Inheritance

Questions about Assignment 5?

Review

- Objects
  - data fields
  - constructors
  - Methods
- Classes

Using the Ball class

Treat in a manner very similar to a primitive data type.

```
Ball[] balls = new Ball[20];
```

Declare an array of Balls.

```
void setup() {
  size(500, 500);
  fill(255, 0, 0);
  smooth();
  ellipseMode(CENTER);

  // Create all new Ball objects
  for (int i = 0; i < balls.length; i++) {
    balls[i] = new Ball();
  }
}
```

New objects are created with the ***new*** keyword.

```
void draw() {
  background(255);

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

Methods of objects stored in the array are accessed using dot-notation.

PieChart Class/Birthdays.pde

- How do we go from Imperative code to Object Oriented code?
  - Identify which variables are fields
    - variables that would give the object meaning
  - Identify code where the selected variables are initialized
    - put that code before or inside your constructor.
      - if the value can be derived from other fields, then compute the value in the constructor
      - otherwise set the value, then pass it into the constructor.
  - Identify code that operates on the selected fields
    - make that code into a method

Identify which variables are fields

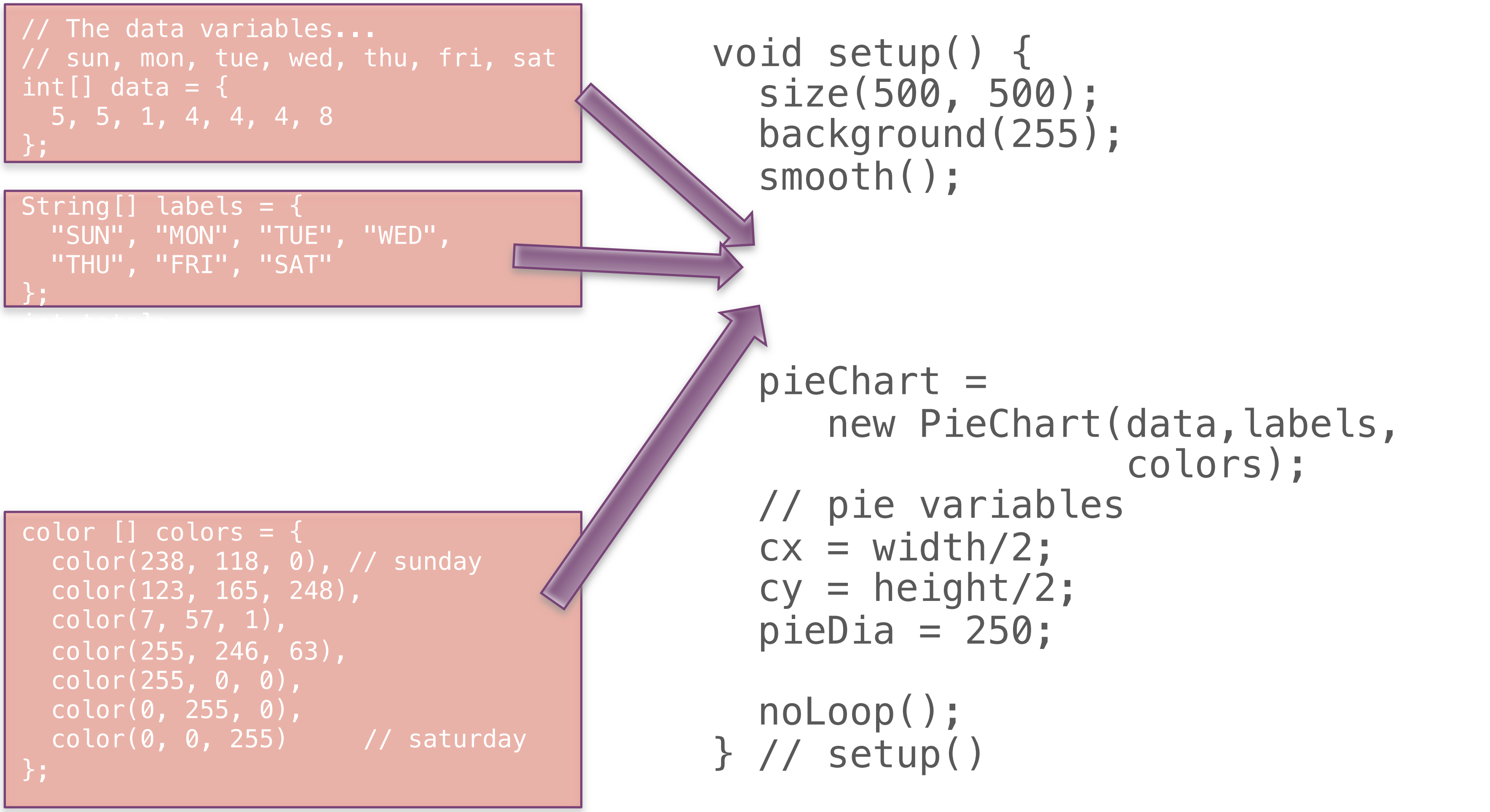
Global variables

