

Object Oriented Programming



Key Points



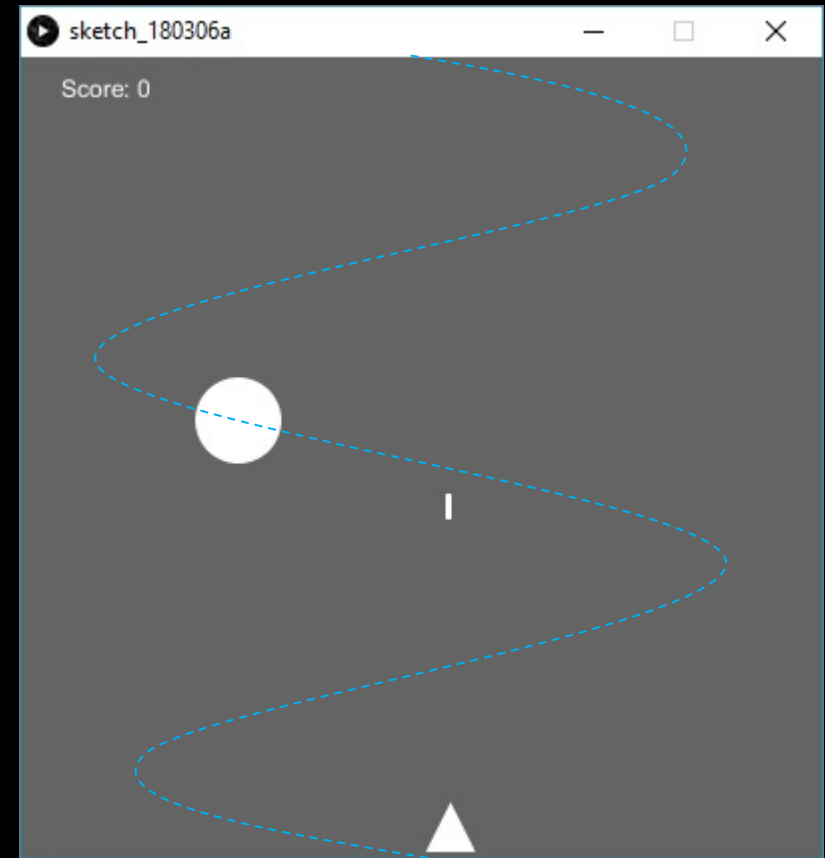
- 1) Introducing OOP (Object Oriented Programming) in Processing
- 2) Define classes, objects, methods, and attributes (instance variables).
- 3) Create objects from classes using `new`.
- 4) Using OOP in your Animations

Previously...

- Java constructs and data types
 - **Variables:** `int`, `float`, `boolean`, ...
 - **Drawing:** `line`, `ellipse`, `image`, `text`, ...
 - **Conditionals:** `if`, `if-else`, `switch`
 - **Loops:** `while`, `for`
 - **Functions**
- You used these concepts for simple, interesting programs
 - Bouncing ball, your custom superhero, fading images, moving shapes, ...
- There are more interesting ways of building programs...

Introductory Problem: The Space Shooter Game Revisited

- You have exactly
 - 1 spaceship
 - 1 enemy
 - 1 bullet at any time
- Spaceship moves horizontally with the mouse, but it doesn't move vertically at all.
- Enemy moves in a sin wave
 - Two full cycles
- You can shoot one bullet by mouse click – you can't shoot another one until bullet goes out of screen (top)
- If bullet hits enemy, score++



```
float ship_x = 0, ship_y = 375;
float bullet_x = 0, bullet_y = 375, bullet_len = 10;
boolean bullet_active = false;
float enemy_x = 200, enemy_y = 0, enemy_size = 40;
int score = 0;
```

```
void setup(){
  size(400,400);
  stroke(255);
  strokeWeight(3);}

void draw(){
  background(100);
  text("Score: " + score, 20,20);
  //game loop
  moveSpaceship();
  moveBullet();
  moveEnemy();
  detectCollisions();
  displaySpaceship();
  displayBullet();
  displayEnemy();
}
```

Without Objects...

Green is *Spaceship's* attributes & actions

Blue is *Bullet's* attributes & actions

Red is *Enemy's* attributes & actions

```
void moveSpaceship(){ ship_x = mouseX; }
```

```
void moveEnemy(){  
    enemy_y += 0.3;  
    enemy_x = 200+100*sin(map(enemy_y, 0, height, 0, 8*PI));  
}
```

```
void moveBullet(){  
    if(bullet_active){  
        bullet_y-=6;  
        if(bullet_y<0) bullet_active = false;  
    }else{  
        bullet_x = ship_x;  
        bullet_y = ship_y;  
    }  
}
```

```
void displaySpaceship(){  
    triangle(ship_x,ship_y,ship_x+10,ship_y+20,ship_x-10,ship_y+20);  
}
```

```
void  
displayBullet(){line(bullet_x,bullet_y,bullet_x,bullet_y+bullet_len);}
```

```
void displayEnemy(){ ellipse(enemy_x,enemy_y,enemy_size,enemy_size); }
```

```
void detectCollisions(){  
    if(dist(bullet_x,bullet_y,enemy_x,enemy_y)<enemy_size/2)  
        score++; //not accurate  
}
```

```
void mouseReleased(){  
    bullet_active = true;
```

It is very tedious to name the variables and functions for each item in the game. This can get worse when we have a real game with many many items.

