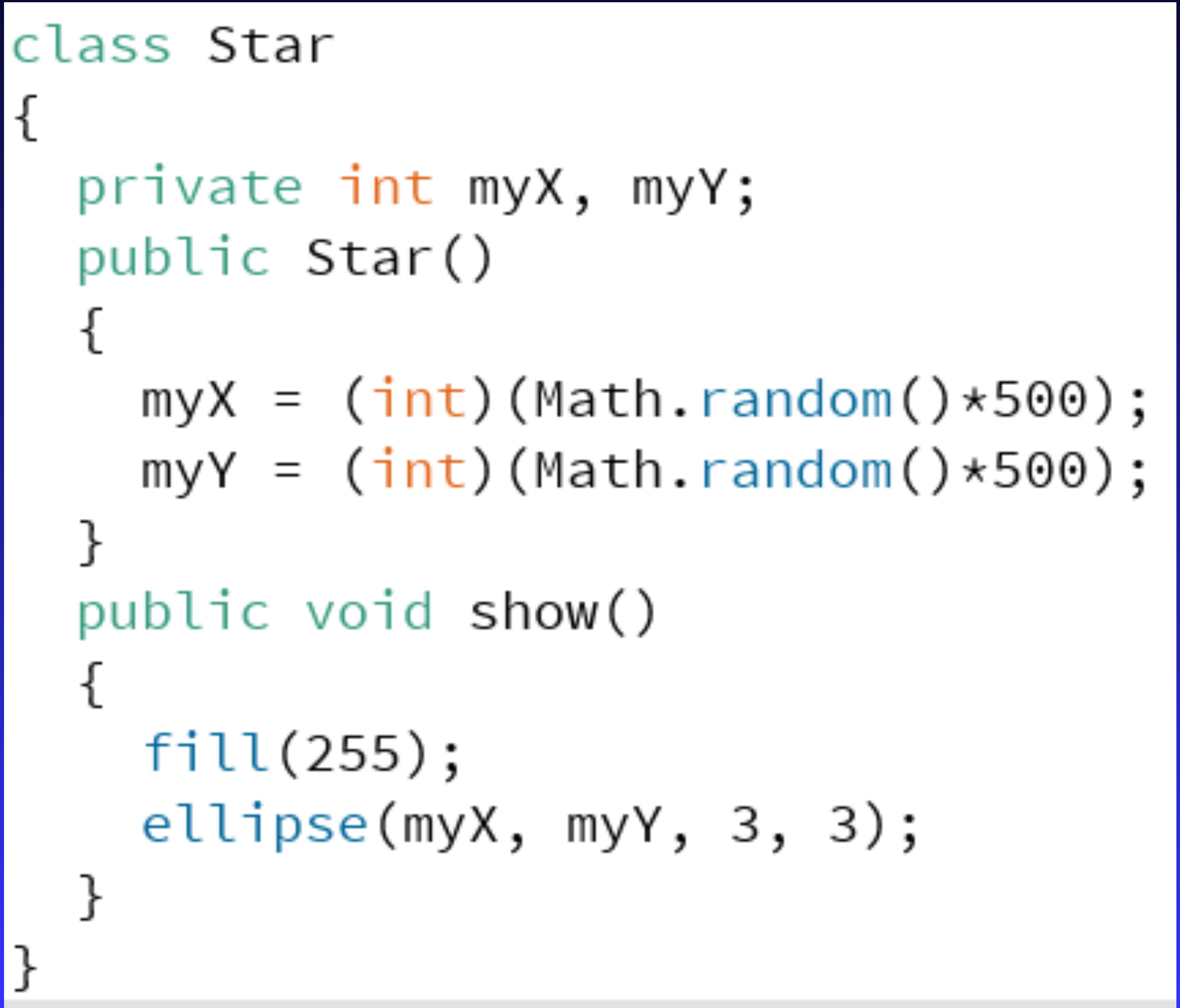
```
public void setup()
{
  size(500,500);
}
public void draw()
{
  background(0);
  for (int i = 0; i < 200; i++)
  {
    Star bob = new Star();
    bob.show();
  }
}
```