```
// The data variables...
// sun, mon, tue, wed, thu, fri, sat
int[] data = {
  5, 5, 1, 4, 4, 4, 8
};

String[] labels = {
  "SUN", "MON", "TUE", "WED",
  "THU", "FRI", "SAT"
};
int total;
float[] perc = new float[7];

// The sketch variables
float cx, cy, pieDia;
float startAngle, stopAngle;

color [] colors = {
  color(238, 118, 0), // sunday
  color(123, 165, 248),
  color(7, 57, 1),
  color(255, 246, 63),
  color(255, 0, 0),
  color(0, 255, 0),
  color(0, 0, 255)     // saturday
};
```

Fields

```
int[] data; // the values

String[] labels; // labels for each value

color[] colors; // colors for each value

float[] perc; // the plotted value



/*

 What about total, cx, cy, pieDia,
startAngle, and stopAngle?

*/
```

Identify code where the selected variables are initialized



Initialize values locally	Pass the value in the constructor

```
// The data variables...
// sun, mon, tue, wed, thu, fri, sat
int[] data = {
  5, 5, 1, 4, 4, 4, 8
};
```

```
String[] labels = {
  "SUN", "MON", "TUE", "WED",
  "THU", "FRI", "SAT"
};
```

```
color [] colors = {
  color(238, 118, 0), // sunday
  color(123, 165, 248),
  color(7, 57, 1),
  color(255, 246, 63),
  color(255, 0, 0),
  color(0, 255, 0),
  color(0, 0, 255)      // saturday
};
```

```
void setup() {
  size(500, 500);
  background(255);
  smooth();



  pieChart =
     new PieChart(data,labels,
                        colors);
  // pie variables
  cx = width/2;
  cy = height/2;
  pieDia = 250;

  noLoop();
} // setup()
```

Identify code where the selected variables are initialized

derived from other fields

```
void setup() {
  size(500, 500);
  background(255);
  smooth();

  // process
  // compute the total population
  total = 0;
  for (int i=0; i < data.length; i++) {
    total += data[i];
  }

  // compute percentages
  for (int i=0; i < data.length; i++) {
    perc[i] = float(data[i])/total;
  }

  // pie variables
  cx = width/2;
  cy = height/2;
  pieDia = 250;

  noLoop();
} // setup()
```

compute in Constructor

```
PieChart(float[] data,
         String[] labels,
         color[]  colors) {


   this.data = data;
   this.labels = labels;
   this.colors = colors;
   // instantiate float[] for perc
   perc = new float[data.length];
   // compute the total population
   float total = 0;
   for (int i=0; i < data.length; i++) {
     total += data[i];
   }

   // compute percentages
   for (int i=0; i < data.length; i++) {
     perc[i] = float(data[i])/total;
   }
}
```

Identify code that operates on the selected fields

draw based on perc, labels and colors

```
startAngle = 0;
stopAngle = 0;
for (int i=0; i < perc.length; i++) {
    // set up pie parameters
    // for ith slice
    startAngle = stopAngle;
    stopAngle = startAngle +
                   TWO_PI*perc[i];