One way to organize code – Use Tabs

- Put all the attributes and methods for each game item in one tab

```
spaceshooter bullet enemy ship ▼
1 int score = 0;
2 void setup(){
3   size(500,500);
4   textSize(22);
5 }
6 void draw(){
7   background(0);
8   fill(255); text("Score: " +score, 50,50);
9   moveShip();
10  moveEnemy();
11  moveBullet();
12  detectCollisions();
13  displayShip();
14  displayEnemy();
15  displayBullet();
16 }
```

```
spaceshooter bullet enemy ship ▼
1 float shipX, shipY=475;
2 void displayShip(){
3   fill(255); noStroke();
4   triangle(shipX,shipY, shipX+10,shipY+20, shipX-10,shipY+20);
5 }
6 void moveShip(){
7   shipX = mouseX;
8 }
9 }
```

```
spaceshooter bullet enemy ship ▼
1 float enemyX, enemyY, enemySize=50;
2 void displayEnemy(){
3   fill(255,0,255); noStroke();
4   ellipse(enemyX, enemyY, enemySize, enemySize);
5 }
6
7 void moveEnemy(){
8   enemyY += 2;
9   enemyX = 250 + 100*sin(map(enemyY,0,height,0,4*PI));
10 // enemyX = 250 + 100*sin(enemyY/height * 4*PI);
11 if(enemyY > height + enemySize/2) enemyY = -enemySize;
12 }
```

```
spaceshooter bullet enemy ship ▼
1 float bulletX, bulletY;
2 boolean shooting = false;
3 void displayBullet(){
4   stroke(0,255,255); strokeWeight(3);
5   line(bulletX, bulletY, bulletX, bulletY+10);
6 }
7 void moveBullet(){
8   if(!shooting){
9     bulletX = shipX;
10    bulletY = shipY;
11  }else{
12    bulletY -= 15;
13    if(bulletY<-10) shooting = false;
14  }
15 }
```

An Even Better way?

- A better way is *group* (or *encapsulate*) all *functions and attributes* of *each item* under a *code block*, and then giving them *simpler* names.
- This can be done using **OOP**

A solid red square is positioned on the left side of the slide, adjacent to the title bar.

OOP Basics

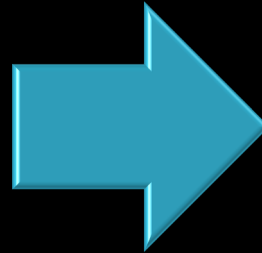
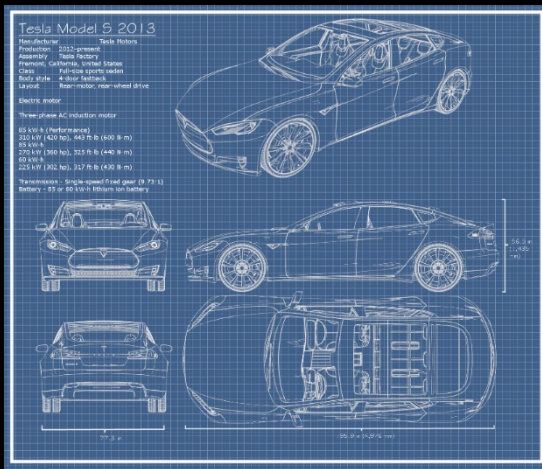
Two Phases to Code With Objects

- The Object Oriented Programming allow you to **encapsulate** the attributes and actions (functions) into objects, and then ask these objects to **act!**
- In order to use objects, we must go through two phases,
 - Phase (1) create the **design** of the object
 - The design is represented using a **class** which has
 - The **attributes** as **variables**
 - The **actions** as **functions**
 - Phase (2) **build** objects based on the design and use them.

Two Phases to Code With Objects, cont'd

■ How are objects created in the real-world?

■ TWO PHASES. Example: Cars.



Phase 1: Blueprint

- Attributes
- Behaviour (Actions)

Phase 2: Construction

In Processing, all objects of a design have the same actions and attributes (although the attribute values can be different).

Phase 1: Class Definition

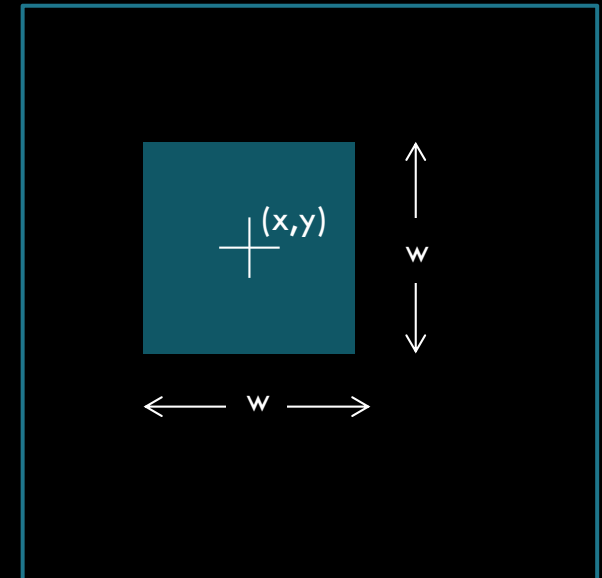
- To define a **class** that represents a **design**:
 - use the keyword **class** and provide a name for your class
 - enclose the contents of your class in brackets “{”, “}”
 - define any **attributes** (*instance variables*) for your class
 - define any **actions** (*functions*) in your class