If you wrote this **`Star`** class on the AP, they would deduct points. Why?



```
class Star
{
  int myX, myY;
  Star()
  {
    myX = (int)(Math.random()*500);
    myY = (int)(Math.random()*500);
  }
  void show()
  {
    fill(255);
    ellipse(myX, myY,3,3);
  }
}
```

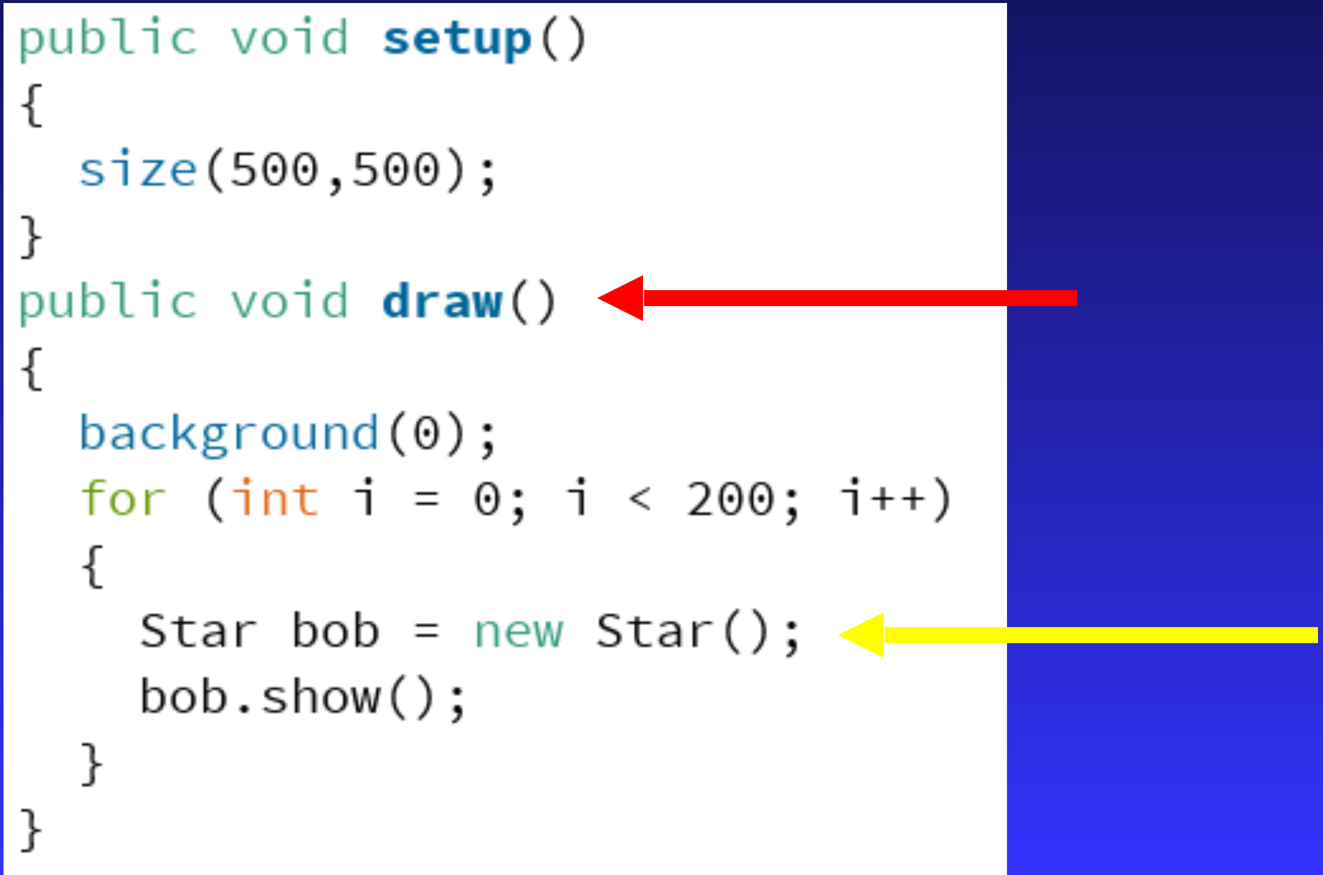
Like every class on the AP exam the **`Star`** class needs to be encapsulated



```
class Star
{
  private int myX, myY;
  public Star()
  {
    myX = (int)(Math.random()*500);
    myY = (int)(Math.random()*500);
  }
  public void show()
  {
    fill(255);
    ellipse(myX, myY, 3, 3);
  }
}
```