    // draw the pie
    …

    // draw legend
    // draw title
    …
}
```

make a display() method

```
void display(float cx, float cy,
             float pieDia) {
  startAngle = 0;
  stopAngle = 0;
  for (int i=0; i < perc.length; i++) {
    // set up pie parameters
    // for ith slice
    startAngle = stopAngle;
    stopAngle = startAngle +
                   TWO_PI*perc[i];

    // draw the pie
    …

    // draw legend
    // draw title
    …
  }
}
```

Identify code that operates on the selected fields

call display from void setup() or void draw()

```
// pie variables
  float xCenter = width/2;
  float yCenter = height/2;
  float dia = 250;
  birthdayChart.display(xCenter,
                        yCenter,dia);
```

make a display() method

```
void display(float cx, float cy,
             float pieDia) {
  startAngle = 0;
  stopAngle = 0;
  for (int i=0; i < perc.length; i++) {
    // set up pie parameters
    // for ith slice
    startAngle = stopAngle;
    stopAngle = startAngle +
               TWO_PI*perc[i];

    // draw the pie
    …

    // draw legend
    // draw title
    …
  }
}
```

Object Oriented Programming

- Encapsulation
  - Classes encapsulate **state** (fields) and **behavior** (methods)

- Polymorphism
  - Signature Polymorphism – **Overloading**
  - Subtype Polymorphism – **Inheritance**

+

gets (Accessors) and sets (Mutators)

- Instead of accessing data fields directly
  - ball.x = 5;

- Define methods to access them
  - int getX () { return x;} // accessor for x
  - int getFoo () { return foo;} // accessor for foo
  - void setX(int x) {this.x = x;} // mutator for x
  - void setFoo(int foo) {this.foo = foo;} // mutator for foo
- Call methods
  - ball.setX(5); // changing x of ball
  - int added = ball.getFoo() + ball.getX();

+ 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?

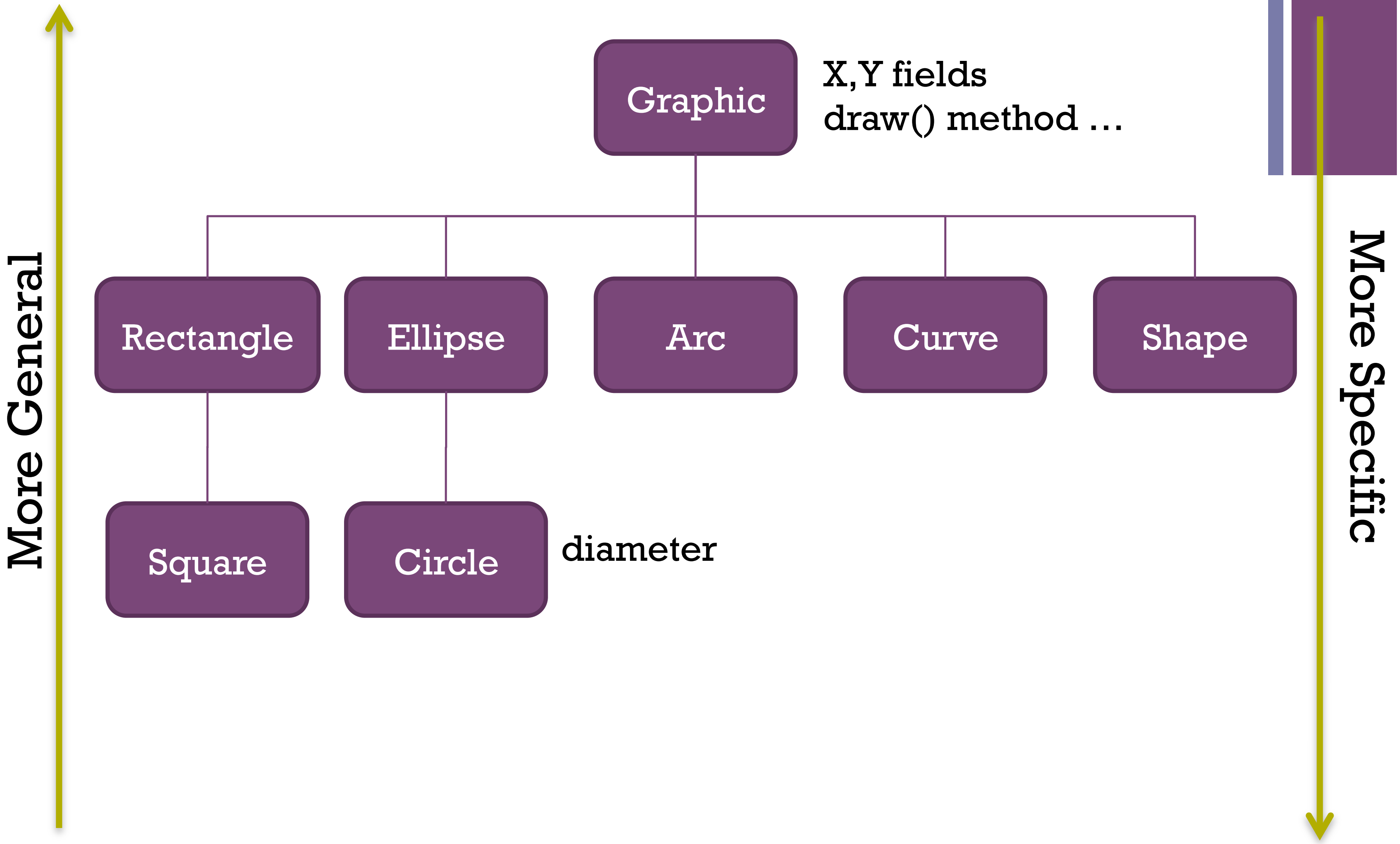
    What if one class had extra methods or special arguments?

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

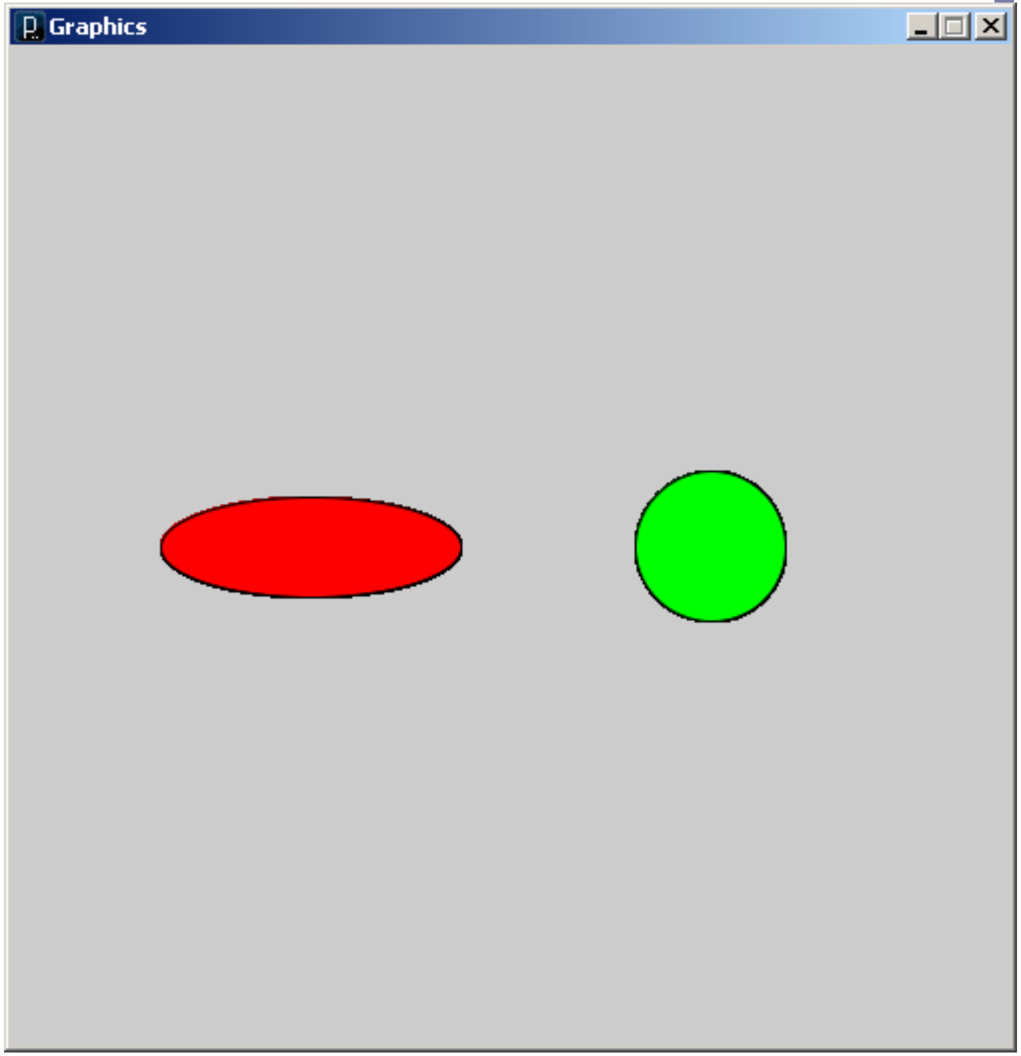
```
// Ellipse base class
class Ellipse {