- Syntax:

```
class classname {  
    //attributes  
    variableType variableName;  
  
    //actions  
    returnType functionName(par1, par2, ..., parN){  
        function implementation  
    }  
}
```

Let's Design our First Class

- Instead of using objects in the SpaceShooter game, let's start with something simple ... a **Square**!
- Assume we want to create a Square class as follows:
 - **Attributes:**
 - A square has position x,y and side length w , all of float type.
 - A square has fill color but not an outline.
 - **Functions:**
 - A square can move in the four directions
 - right, left, up, and down
 - A square can tell us its area using `getArea()` that returns the area value.
 - A square draw itself at (x,y)



The Square Class

Square

x : float
y : float
w : float
c : color

moveUp(): void
moveDown(): void
moveLeft(): void
moveRight(): void
getArea(): float
display(): void

```
class Square {  
    //attributes  
    float x = 100, y = 100, w = 50;  
    color c = color(0,0,255);  
    //behavior  
    void moveRight() {x += 10;}  
    void moveLeft() {x -= 10;}  
    void moveUp() {y -= 10;}  
    void moveDown() {y += 10;}  
    float getArea(){ return w*w;}  
    void display() {  
        rectMode(CENTER);  
        fill(c);  
        noStroke();  
        rect(x, y, w, w);  
    }  
}
```

Phase 2: Creating and Using Objects

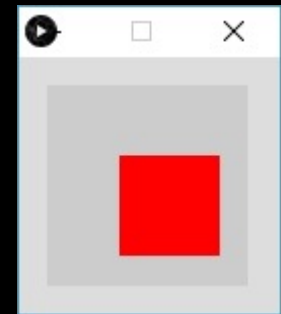
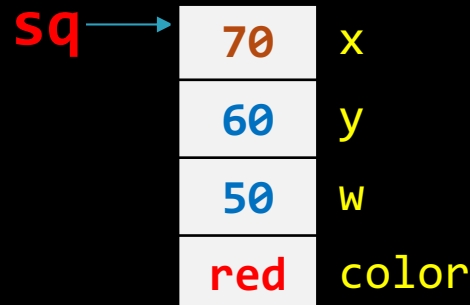
- A class is just a blue-print for creating objects.
 - By itself, a class performs no work or stores no data.
- For a class to be useful, we must create objects of the class.
 - Each object created is called an **object instance**.
- To create an object, we use the **new** keyword.
- When an object is created using the **new** keyword:
 - Processing allocates space for the object in memory.
 - The attributes are initialized to some values.
 - If no values specified, **default values** are used: 0 for numeric attributes, null for string, and false for Boolean variables.
 - Processing returns a pointer to where the object is stored in memory which we will call an **object reference**.

Creating and Using Objects (2)

- Objects are created using the **new** keyword.
- To access any of the object's attributes or functions, provide the object reference, followed by a dot (.), and then the name of the variable or function.
- Example:

```
// create a square object
Square sq = new Square();
// assign values to attributes
sq.x = 60;
sq.y = 60;
sq.w = 50;
sq.c = color(255,0,0);
// call square's functions
sq.moveRight();
sq.display();
```

Each object has its
own space in **memory**



Object Oriented SpaceShooter Game

- Now, let's go back to our space-ship game and try to use objects to model the game.
- As you saw, each of the Spaceship, Bullet, and Enemy has its own *attributes* and *actions*.
- Let's now design our classes and use them to create the game objects.

Object Oriented SpaceShooter Game

- Let's map the attributes and actions (functions) from the previous code to these three objects

Spaceship

ship_x : float

ship_y : float

moveSpaceship(): void

displaySpaceship():
void

Bullet

bullet_x : float

bullet_y : float

bullet_len : float

bullet_active : boolean

moveBullet(): void

displayBullet(): void

Enemy

enemy_x : float

enemy_y : float

enemy_size : float

moveEnemy(): void

displayEnemy(): void

Object Oriented SpaceShooter Game

- Now, let's simplify the function and attribute names

Spaceship

x : float

y : float

move(): void

display(): void

Bullet

x : float

y : float

len : float

isActive : boolean

move(): void

display(): void

Enemy

x : float

y : float

size : float

move(): void

display(): void

Spaceship Class

- The Spaceship class is used for describing a spaceship object. It uses the same attributes and functions from our game example.

Spaceship

x : float

y : float

move(): void

draw(): void

```
class Spaceship {  
    //attributes  
    float x = 0 , y = 375;  
    //behavior  
    void move(){  
        x = mouseX;  
    }  
    void display(){  
        triangle(x,y,x+10,y+20,x-10,y+20);  
    }  
}
```

Enemy Class

- The Enemy class is used for describing an enemy object. It uses the same attributes and functions from our game example.

Enemy

x : float
y : float
size : float

move(): void
display(): void

```
class Enemy {  
  //attributes  
  float x = 0 , y = 375, size = 40;  
  //behavior  
  void move(){  
    y+=0.3;  
    x=200+100*sin(map(y,0,height,0,8*PI));  
  }  
  void display(){  
    ellipse(x,y,size,size);  
  }  
}
```

Bullet Class

- The Bullet class is used for describing a bullet object. It uses the same attributes and functions from our game example.