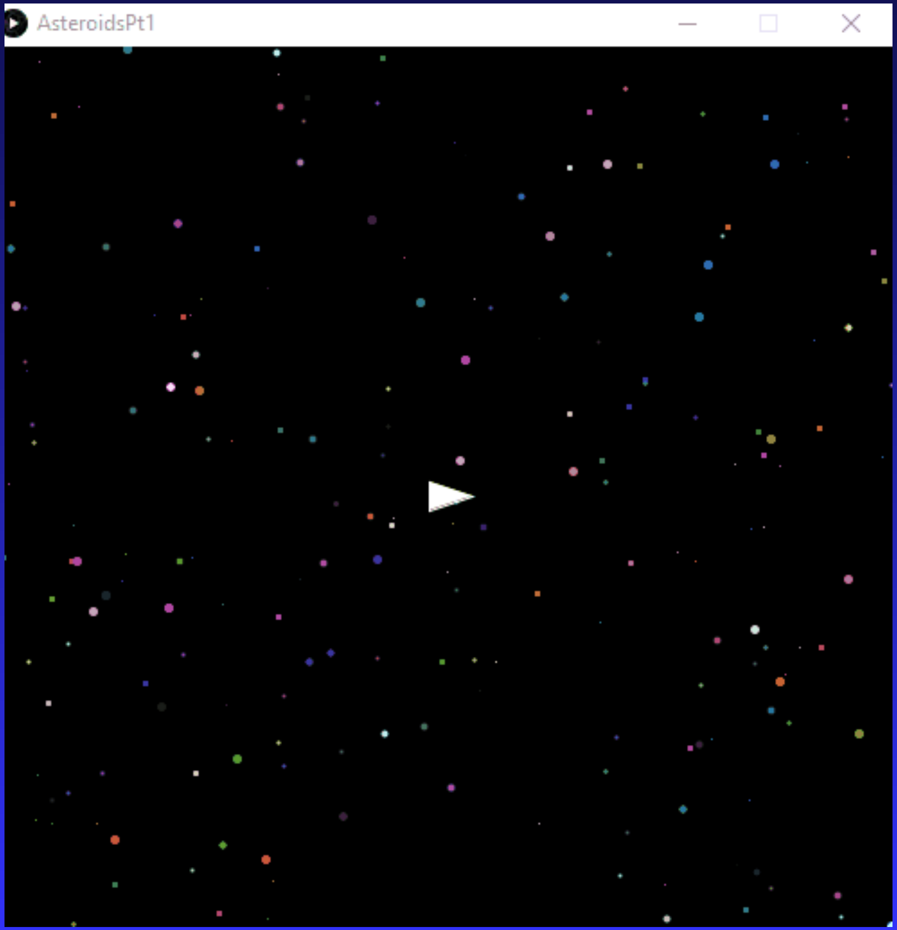
Creating an Array of Stars

- Every time we draw the screen we are creating 200 **new** Stars in new positions