  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.

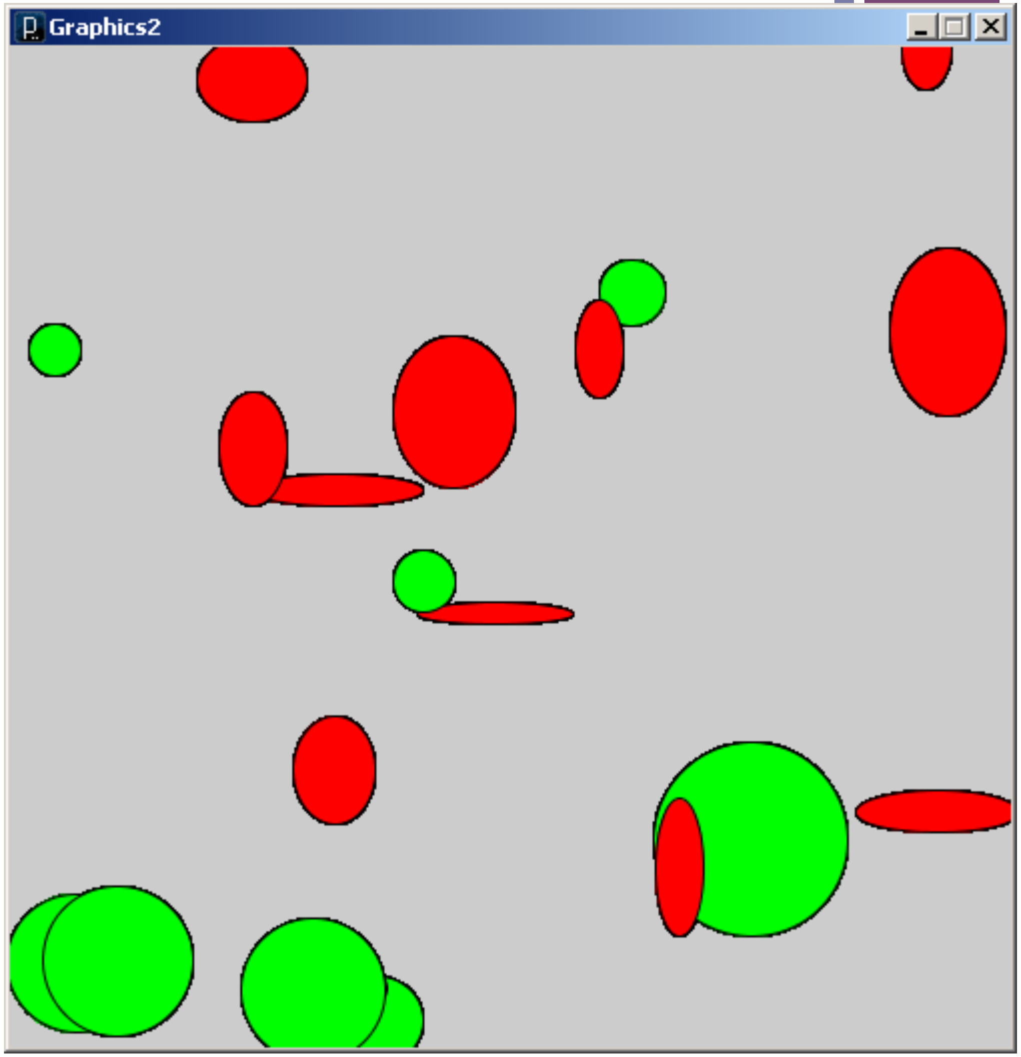
Graphics.pde

```
// 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 are 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() {}
}
```

