Welcome back!

Object Oriented Programming

- Encapsulation
  - Classes encapsulate **state** (fields) and **behavior** (methods)
- Polymorphism
  - Signature Polymorphism – **Overloading**
  - Subtype Polymorphism – **Inheritance**

Gets and sets

- Instead of accessing data fields directly
  - ball.x = 5;
- Define methods to access them
  - int getX () { return x;}
  - int getFoo () { return foo;}
  - void setX(int x) {this.x = x;}
  - void setFoo(int foo) {this.foo = foo;}
  - ball.setX(5);

Creating a set of Graphic Object Classes

- All have…
  - X, Y location
  - width and height fields
  - fill and stroke colors
  - A draw() method
  - A next() method defining how they move
  - …
- Implementation varies from class to class

Creating a set of Graphic Object Classes

- Problems

How would you hold all your objects?

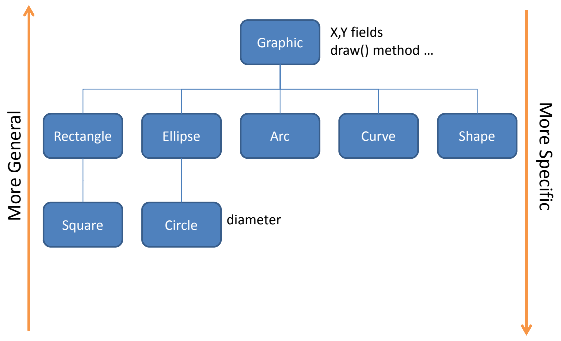
– Array?

What if one class had extra methods or special arguments?

Sometimes you want to think of an object as a generic Graphic (X,Y location and draw() method)

Sometimes you want to think of an object as a specific type (extra methods, extra fields, …)

Graphic Object Hierarchy



Inheritance gives you a way to relate your objects in a hierarchical manner

Inheritance

- **Superclass (base class)** – higher in the hierarchy
- **Subclass (child class)** – lower in the hierarchy
- A subclass is **derived from** from a superclass
- Subclasses **inherit** the **fields** and **methods** of their superclass.
  - I.e. subclasses automatically **"get"** stuff in superclasses
- Subclasses can **override** a superclass method by redefining it.
  - They can replace anything by redefining locally

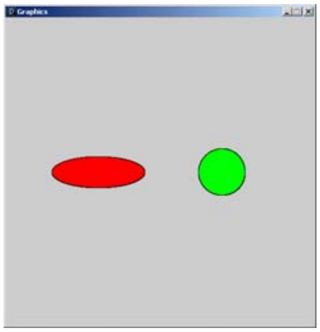
```
// Ellipse base class
class Ellipse {

  float X;
  float Y;
  float W;
  float H;

  // Ellipses are always red
  color fillColor =
             color(255,0,0);

  Ellipse(float X, float Y,
          float W, float H)
  {
    this.X = X;
    this.Y = Y;
    this.W = W;
    this.H = H;
  }

  void draw() {
    ellipseMode(CENTER);
    fill(fillColor);
    ellipse(X, Y, W, H);
  }
}
```

```
// Circle derived class
class Circle extends Ellipse {

  Circle(float X, float Y, float D) {
    super(X, Y, D, D);

    // Circles are always green
    fillColor = color(0,255,0);
  }
}
```

- The **extends** keyword creates hierarchical relationship between classes.
- The Circle class gets all fields and methods of the Ellipse class, automatically.
- The **super** keyword refers to the base class in the relationship.
- The **this** keyword refers to the object itself.

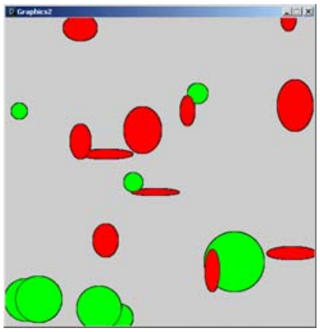
Graphics.pde

```
// Graphics
Ellipse e = new Ellipse(150, 250, 150, 50);
Circle c = new Circle(350, 250, 75);

void setup() {
  size(500, 500);
  smooth();
}

void draw() {
  e.draw();
  c.draw();
}
```



Graphics.pde

```
// Graphics2
Ellipse[] e = new Ellipse[20];

void setup() {
  size(500, 500);
  smooth();

  for (int i=0; i<e.length; i++) {

    float X = random(0, width);
    float Y = random(0, height);
    float W = random(10, 100);
    float H = random(10, 100);

    // Ellipses and Circles are
    // stored in the same array
    if (random(1.0) < 0.5)
      e[i] = new Ellipse(X,Y,W,H);
    else
      e[i] = new Circle(X,Y,W);
  }
}

void draw() {
  for (int i=0; i<e.length; i++)
    e[i].draw();
}
```



Ellipses and Circles in the same array!

Graphics2.pde

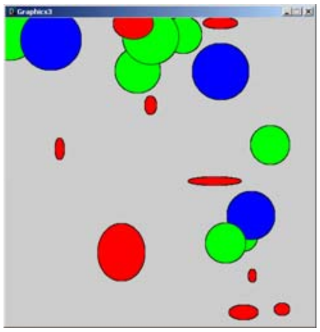
```
// Ellipse base class
class Ellipse {

  float X;
  float Y;
  float W;
  float H;

  // Ellipses are always red
  color fillColor =
             color(255,0,0);

  Ellipse(float X, float Y,
          float W, float H)
  {
    this.X = X;
    this.Y = Y;
    this.W = W;
    this.H = H;
  }

  void draw() {
    ellipseMode(CENTER);
    fill(fillColor);
    ellipse(X, Y, W, H);
  }

  // Do nothing
  void mousePressed() {}
}
```

```
// Circle derived class
class Circle extends Ellipse {

  Circle(float X, float Y, float D) {
    super(X, Y, D, D);

    // Circles are always green
    fillColor = color(0,255,0);
  }

  // Change color of circle when clicked
  void mousePressed() {
    if (dist(mouseX, mouseY, X, Y) < 0.5*W)
      fillColor = color(0,0,255);
  }
}
```

- The mousePressed behavior of the Circle class **overrides** the default behavior of the Ellipse class.

Graphics3.pde

```
// Graphics3
Ellipse[] e = new Ellipse[20];

void setup() {
  size(500, 500);
  smooth();

  // Stuff removed …
}

void draw() {
  for (int i=0; i<e.length; i++)
    e[i].draw();
}

void mousePressed() {
  for (int i=0; i<e.length; i++)
    e[i].mousePressed();
}
```



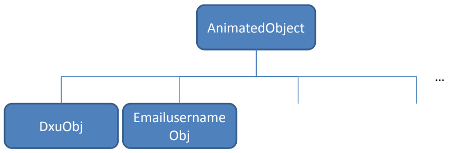
Graphics3.pde

A few more rules about inheritance …

- A child's constructor is responsible for calling the parent's constructor
- The first line of a child's constructor should use the *super* reference to call the parent's constructor
- The *super* reference can also be used to reference other variables and methods defined in the parent's class

Use inheritance to solve our aquarium problem

- The AnimatedObject class has two methods that need to be overridden.
  - void display(), void move()



AnimatedObject

DxuObj

Emailusername Obj

…