Bullet

x : float
y : float
len : float
isActive : boolean

move(): void
display(): void

```
class Bullet {  
    //attributes  
    float x = 0 , y = 0, len = 10;  
    boolean isActive = false;  
    //behavior  
    void move(Spaceship ship){  
        if(isActive){  
            y -= 6;  
            if(y<0) isActive = false;  
        }else{ x = ship.x; y = ship.y; }  
    }  
    void display(){  
        line(x, y, x, y+len);  
    }  
}
```

Object Oriented SpaceShooter Game

```
int score = 0;
Spaceship ship = new Spaceship();
Enemy enemy = new Enemy();
Bullet bullet = new Bullet();
```

Create
objects
here!

```
void setup(){
    size(400,400); stroke(255); strokeWeight(3);
}
```

```
void draw(){
    background(100);
    text("Score: " + score, 20,20);
    //game loop
    ship.move();
    enemy.move();
    bullet.move(ship);
    detectCollisions();
    ship.display();
    enemy.display();
    bullet.display();
}
```

Use the objects in
the functions!

```
void detectCollisions(){
    if(dist(bullet.x, bullet.y, enemy.x, enemy.y) < enemy.size/2)
        score++;
}

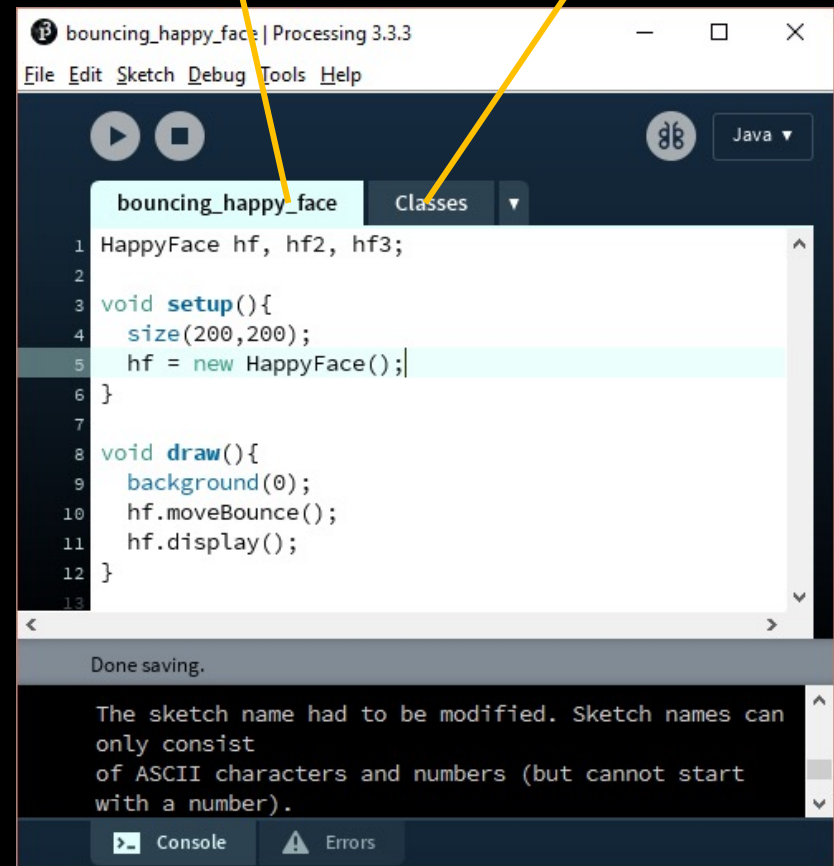
void mouseReleased(){
    bullet.isActive = true;
}
```

Remember Sketchbook Tabs?

- As the program gets larger, you will need to find a way to organize the code so that it is easy to follow.
- One way to do this is to separate code blocks using Sketchbook Tabs in the PDE.
- For example, you can put the HappyFace class you just created in its own tab instead of putting it below the draw() method.

Write the code for the main program here

Create more tabs to hold the code for your classes.



Object-Oriented Terminology

- An **object** is an instance of a class that has its own attributes and functions. Attributes and functions define what the object is and what it can do. *Each object has its own area in memory.*
- A **class** is a generic template (blueprint) for creating an object. All objects of a class have the same functions and attributes (although the attribute values can be different).
- An **attribute** (or **property**, or **instance variable**) is an attribute of an object.
- A **function** (or a **method**) is a set of statements that performs an action.
- A **parameter** is data passed into a method for it to use.

Practice with Object Oriented Programming



Key Points



- 1) Object Oriented Thinking
- 2) Old Problems Revisited
- 3) Exercises

Object Oriented Thinking

- Many of the programs we did before can be written using OOP
- You need apply **Object-Oriented Thinking**. Instead of focusing on designing functions, you need to focus on how to group **data (attributes)** and **functions (actions)** together into objects.
 1. **Identify the items** needed in your animation and create a class for each category of items (i.e. items that share the same design).
 2. **Create objects** in your game, initialize their attributes, and ask them to *act* (by calling their functions)
- You can even create **objects for non-physical objects**. For example, to detect collisions in our space-shooter game, you can create an object, **CollisionDetector**, whose sole purpose is to monitor the game and detect any collisions.
 - Note that this can also be done using a function `detectCollision()` as we did in the previous program.

