10: Classes

Special note

What are classes?

Object Oriented Programming

- Object-oriented Programming (OOP) is probably the most influential idea in programming since programming itself.
- (Pause here for effect.)
- Almost every application you use was influenced by OOP.
    - Desktop applications.
    - Video games.
    - Mobile applications.
    - The turtle application you've been using is built upon OOP.
- This concept, without a doubt, is critical to modern software development.
- Very rarely will you use software that isn't built with OOP.

What is Object-Oriented Programming.

- Object-oriented programming is a style of programming which focuses on objects.
- An object is represented by three things:
  - It's name.
  - What it knows.
  - What it can do.

Instaquiz!

What are the three things which represent an object?

Instaquiz!

What are the three things which represent an object?

- It's name.
- What it knows about.
- What it can do.

Example: Cars

Let's talk about cars as if they are a **class**.

- The name of the class is "Car".
- What do cars know about? In other words, what is something nearly all cars have?
- What do cars do?

This is an open conversation. Please answer these questions out loud.

Cars know.

- A car has an engine.
- A car has four tires.
- A car has a color.
- A car has one or more seats.
- A car has many other features not mentioned on this list.

Cars do.

- A car can drive.
- A car can put the windows up or down.
- A car can run the air conditioner or not.
- A car can run the radio or not.

We divide knowledge and ability.

Here's another quiz. Which of these represent what a car knows or what a car does?

- A car has zero or more cupholders.
- A car can open the trunk.
- A car has an automatic transmission.
- A car can shift into "Reverse".

Special Rule: Don't add what isn't there.

If a car doesn't know something or can't do something, then we shouldn't add that to our code. Classify each of these statements as "part of a car" or "not part of a car".

- A car can apply the breaks.
- A car can fly into space.
- A car has headlights.
- A car has an invisibility cloak.

Terminology

- What a car knows is called the “member variables”.
- What a car can do is called the “methods”.
- Each car in existence has its own unique properties, including member variables and methods.

Classes and Objects

- Objects are what we've been discussing this entire time.
- Classes are what defines an object.
- **Turtle** (with the uppercase "T") is a **class**.
- **turtle** (with the lowercase "t") is an **object**.
- Look at some of your old turtle drawing code and see if you can find an example of a class and an object.

A special note about Java.

- We always name our classes with the first letter uppercase. For example, “Turtle” and “String”.
- We always name our objects with the first letter lowercase. For example, “turtle”.
- Java doesn’t care if you follow this advice, but many programmers will like you more if you follow this rule.
  - In the movie “Pirates of the Caribbean” there’s a good line: **“They aren’t rules; They’re guidelines.”**
  - This is true of many things in Java.

Instaquiz

(I enjoy writing these instaquizzes to see if you are paying addition.)
Which of these is the class and which is an object?

- flower
- Flower

Instaquiz

(I enjoy writing these instaquizzes to see if you are paying addition.)
Which of these is the class and which is an object?

- flower
- Flower

Our next project will involve a flower generation class.

Writing our first class

New Project: FlowerArt

- Create a new project called “FlowerArt”.
- Import the TurtleLog.jar library.
- In this project, we will do basic shapes, like yeterday, but with a new twist.
- We are drawing flowers. They are pretty and hopefully you will see why.
- They also require lots of math. If you aren’t sure about the math, follow along.

Just like last time.

Modify the line containing “public class FlowerArt {” to look like this.

```
public class FlowerArt extends Sandbox
```

You will have to fix your code’s imports to make the error on “Sandbox” go away.

Start Programming.

In your main method, add one line of code. It will look like this.

```
public static void main(String[] args) {
    launch(args);
}
```

Place a turtle

Create a new method called “draw”. **Don’t add any code yet, but make the method.**

```
@Override
public void draw() {
}
```

Writing a class

- Right click on the package “flowerart” in the Project view.
- Select “New. . . ” and then “Java Class”.
- Name this new class “Flower”.
- Make sure that the package says “flowerart”. If you don’t, this may not work.

My code.

My code looks something like this.

```
package flowerart;

public class adf {

}
```

Everything we type in this file will between those two curly brackets. If you don't see the package line, raise your hand. We will delete the file and try again.

Flowers are drawn with Turtles.

In every flower there's a turtle which made that flower. Add a turtle to the code.

```
public class Flower {
    private Turtle turtle;
}
```

Add a Constructor

The constructor is called when we need to build the member variables of the object.

```
public class Flower {
    private Turtle turtle;

    public Flower() {
        turtle = new Turtle();
    }
}
```

From here on out, we will be posting methods that go under the constructor, but before that final "}". Make sure that you have a "{" for every "}" in your code. There should always be one "{" at the top and "}" at the very bottom.

Get the Turtle.

It's the turtle that does the drawing. We need to get the turtle to ask it what it drew.

```
public Turtle get() {
    return turtle;
}
```

This method uses a return type of "Turtle". What is the method name?

Drawing Lines.

This method will draw a line defined by `length`, then turn defined by `angle`, and do this `seg` times. Here, `seg` is short for “segments”.

```
public void lines(int seg, double length, double angle) {
    for (int i = 0; i < seg; i++) {
        turtle.forward(length);
        turtle.left(angle);
    }
}
```

Where have we seen this code before? If you said “polygons”, then you’d be right.

Let's test: FlowerArt.java

Return to your `draw` method in the `FlowerArt` class. (Java programmers are often bouncing back and forth between files. We will do our best to show you which file to edit.)

```
@Override
public void draw() {
    Flower flower = new Flower();
    add(flower.get());

    flower.lines(100, 4, 2);
}
```

Run the code. You should get a curved line on the screen. Explore a bit with the code. Change the 2 for the angle to a 10. What happens?

