Review

- "plain text file"
- loadStrings()
- split()
- splitTokens()
- selectInput()
- println(), float(), int(), …
  - can take an array argument, will return an array
  - easy way to convert an array of Strings to an array of floats or ints
- Examples of data visualization
- Assignment 7

TA Schedule

- Emily Levine Friday 2-4, Intro Lab, Intro to CS
- Daisy Sheng Wednesday 12-2, Intro Lab, Intro to CS

- Watch the following for more announcements
  - http://wiki.roboteducation.org/BMC_TA_Schedule

Interactivity

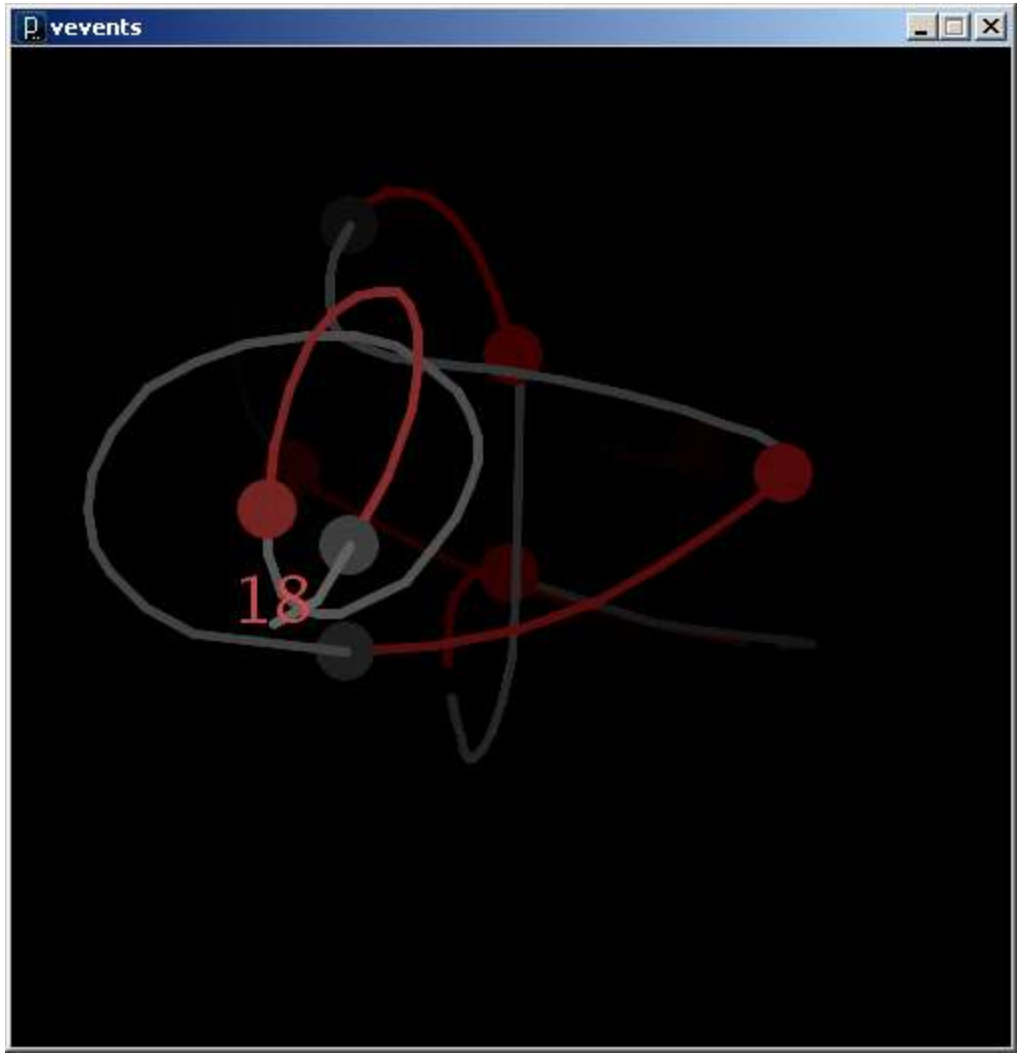
- Mouse Events
  - mousePressed()
  - mouseReleased()
  - mouseClicked()
  - mouseMoved()
  - mouseDragged()

- Keyboard Events
  - keyPressed()
  - keyReleased()
  - keyTyped()

- Mouse System Variables
  - mouseX
  - mouseY
  - pmouseX
  - pmouseY
  - mouseButton
  - mousePressed