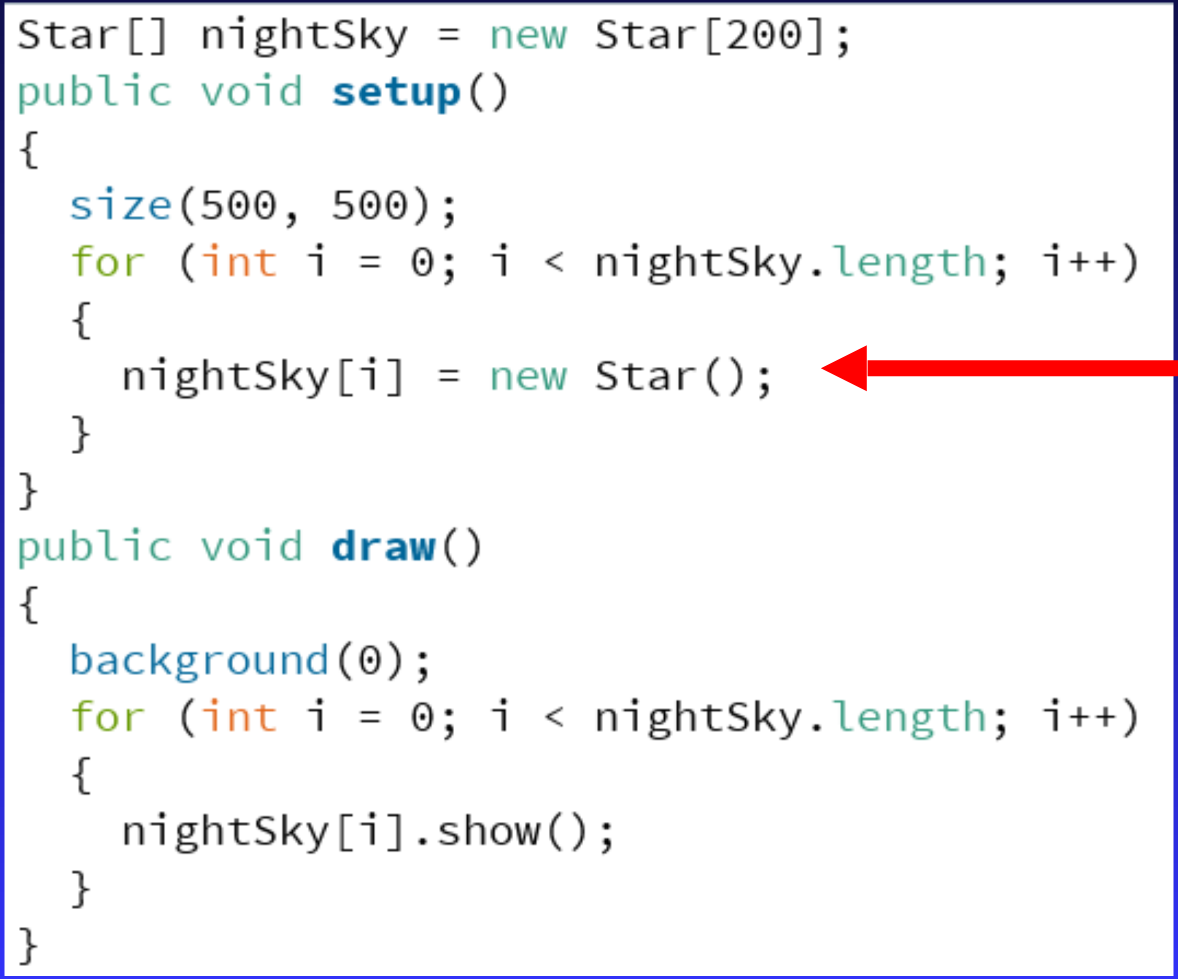
```
public void setup()
{
  size(500,500);
}
public void draw()
{
  background(0);
  for (int i = 0; i < 200; i++)
  {
    Star bob = new Star();
    bob.show();
  }
}
```

This is what creating new Stars in new positions everytime you draw the screen looks like (why?)