Old Programs *REVISTED*

- Let's try to convert some of the programs we wrote so far to Object-Oriented Programs
 - Bouncing Ball
 - Spaceship
 - Buttons
 - *important as it facilitates creating GUIs*

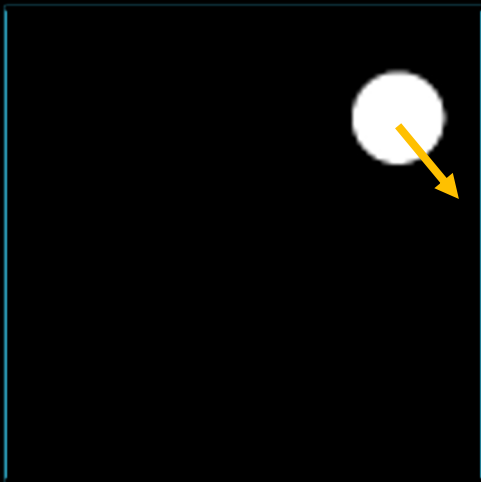
Example 1

Bouncing Ball *WITHOUT OOP*

This is the code we wrote before without OOP.

In order to re-think the code in OOP:

1. Identify items in animation → ball
2. Identify item's attributes → x, y, speedX, speedY, and r
3. Identify item's behavior → move(), bounce(), and display()




```
float speedX = 1, speedY = 2;
float x=20, y=100, r = 20;
void setup(){
    size(200,200);
}
void draw(){
    background(0);
    ellipse(x,y,2*r,2*r);
    //increment x
    x += speedX;
    //if ball at edge, reverse direction
    if( x > width-r || x < r )
        speedX = -speedX;
    //increment y. reverse when at edge
    y += speedY;
    if(y > height-r || y < r)
        speedY = -speedY;
}
```

Example 1

OOP Bouncing Ball – basic design

```
Ball ball;
void setup(){
    size(200,200);
    //create ball object
    ball = new Ball();
    //initialize ball object
    ball.x = width/2;
    ball.y = height/2;
    ball.speedX = 2;
    ball.speedY = 3;
    ball.r = 20;
}
void draw(){
    background(0);
    ball.move();
    ball.bounce();
    ball.display();
}
```

```
class Ball {
    //attributes
    float x,y,r,speedX,speedY;
    //behavior
    void move() {
        x = x + speedX;
        y = y + speedY;
    }
    void bounce() {
        if (x<r||x>width-r) speedX = -speedX;
        if (y<r||y>height-r)speedY = -speedY;
    }
    void display() {
        fill(0,100,255);
        stroke(255);
        strokeWeight(r/7);
        ellipse(x, y, 2*r, 2*r);
    }
}
```



should be part of the ball attributes?

OOP Bouncing Ball – adding color attributes

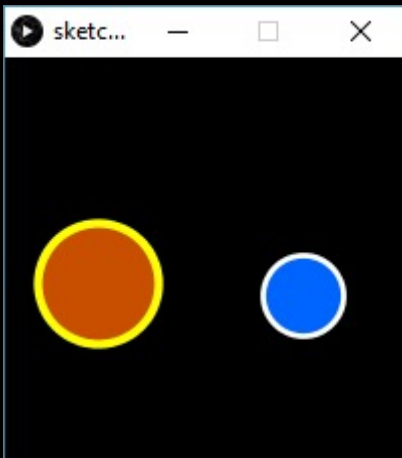
```
Ball ball;
void setup(){
    size(200,200);
    //create & initialize ball
    ball = new Ball();
    ball.x = width/2;
    ball.y = height/2;
    ball.speedX = 2;
    ball.speedY = 3;
    ball.r = 20;
    ball.fillColor = color(0,100,255);
    ball.strokeColor = color(255);
}
void draw(){
    background(0);
    ball.move();
    ball.bounce();
    ball.display();
}
```

```
class Ball {
    //attributes
    float x,y,r,speedX,speedY;
    color fillColor, strokeColor;
    //behavior
    void move() {...}
    void bounce() {...}
    void display() {
        fill(fillColor);
        stroke(strokeColor);
        strokeWeight(r/7);
        ellipse(x, y, 2*r, 2*r);
    }
}
```

Same as previous slide, except we created attributes for the colors

OOP: Many bouncing balls!

- Now, we can create several bouncing balls using the same Ball class



```
Ball b1, b2;
void setup(){
  size(200,200);
  b1 = new Ball();
  b1.x = width/2;  b1.y = height/2;
  b1.speedX = 2;   b1.speedY = 3;
  b1.r = 20;
  b1.fillColor = color(0,100,255);
  b1.strokeColor = color(255);
  b2 = new Ball();
  b2.x = width/2;  b2.y = height/2;
  b2.speedX = -3;  b2.speedY = -2;
  b2.r = 30;
  b2.fillColor = color(255,100,0,200);
  b2.strokeColor = color(255,255,0);
}
void draw(){
  background(0);
  b1.move(); b1.bounce(); b1.display();
  b2.move(); b2.bounce(); b2.display();
}

class Ball {
  same as before
}
```

Example 4

Advanced Ball!