- Keyboard System Variables
  - key
  - keyCode
  - keyPressed

vevents.pde



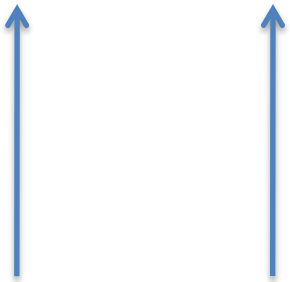
```
void setup() {
  size(500,500);
  smooth();
  textSize(32);
  textAlign(CENTER);
  strokeWeight(10);
  ellipseMode(CENTER);
  background(0);
}

void draw() {
  // Fade existing background to black
  float fade = 0.98;

  loadPixels();

  for (int i = 0; i < pixels.length; i++) {
    color p = pixels[i];
    pixels[i] = color(fade*red(p), fade*green(p), fade*blue(p));
  }

  updatePixels();
}
```

vevents.pde

```
void mousePressed()  {
  // Draw red circle when mouse pressed (mouse down)
  fill(255,95,95);
  noStroke();
  ellipse(mouseX, mouseY, 30, 30);
}

void mouseReleased() {
  // Draw filled gray circle when mouse released (mouse up)
  fill(127);
  noStroke();
  ellipse(mouseX, mouseY, 30, 30);
}

void mouseClicked()  {
  // Draw unfilled white circle to signify a mouse click
  noFill();
  stroke(255);
  ellipse(mouseX, mouseY, 40, 40);
}
```

vevents.pde

```
void mouseMoved() {
  // Draw light gray line when mouse moved
  stroke(192);
  line(pmouseX, pmouseY, mouseX, mouseY);
}

void mouseDragged()  {
  // Draw red line when mouse dragged
  stroke(255,95,95);
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```



Holds values of mouseX and mouseY the last time this event handler was invoked.

mousePressed() vs. mousePressed

- mousePressed()
  - A function (event handler) called by Processing when the mouse is pressed (goes down)

- mousePressed
  - A system variable (boolean) that is
    - true when the mouse is down and
    - false when the mouse is up

mousePressed() vs. mousePressed

```
void setup() {
  size(500, 500);
  frameRate(2);
}
```

```
void draw() {
  if (mousePressed == true) {
    println("mousePressed is true");
  }
}
```

Use the mousePressed system variable when you need to test the state of the mouse while in draw().

```
void mousePressed() {
  println("mousePressed() was called");
}
```

Use the mousePressed() function when you want the pressing of the mouse to initiate an action.

```
void keyPressed() {
  // Print red key code at mouse location on key press
  fill(255,95,95);
  text(keyCode, mouseX, mouseY);
}

void keyReleased() {
  // Print gray key code at mouse location on key released
  fill(127);
  text(keyCode, mouseX, mouseY);
}