Creating an Array of Stars

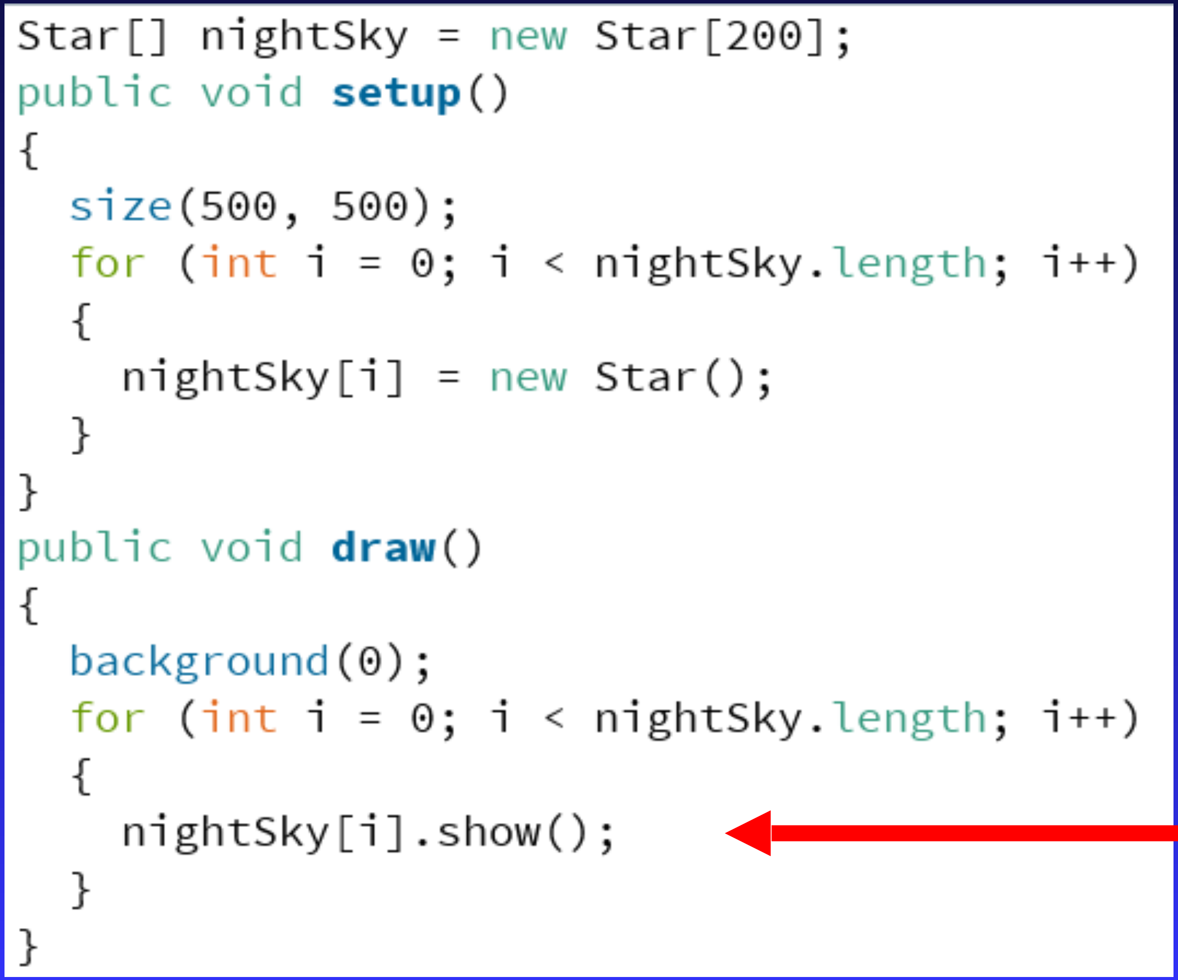
- A better idea would be to create 200 **`new`** Stars *once* in **`setup()`** using an Array



```
Star[] nightSky = new Star[200];
public void setup()
{
  size(500, 500);
  for (int i = 0; i < nightSky.length; i++)
  {
    nightSky[i] = new Star();
  }
}
public void draw()
{
  background(0);
  for (int i = 0; i < nightSky.length; i++)
  {
    nightSky[i].show();
  }
}
```