- This design is for a ball that
 - can move, bounce off the edges, and wrap around the sketch
 - Initializes its attributes to some random values

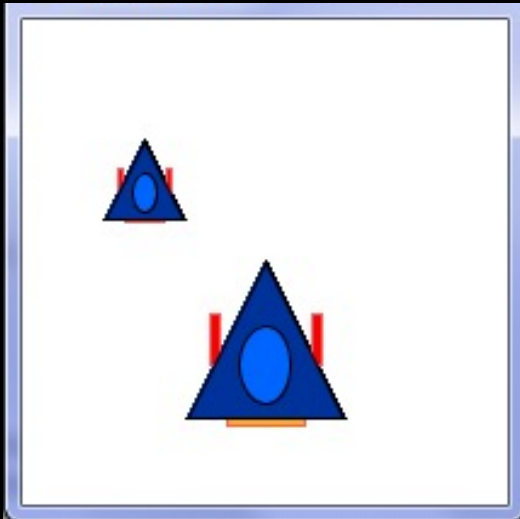
```
Ball b1, b2;
void setup(){
  size(200,200);
  b1 = new Ball(); b2 = new Ball();
}
void draw(){
  background(0);
  b1.move(); b1.bounce(); b1.display();
  b2.move(); b2.wrap(); b2.display();
}

class Ball {
  //attributes
  float x=random(20,80), y=random(20,80), r = 20;
  float speedX = random(-2,2), speedY = random(-2,2);
  color filling=color(random(255), random(255),random(255));
  color outline=color(255);
  //methods (actions)
  void move(){
    x += speedX; y += speedY;
  }
  void bounce(){
    if( x > width-r || x < r ) speedX = -speedX;
    if(y > height-r || y < r) speedY = -speedY;
  }
  void wrap(){
    if(x > width+r) x = -r;
    else if(x < -r) x = width + r ;
    if(y > height+r) y = -r;
    else if(y < -r) y = height + r ;
  }
  void display(){
    fill(filling); stroke(outline);
    ellipse(x,y,2*r,2*r);
  }
}
```

Example 4

Spaceships *WITHOUT OOP*

1. Items in animation → spaceship
2. item's attributes → location and size
3. item's behavior → move and display



```
void setup(){
    size(300,300);
}
void draw() {
    drawSpaceship(mouseX,mouseY,64);
    drawSpaceship(50,50,32);
}
void drawSpaceship(int x, int y, int size) {
    // draw side guns
    rectMode(CENTER);
    stroke(255, 90, 90);
    strokeWeight(1);
    fill(255, 0, 0);
    rect(x-size/3, y+size/2, size/15, size/3);
    rect(x+size/3, y+size/2, size/15, size/3);
    // draw jet engine
    fill(255, 180, 0);
    rect(x, y+size, size/2, size/10);
    // draw main body
    stroke(0);
    fill(0, 50, 155);
    triangle(x,y,x+size/2,y+size,x-size/2, y+size);
    fill(0, 100, 255);
    ellipse(x, y+2*size/3, size/3, size/2);
}
```

Example 4

OOP Spaceship

```
Spaceship s1, s2;
void setup(){
    size(300,300);
    s1 = new Spaceship();
    s1.size=64;
    s2 = new Spaceship();
    s2.size=32;
}

void draw() {
    background(0);
    s1.moveTo(mouseX,mouseY);
    s2.moveTo(50,50);
    s1.display();
    s2.display();
}
```

```
class Spaceship {
    float x, y, size;
    void moveTo(int x1,int y1){
        x = x1;
        y = y1;
    }
    void display() {
        rectMode(CENTER);
        stroke(255, 90, 90);
        strokeWeight(1);
        fill(255, 0, 0);
        rect(x-size/3, y+size/2, size/15, size/3);
        rect(x+size/3, y+size/2, size/15, size/3);
        // draw jet engine
        fill(255, 180, 0);
        rect(x, y+size, size/2, size/10);
        // draw main body
        stroke(0);
        fill(0, 50, 155);
        triangle(x,y,x+size/2,y+size,x-size/2,y+size);
        fill(0, 100, 255);
        ellipse(x, y+2*size/3, size/3, size/2);
    }
}
```

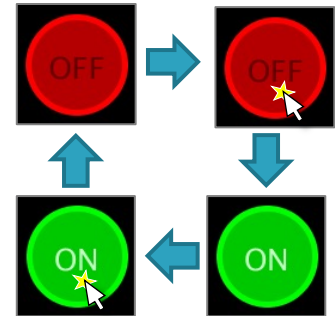
We can also use coordinate translation

Example 5

Toggle Button **WITHOUT OOP**



```
boolean active = false;           // button attributes
int x = 50, y = 50, r = 40;
void setup() {
  size(200, 200);
}
void draw() {
  background(0);
  strokeWeight(5);
  textSize(25);   textAlign(CENTER,CENTER);
  if(active) {
    fill(0, 200, 0);  stroke(0,255,0);
    ellipse(x,y,2*r,2*r);
    fill(200,255,200);text("ON",x,y);
  }else{
    fill(180, 0, 0);  stroke(255,0,0);
    ellipse(x,y,2*r,2*r);
    fill(100,0,0);    text("OFF",x,y);
  }
}
void mousePressed() {
  if(dist(mouseX,mouseY,x,y)<r) active = ! active;
}
```