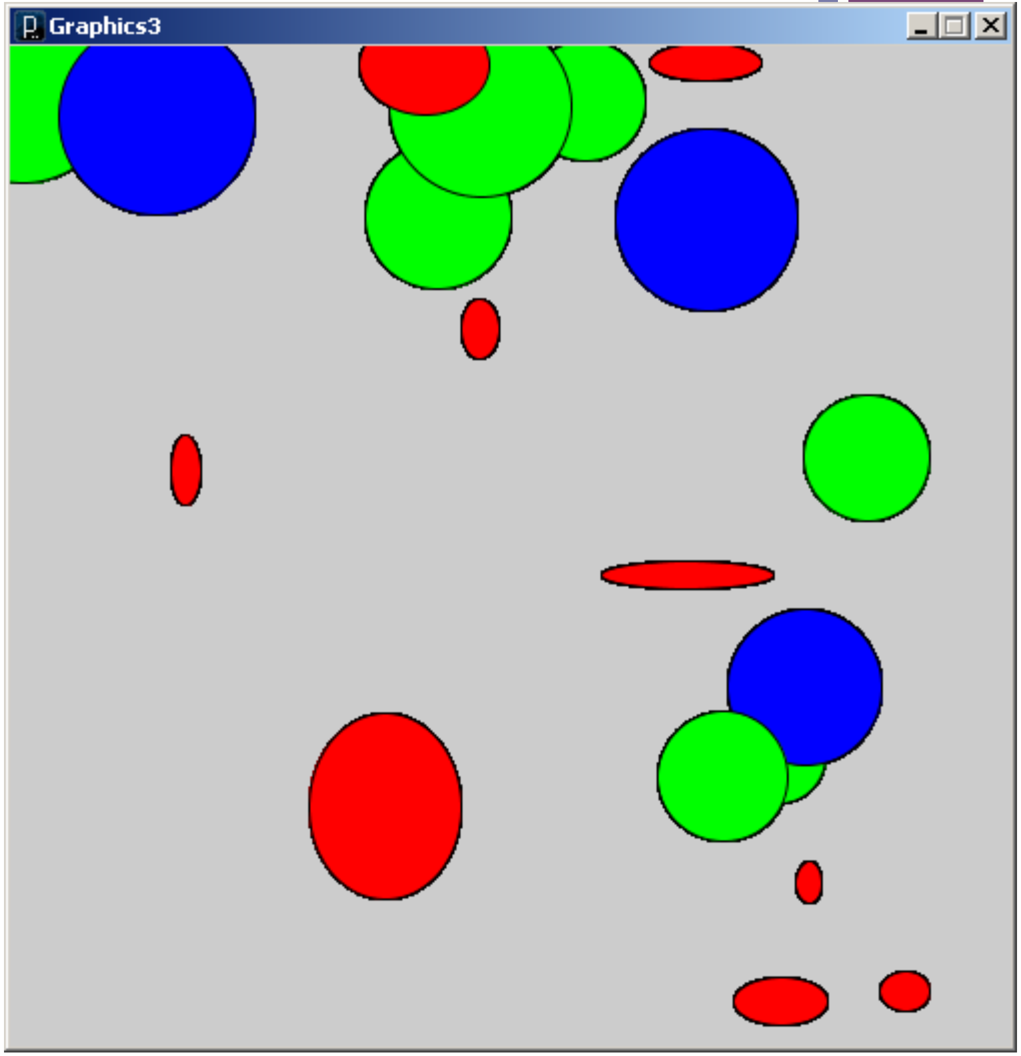
```
// Circle derived class
class Circle extends Ellipse {

  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();
}
```