Make Polygons: Flower.java

Return to Flower.java. Add a new method called “polygon”:

```
public void polygon(int sides, double length) {
    double angle = 360 / sides;
    lines(sides, length, angle);
}
```

This code will call the `lines` method to correctly draw any polygon we want. Notice that each method will call a previous method. We are going to be reusing code heavily to save us from typing so much. (Tip: The file to edit will always be in the title bar.)

Test our polygon maker: FlowerArt.java.

In FlowerArt.java, remove the line with a call to “flower.lines. . . ”. Replace it with this:

```
flower.polygon(8, 50);
```

This should draw an octagon on the screen. You can replace the “8” with any number greater than 2 to see that polygon.

Make Arcs: Flower.java

An arc is circle that didn't finish the job of making a complete circle. Here's the formula for the arc length of a circle using degrees.

$$arc = \frac{\pi r \theta}{180}$$

Here, r means the radius of a circle. We need this formula in order to make our flowers.

Make Arcs: Flower.java

Return to Flower.java and make a method called “arc”:

```
public void arc(double radius, double angle) {
    double arc_length = Math.PI * radius * angle / 180.0;
    int segments = (int) (arc_length / 4) + 1;

    double step_length = arc_length / segments;
    double step_angle = angle / segments;

    lines(segments, step_length, step_angle);
}
```

Test Flower Arcs: FlowerArt.java.

Return to FlowerArt.java. Remove the line with “flower.polygon” and replace it with this:

```
flower.arc(100, 270);
```

You should see a circle that stops about three-fourths of the way to completion.

Make Petals: Flower.java

To make a petal, we make an arc, do a turn, then make another arc, then do the same turn.

```
public void petal(double radius, double angle) {
    arc(radius, angle);
    turtle.left(180-angle);
    arc(radius, angle);
    turtle.left(180-angle);
}
```

Notice that we call “arc” twice. This is code reuse and a big part of object-oriented programming!

Test Petals: FlowerArt.java.

Return to FlowerArt.java. Remove the line with “flower.arc” and replace it with this:

```
flower.petal(100, 135);
```

A petal can have any angle greater than 30 and less than 180 degrees. Test out various values for the angle. It’s the second parameter. The first parameter impacts the size. The second impacts the shape. I think that 75 is the perfect angle. Test this code again with that value.

Draw Flowers. Flower.java

Finally, we can draw flowers. Add this method to Flower.java named "draw". The first parameter sets the color, the second tells how many petals to draw, the third sets the size, and the fourth sets the shape.

```
public void draw(Color color, double petals,
                 double radius, double angle) {
    turtle.setColor(color);
    for (int i = 0; i < petals; i++) {
        petal(radius, angle);
        turtle.left(360.0 / petals);
    }
}
```

Test the Flower art. FlowerArt.java

Return to Flower.java. Remove the line that has “flower.petal” and replace it with this:

```
flower.draw(Color.GREEN, 8, 100, 75);
```

- This flower will be green.
- This flower will have 8 petals.
- The size of each petal will be 100 units.
- The radius of each petal will be 75 units.

Experiment with the FlowerArt.java file.

- Play with the 4 parameters until you have a pretty flower.
  - The first parameter is the color.
  - The second is the number of petals.
  - The third is the size.
  - The fourth is the shape of the petal. Find the best value of the shape.

More Flowers to Try

```
flower.draw(Color.GREEN, 8, 100, 75);
flower.draw(Color.BLUE, 3, 100, 135);
flower.draw(Color.GREEN, 20, 100, 50);
flower.draw(Color.GREEN, 6, 100, 60);
```

Field of Flowers

Field of Flowers

Let’s great a giant field of flowers. With our code tucked away into a class, we can reuse it however we wish.

Create a new class named “RandomFlower”.

- Right click on the package “flowerart” in the Project view.
- Select “New. . . ” and then “Java Class”.
- Name this new class “RandomFlower”.
- Make sure that the package says “flowerart”. If you don’t, this may not work.

Modify the Class

Because a RandomFlower is a Flower, we need to modify the class. Find the line with "public class RandomFlower {". Change it to this:

```
public class RandomFlower extends Flower {
```

This handy trick allows us to have every method in Flower without having to rewrite the code. This trick is one that I use often. If you want to create a specialized version of a class, extend it.

Create a `draw` method.

This method will draw one flower randomly on the screen. Here, I create an array of my favorite seven colors. Make sure that you have exactly 7 elements. You may change the colors.

```
public void draw() {
    Random rng = new Random();
    Color[] colors = {Color.RED, Color.ORANGE,
                      Color.YELLOW, Color.GREEN,
                      Color.BLUE, Color.INDIGO,
                      Color.VIOLET};
```

Part 2 of 3: Generate a bunch of random values.

We are generating a bunch of random values. We randomly generate a size, number of petals, radius, and a color. We also generate a random x and y where we will display the center of the flower.

```
double size = rng.nextDouble() * 100 + 50;
int petals = rng.nextInt(5) + 8;
double radius = rng.nextDouble() * 75 + 15;
Color color = colors[ rng.nextInt(7) ];

double x = rng.nextInt(600);
double y = rng.nextInt(600);
```

Part 3 of 3: Move turtle and display.

We must get the turlte, move it to an (x,y) location, then draw the flower.

```
    Turtle turtle = get();

    turtle.up();
    turtle.goTo(x, y);
    turtle.down();
    draw(color, petals, radius, size);
}
```

Modify FlowerArt.java

Remove your random flowers in the `draw` method and replace it with this. This will draw 100 randomly genreated flowers on the screen.

```
for (int i = 0; i < 100; i++) {
    flower.draw();
}
```