CART 263
Creative Computation 2

Email: l.wilkins@concordia.ca
Office Hours: Tuesday 12-1
Course Github: https://github.com/LeeCyborg/CART263-W-23
TA: tricia.enns@gmail.com

What we'll be doing today

- Intro to OOP & live coding
- 10 small challenges (in pairs)
- Project work time

What is Object Oriented Programming?

- A way of efficiently organizing code and thinking about programming
- An easy way to make a template for making many things with 1 piece of code
- A tool for creating more complexity

Terminology

Class: The part of code that holds all the information to make an object, like a blue print.

Object: The structure created by a class.

Instance: A specific object created from a class

Constructor: A piece of code that runs when you make an instance, like Setup.

Method: a function that belongs to a class.

“This” is a way of referring to local variables that only exist within that class.

Concepts

Class:
Fruit

Objects:
Mango, Apple, Pear
(Each is a fruit with different sizes, shapes, tastes, colours, textures)

Using OOP helps us create larger overacting concepts that apply to a subset of items we can create. All fruit have specific parameters we can describe them by. They all of specific functions. We can describe all fruits in this "fruit" class by just changing a few things about them.

Lets start basic…

We'll start by creating similar objects, and as we learn more, we can begin to imagine how to create different objects. For now, lets think of:

Class:
Particle

Object:
All objects are balls, with different X and Y.

Anatomy of an object

Create a new variable, and get it to hold "MyClass"

```
Let ball = new MyClass();
```

Run a method of "MyClass", "myMethod"

```
MyClass.myMethod();
```

Create a class, MyClass

A constructor for MyClass

```
class MyClass {
  constructor() {

  }
  myMethod() {
  }
}
```

A method that belongs to MyClass

Lets make a class to describe all balls, called "Particle"

This is the name of the class, Particle

```
class Particle {
  constructor() {
    this.x = height/2;
    this.y = width/2;
 }
  display() {
    ellipse(this.x, this.y, 10);
  }
}
```

A constructor is a set of instructions that happens when an object is created
<u>Every class needs a constructor. Its a good place to put all your variables.</u>

This is an example of a method, a function that is a part of the class. Every particle can be displayed.

The word "this" is used to indicate we are talking about a variable that exists in this class.

Lets make a ball appear

We will make a variable to store our object

```
let ball; // Declare object
```

The variable we made is a new instance of "particle", the class we made. When you run this, the constructor of the class happens here.

```
function setup() {
  createCanvas(400, 400);
  ball = new Particle();
}
```

Here we display the ball using NameOfVariable.Method();

```
function draw() {
  background(200, 50, 100);
  ball.display();
}
```

"." Is used to access anything inside the instance.
For example, we can use ball.x to see the X position we created in the constructor

Break it down…

The class “Particle”: A general blueprint for a particle. All particles an X and Y position. All particles can be “displayed”.

The object “Ball”: A specific instance of Particle. It has its own unique X and Y position. ball.display(); uses these unique pieces of information.

Lets try:

```
let ball; // Declare object

function setup() {
  createCanvas(400, 400);
  ball = new Particle();
}

function draw() {
  background(200, 50, 100);
  ball.display();
}


class Particle {
  constructor() {
    this.x = height/2;
    this.y = width/2;
 }
  display() {
    ellipse(this.x, this.y, 10);
  }
}
```

1. Make the ball appear at a random spot
2. Make the ball shake and jitter

Add another method

```
class Particle {
  constructor() {
    this.x = height/2;
    this.y = width/2;
    this.speed = 5;
  }

  move() {
    this.x += random(-this.speed, this.speed);
    this.y += random(-this.speed, this.speed);
  }

  display() {
    ellipse(this.x, this.y, 10);
  }
}
```

Adding a speed variable for this class

Add a new method for moving that adds the speed to X and Y

Using your new method

```
let ball; // Declare object


function setup() {
  createCanvas(400, 400);
  ball = new Particle();
}

function draw() {
  background(200, 50, 100);
  ball.move();
  ball.display();
}
```

We can use the new method like this, exactly like a regular function but it applies only to the object we created

What if we want another ball?

```
let ball; // Declare object
let ball2;

function setup() {
  createCanvas(400, 400);
  ball = new Particle();
  ball2 = new Particle();
}

function draw() {
  background(200, 50, 100);
  ball.move();
  ball2.move();
  ball.display();
  ball2.display();
}
```

What is another way we can do this?

An array of objects

```
let ball = []; // Declare array
```

Create an empty array

```
function setup() {
  createCanvas(710, 400);
  for(let i = 0; i < 50; i++){
    ball[i] = new Particle();
  }
}
```

Use a for loop to create objects in each array element. I chose 50 particles here.

```
function draw() {
  background(0);
  for(let i = 0; i < ball.length; i++){
    ball[i].move();
  }
}
```

You can again use an array to loop through your objects to move them

Passing parameters to new objects

```
function setup() {
  createCanvas(710, 400);
  for(let i = 0; i < 50; i++){
    ball[i] = new Particle(i);
  }
}
[…]
class Particle {
  constructor(pSize) {
    this.x = random(width);
    this.y = random(height);
    this.diameter = pSize;

  }
```

Pass the index from the for loop into your new objects

Much like a function, you can pass a parameter to your constructor.

Lets play with the system, in teams of 2

1. **Merge move and display**
2. **Make the speed of the particle shaking depend on the mouse position**
3. **Change the color based on the particle position**
4. **Make the particles bounce off the walls**
5. **Make particles random sizes**
6. **Create a second type of particle that behaves differently**
7. **Make particles disappear if you click on them**
8. **Make new particles appear if you press a key**
9. **Make the particles change color the longer they are on screen**
10. **Make the particles into an abstract shape that changes over time**