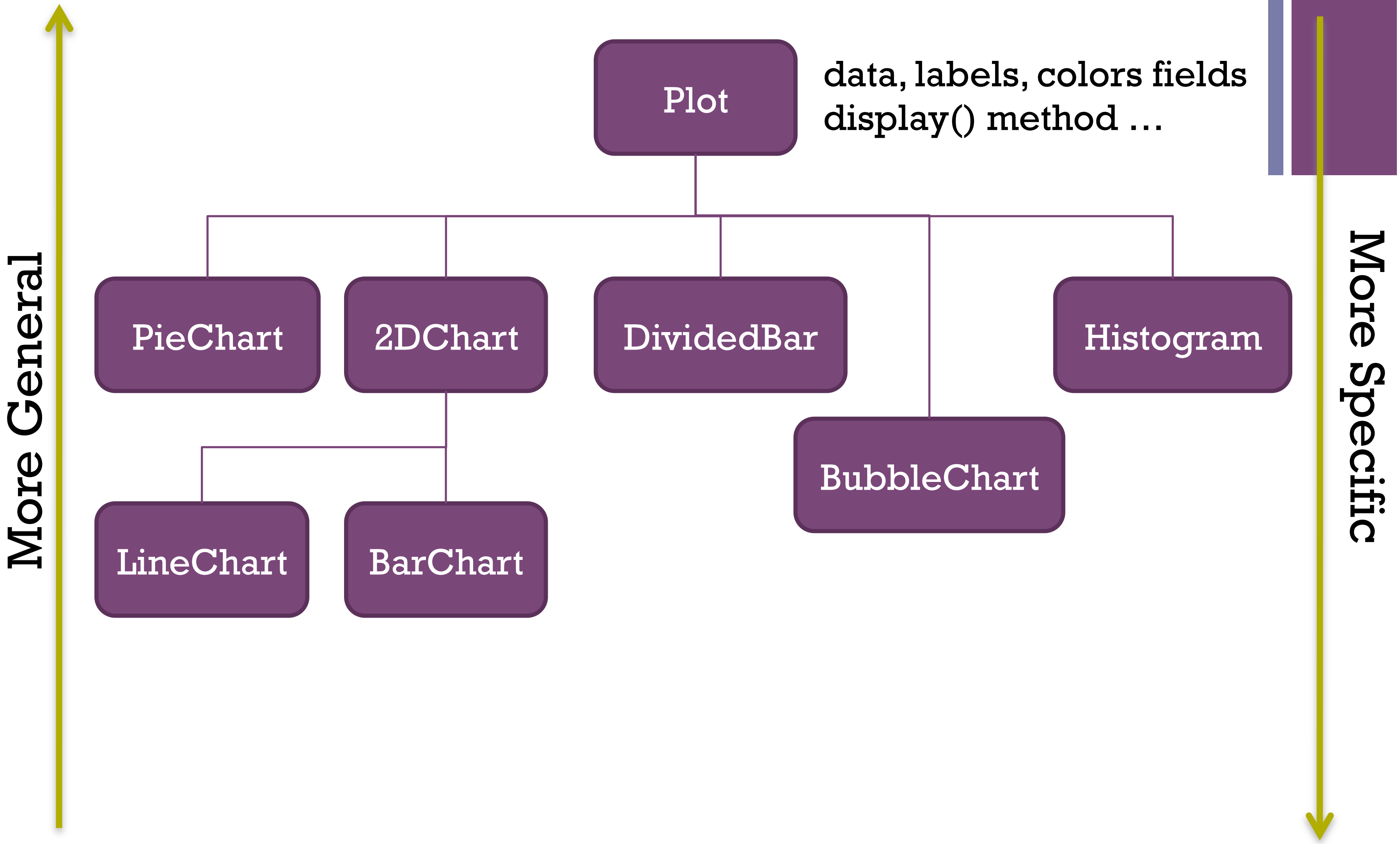
Graphics3.pde

+ Creating a set of Plot Classes

- All have…
  - data
  - labels
  - colors
  - A display()
  - …
- Implementation varies from class to class

+

Plot Object Hierarchy



How could you change the Pie chart to extend from a Plot class?