void keyTyped() {
  // Draw character typed in upper left corner when key typed
  fill(255);
  text(key, 10, 35);
}
```

vevents.pde

keyCode vs. key

- System Variables

key

- A built-in variable that holds the char that was just typed with the keyboard

keyCode

- A built-in variable that holds the numeric code for the keyboard key that was typed

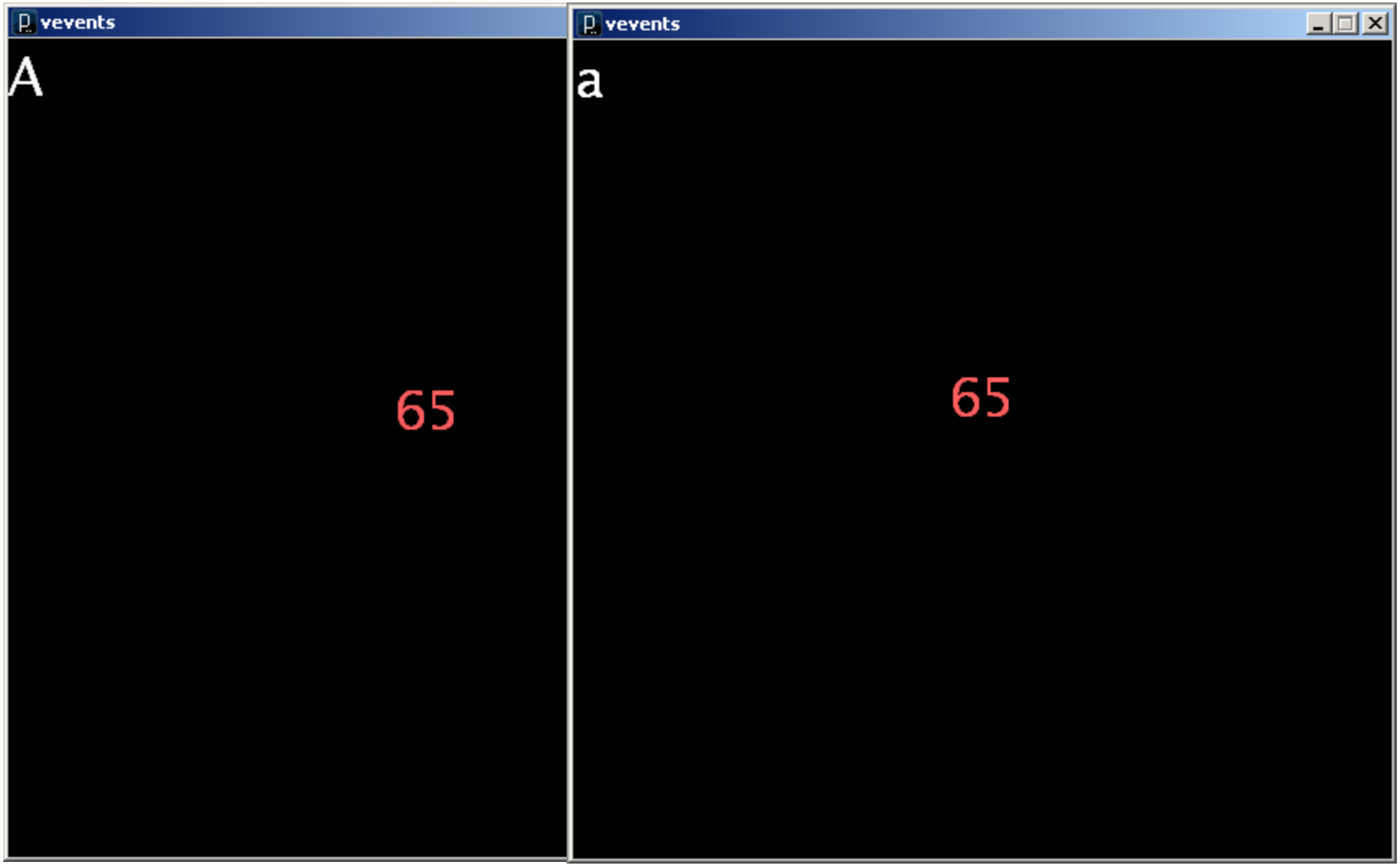
All built-in keyboard interaction functions …

- Set *keyCode* to the integer that codes for the keyboard key
- Set *key* to the character typed
- All keyboard keys have a *keyCode* value
- Not all have a *key* value

keys and keyCodes

Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec
(nul)	0	(dc4)	20	(	40	<	60	P	80	d	100	x	120
(soh)	1	(nak)	21	)	41	=	61	Q	81	e	101	y	121
(stx)	2	(syn)	22	*	42	>	62	R	82	f	102	z	122
(etx)	3	(etb)	23	+	43	?	63	S	83	g	103	{	123
(eot)	4	(can)	24	,	44	@	64	T	84	h	104	|	124
(enq)	5	(em)	25	-	45	A	65	U	85	i	105	}	125
(ack)	6	(sub)	26	.	46	B	66	V	86	j	106	~	126
(bel)	7	(esc)	27	/	47	C	67	W	87	k	107	(del)	127
(bs)	8	(fs)	28	0	48	D	68	X	88	l	108		
(ht)	9	(gs)	29	1	49	E	69	Y	89	m	109		
(nl)	10	(rs)	30	2	50	F	70	Z	90	n	110		
(vt)	11	(us)	31	3	51	G	71	[	91	o	111		
(np)	12	(sp)	32	4	52	H	72	\	92	p	112		
(cr)	13	!	33	5	53	I	73	]	93	q	113		
(so)	14	"	34	6	54	J	74	^	94	r	114		
(si)	15	#	35	7	55	K	75	_	95	s	115		
(dle)	16	$	36	8	56	L	76	`	96	t	116		
(dc1)	17	%	37	9	57	M	77	a	97	u	117		
(dc2)	18	&	38	:	58	N	78	b	98	v	118		
(dc3)	19	'	39	;	59	O	79	c	99	w	119		

keyCode : The physical key on the keyboard
key : The encoded character



Models of Interactivity

- Reacting when the mouse is over an item
  1. Draw the item differently
  2. Add a new item to the drawing
  3. …

- Reacting when the mouse clicks on an item
  1. Change the state of the item
  2. Show more information about the item
  3. Start an animation
  4. …

When is the mouse over an item?

- You must test the mouse position against the item location and size