Creating an Array of Stars

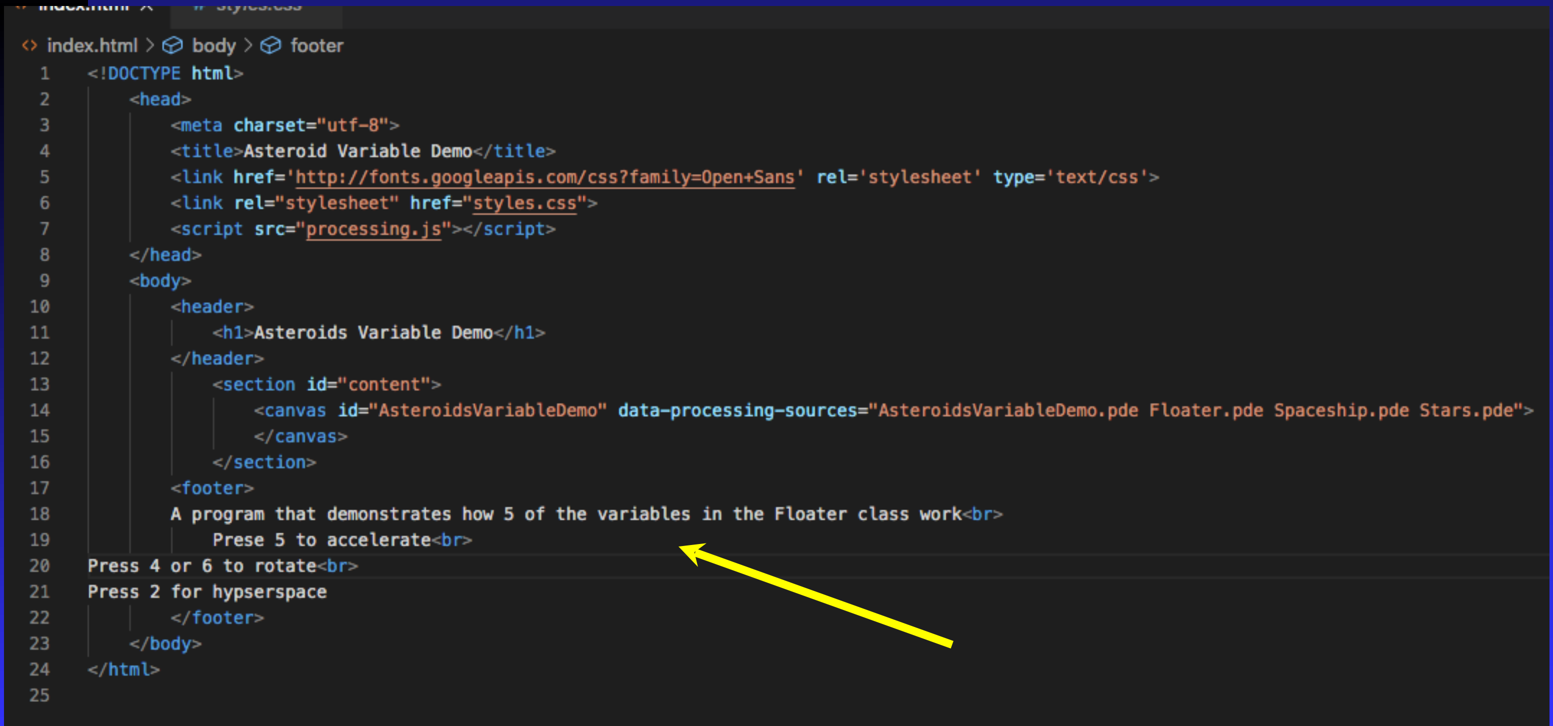
- Then we would **`show()`** the same 200 Stars everytime we **`draw()`** the screen



```
Star[] nightSky = new Star[200];
public void setup()
{
  size(500, 500);
  for (int i = 0; i < nightSky.length; i++)
  {
    nightSky[i] = new Star();
  }
}
public void draw()
{
  background(0);
  for (int i = 0; i < nightSky.length; i++)
  {
    nightSky[i].show();
  }
}
```