Example 5

TWO toggle buttons **WITHOUT OOP**

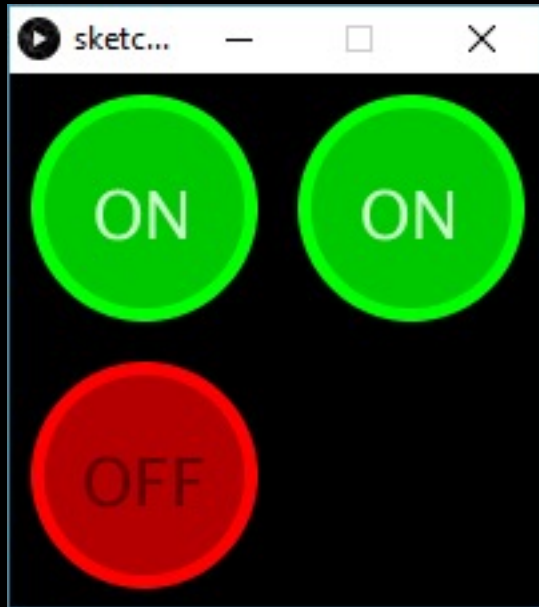


```
boolean active1 = false; // button1
int x1 = 50, y1 = 50, r1 = 40;
boolean active2 = false; // button2
int x2 = 150, y2 = 50, r2 = 40;
void setup() {
  size(200, 200);
}
void draw() {
  background(0);
  strokeWeight(5);
  textSize(25);  textAlign(CENTER,CENTER);
  //button1
  if(active1) {
    fill(0, 200, 0);  stroke(0,255,0); ellipse(x1,y1,2*r1,2*r1);
    fill(200,255,200);text("ON",x1,y1);
  }else{
    fill(180, 0, 0);  stroke(255,0,0); ellipse(x1,y1,2*r1,2*r1);
    fill(100,0,0);    text("OFF",x1,y1);
  }
  //button2
  if(active2) {
    fill(0, 200, 0);  stroke(0,255,0); ellipse(x2,y2,2*r1,2*r2);
    fill(200,255,200);text("ON",x2,y2);
  }else{
    fill(180, 0, 0);  stroke(255,0,0); ellipse(x2,y2,2*r1,2*r2);
    fill(100,0,0);    text("OFF",x2,y2);
  }
}
void mousePressed() {
  if(dist(mouseX,mouseY,x1,y1)<r1) active1 = ! active1;
  if(dist(mouseX,mouseY,x2,y2)<r2) active2 = ! active2;
}
```

Copy/paste
code

Example 5

FOUR toggle buttons **WITHOUT OOP**



What if we have 10 buttons ?? Or 20?? How about a 100?

What if we decided to make a change to the button design?

```
boolean active1 = false; // button1
int x1 = 50, y1 = 50, r1 = 40;
boolean active2 = false; // button2
int x2 = 150, y2 = 50, r2 = 40;
boolean active3 = false; // button3
int x3 = 150, y3 = 50, r3 = 40;
void setup() {
  size(200, 200);
}
void draw() {
  background(0);
  strokeWeight(5);
  textSize(25);  textAlign(CENTER,CENTER);
  //button1
  if(active1) {
    fill(0, 200, 0);  stroke(0,255,0); ellipse(x1,y1,2*r1,2*r1);
    fill(200,255,200);text("ON",x1,y1);
  }else{
    fill(180, 0, 0);  stroke(255,0,0); ellipse(x1,y1,2*r1,2*r1);
    fill(100,0,0);    text("OFF",x1,y1);
  }
  //button2
  if(active2) {
    fill(0, 200, 0);  stroke(0,255,0); ellipse(x2,y2,2*r2,2*r2);
    fill(200,255,200);text("ON",x2,y2);
  }else{
    fill(180, 0, 0);  stroke(255,0,0); ellipse(x2,y2,2*r2,2*r2);
    fill(100,0,0);    text("OFF",x2,y2);
  }
  //button3
  if(active3) {
    fill(0, 200, 0);  stroke(0,255,0); ellipse(x3,y3,2*r3,2*r3);
    fill(200,255,200);text("ON",x3,y3);
  }else{
    fill(180,0,0);    stroke(255,0,0); ellipse(x3,y3,2*r3,2*r3);
    fill(100,0,0);    text("OFF",x3,y3);
  }
}
void mousePressed() {
  if(dist(mouseX,mouseY,x1,y1)<r1) active1 = ! active1;
  if(dist(mouseX,mouseY,x2,y2)<r2) active2 = ! active2;
  if(dist(mouseX,mouseY,x3,y3)<r3) active3 = ! active3;
}
```

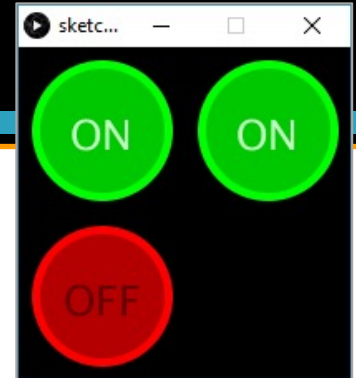
More
copy/paste!!