```
// Test if the mouse is over a circle
// with center (x, y) and radius
if ( dist( mouseX, mouseY, x, y ) < radius )
{
    …
}
```

```
// Test if the mouse is over a rectangle
// with upper-left corner (x, y) and width w, height h
if (    mouseX >= x   && mouseY >= y
     && mouseX <= x+w && mouseY <= y+h )
{
    …
}
```

Model: Draw differently on mouse-over

```
// mouseOver1.pde

float x = 250;       // Circle parameters
float y = 250;
float radius = 50;

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

void draw() {
  // 1. Clear the background
  background(255);

  // 2. Set fill color based on whether or not
  //    the mouse is over the circle.
  if ( dist( mouseX, mouseY, x, y ) < radius ) {
    fill( 255, 0, 0 );
  } else {
    fill( 0, 255, 0 );
  }

  // 3. Draw the shape using the current fill color
  ellipse( x, y, 2*radius, 2*radius );
}
```

Model: Add to drawing on mouse-over

```
// mouseOver2.pde

float x = 250;       // Circle parameters
float y = 250;
float radius = 50;

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

void draw() {
  // 1. Clear the background
  background(255);

  // 2. Draw the shape using the current fill color
  fill(255, 0, 0);
  ellipse( x, y, 2*radius, 2*radius);

  // 3. Add tooltip if the mouse is over the circle.
  if ( dist( mouseX, mouseY, x, y ) < radius ) {
    fill(0);
    text("I'm over the circle", mouseX, mouseY);
  }
}
```

```
// mouseHover1.pde
// Implement a tooltip on mouse hover
float x = 250;          // Ellipse parameters
float y = 250;
float radius = 50;
float hoverStart;       // Keep track of hover start time

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

void draw() {
  // 1. Clear the background
  background(255);

  // 2. Draw shape using the current fill color
  fill(255, 0, 0);
  ellipse( x, y, 2*radius, 2*radius);

  // 3. Add tooltip if the mouse is over ellipse
  if ( dist( mouseX, mouseY, x, y ) < radius ) {

    // 4. Only show if hover time is > 1 second
    if ( millis() - hoverStart > 1000 ) {
      fill(0);
      text("I'm over the ellipse", mouseX, mouseY);
    }
  }
}

void mouseMoved() {
  // 5. Reset hover start time on each move over ellipse
  if ( dist( mouseX, mouseY, x, y ) < radius ) {
    hoverStart = millis();
  }
}
```

Model:

Add tooltip on mouse-hover

Model: Change state when clicked

```
// mousePressed1.pde

float x = 250;          // Ellipse parameters
float y = 250;
float radius = 50;
color fillColor = color(255);

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

void draw() {
  // 1. Clear the background
  background(255);

  // 2. Draw the shape using the current fill color
  fill( fillColor );
  ellipse( x, y, 2*radius, 2*radius);
}

void mousePressed() {
  // 3. Change fill color when pressed on ellipse. Reset when not on ellipse
  if ( dist( mouseX, mouseY, x, y ) < radius ) {
    fillColor = color(255, 0, 0);
  } else {
    fillColor = color(255);
  }
}
```

Model: Dragging an item

```
// mouseDragged1.pde
float x = 250;              // Ellipse parameters
float y = 250;
float radius = 50;
boolean dragging = false;  // State of dragging

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

void draw() {
  background(255);                          // 1. Clear the background
  ellipse( x, y, 2*radius, 2*radius );      // 2. Draw shape at the current location
}

void mousePressed() {                       // 3. Change dragging state if mouse
  if ( dist( mouseX, mouseY, x, y ) < radius ) {     // pressed on ellipse
    dragging = true;
  }
}

void mouseReleased() {
  dragging = false;                         // 4. Stop dragging when mouse released
}

void mouseDragged() {
  if ( dragging == true ) {                 // 5. If dragging, move ellipse under mouse
    x = mouseX; y = mouseY;
  }
}
```

Encapsulate interactive behavior in a class

- Store all parameters as object fields
- Delegate all behavior to classes from top-level functions called by Processing
  - draw(), … mousePressed(), … keyTyped(), …
- Allows each class to react differently, implementing unique behavior

Starting Point – A Simple Circle Class

```
// Circle class
class Circle {

  float x;
  float y;
  float radius;
  String name;