Adding instructions in index.html

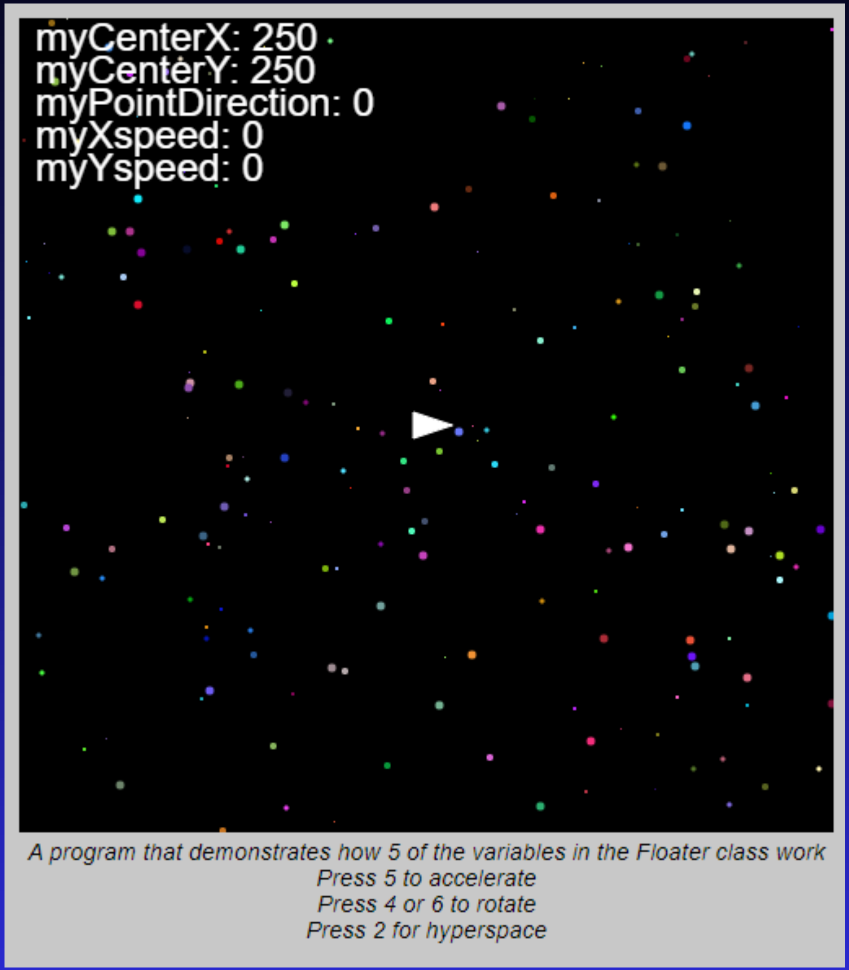
- A requirement for this assignment is to put the instructions on how to control the spaceship in the webpage
- One place would be the footer of index.html



```
<!DOCTYPE html>
    <head>
        <meta charset="utf-8">
        <title>Asteroid Variable Demo</title>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans' rel='stylesheet' type='text/css'>
        <link rel="stylesheet" href="styles.css">
        <script src="processing.js"></script>
    </head>
    <body>
        <header>
            <h1>Asteroids Variable Demo</h1>
        </header>
            <section id="content">
                <canvas id="AsteroidsVariableDemo" data-processing-sources="AsteroidsVariableDemo.pde Floater.pde Spaceship.pde Stars.pde">
                </canvas>
            </section>
        <footer>
        A program that demonstrates how 5 of the variables in the Floater class work<br>
            Prese 5 to accelerate<br>
Press 4 or 6 to rotate<br>
Press 2 for hypserspace
        </footer>
    </body>
</html>
```

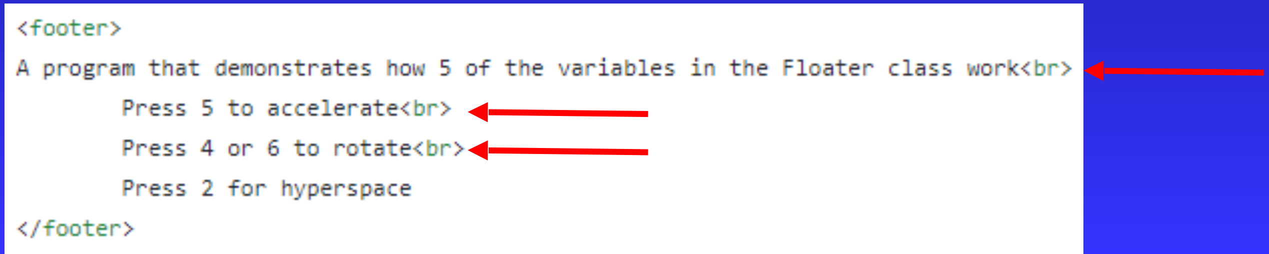
Adding instructions in the html

- `<br>` is the html code (called a *tag*) to insert a line break



```
myCenterX: 250
myCenterY: 250
myPointDirection: 0
myXspeed: 0
myYspeed: 0
```

A program that demonstrates how 5 of the variables in the Floater class work
Press 5 to accelerate
Press 4 or 6 to rotate
Press 2 for hyperspace



```
<footer>
A program that demonstrates how 5 of the variables in the Floater class work<br>
        Press 5 to accelerate<br>
        Press 4 or 6 to rotate<br>
        Press 2 for hyperspace
</footer>
```