Example 5

OOP Toggle Buttons (3 buttons)

```
Button b1, b2, b3;
void setup() {
  size(200, 200);
  b1 = new Button();
  b1.x = 50; b1.y = 50;
  b2 = new Button();
  b2.x = 150; b2.y = 50;
  b3 = new Button();
  b3.x = 50; b3.y = 150;
}
void draw() {
  background(0);
  b1.display();
  b2.display();
  b3.display();
}
void mousePressed(){
  b1.checkClicked();
  b2.checkClicked();
  b3.checkClicked();
}
```

```
class Button{
  boolean active = false;
  int x = 50, y = 50, r = 40;
  void display(){
    strokeWeight(5);
    textSize(25);   textAlign(CENTER,CENTER);
    if(active) {
      fill(0, 200, 0);  stroke(0,255,0);
      ellipse(x,y,2*r,2*r);
      fill(200,255,200);text("ON",x,y);
    }else{
      fill(180, 0, 0);  stroke(255,0,0);
      ellipse(x,y,2*r,2*r);
      fill(100,0,0);    text("OFF",x,y);
    }
  }
  void checkClicked(){
    if(dist(mouseX,mouseY,x,y)<r) active=!active;
  }
}
```



Happy Face Class Definition



- [+2] Implement a class HappyFace:
 - A HappyFace has position x, y and radius r , all of float type.
 - A HappyFace has fill color and outline color.
 - Write a function `display()` that draws the face centered at (x, y) .
 - The mouth is an arc with a diameter equal to $1.6 r$.
 - The eyes are ellipses with dimensions $\frac{r}{2} \times \frac{r}{4}$
 - The outline's thickness is $\frac{r}{20}$
- [+1] Use the HappyFace class to create two different faces and display them on the sketch.
 - Use the **dot operator** to set the attributes. E.g.:
 - `face1.x = width/2;`
 - `face1.y = height/2;`
 - `face1.r = width/4.`
 - `face1.fillColor = color(255,255,0);`
 - ...etc
- [+2] Add code to move the faces and bounce them off the sketch edges



Hint: code for drawing 1 happy face

- Here is the non-OOP code

```
float x = 50, y = 50, r = 20;

fill(255,255,0);

stroke(255,155,0);

strokeWeight(r/20);

ellipse(x,y,2*r,2*r);

//face

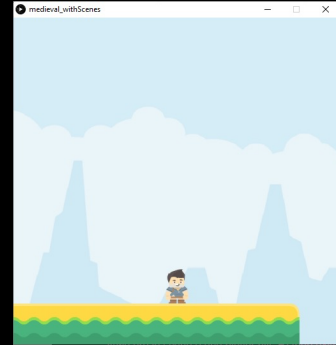
arc(x,y,1.6*r,1.6*r,.1*PI,.9*PI); //mouth

ellipse(x+r/2,y-r/4,r/4,r/2);      //right eye

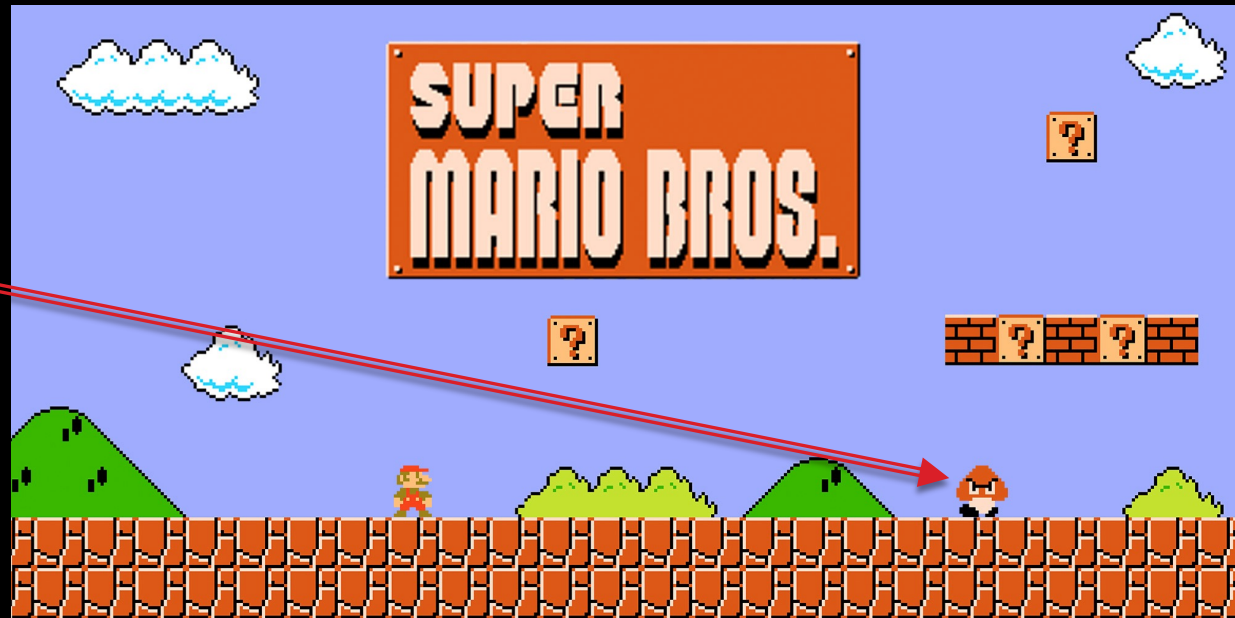
ellipse(x-r/2,y-r/4,r/4,r/2);      //left eye
```

Update your game

1. Start with your game from Lecture Activity 8
2. Copy the code to Lecture Activity 9 (cp)
3. If you haven't already, organize your code so it uses tabs
4. Use object-oriented programming to create two classes:
 1. Ground Enemies (or Goombas from the original Mario)
 2. Flying Enemies (or Paratroopa from the original Mario)
5. Add 3 ground enemies and 5 flying enemies; they should all have different names, and be of different colours; flying enemies should be restricted to the air, and ground enemies cannot fly.
6. Remember that the original names and images are probably copyrighted, so you will have to find your own!



“Goomba”



“Paratrooper”

Images are Copyright of Nintendo