  Circle(float tx, float ty, float tr, String tn)
  {
    x = tx;
    y = ty;
    radius = tr;
    name = tn;
  }

  void draw() {
    fill( 255, 0, 0 );
    ellipse( x, y, 2*radius, 2*radius);
  }
}
```

Create and Manage an Array of Circle Classes

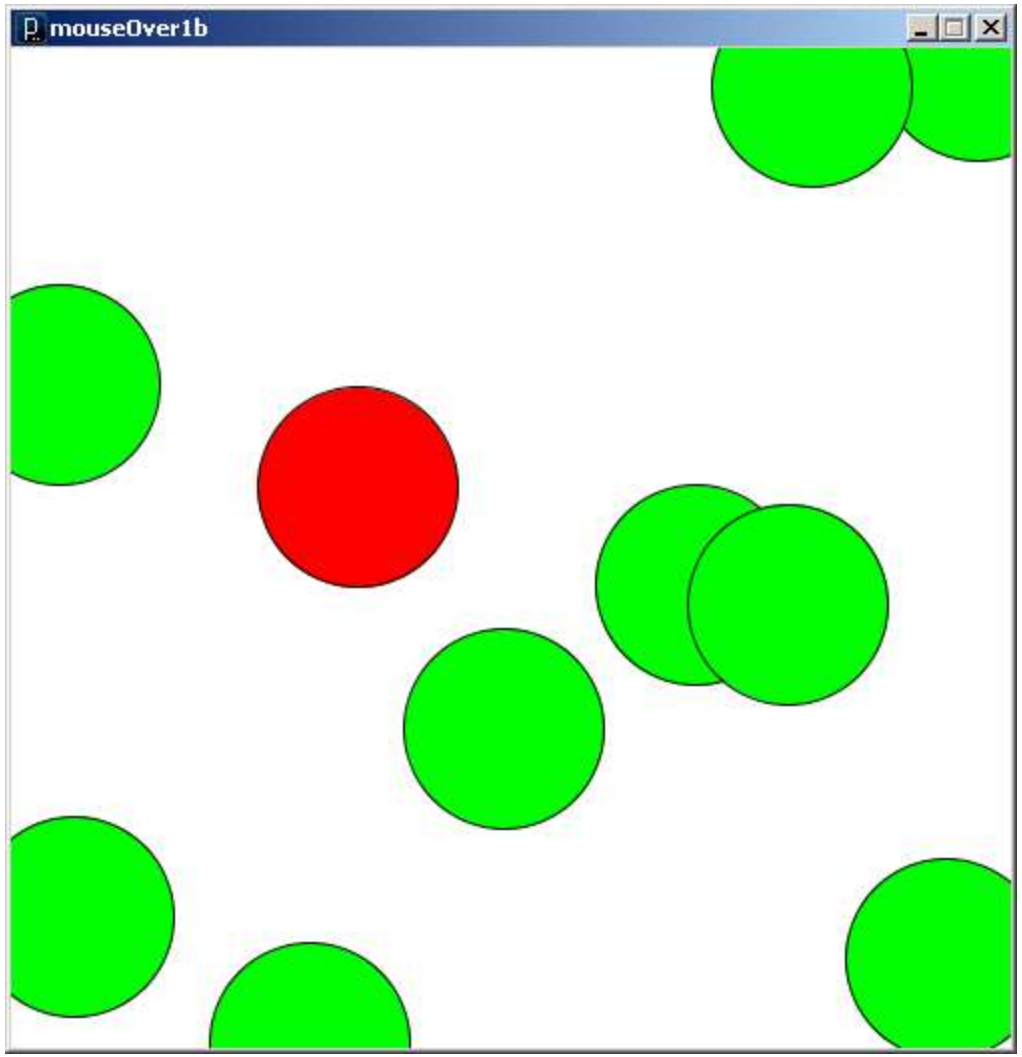
```
// mouseOver1b.pde
Circle[] circs;      // 1. Declare array variable to hold all circles

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

  // 2. Create an array to hold Circles
 circs = new Circle[10];

  // 3. Create Circles and assign to array locations
  for (int i=0; i<circs.length; i++) {
    String name = "Circle" + i;
    float x = random(width);
    float y = random(height);
    circs[i] = new Circle(x, y, 50, name);
  }
}

void draw() {
  background(255);

  // 4. Delegate all draw behavior to Ellipse objects
  for (int i=0; i<circs.length; i++) {
    circs[i].draw();
  }
}
```

mouseOver1b.pde

Expand Circle Class with mouse-over behavior

```
// Circle class
class Circle {

  // stuff removed
  …

 void draw() {

    // 5. Set fill color based on whether or not
    //    the mouse is over circle.
    if ( dist( mouseX, mouseY, x, y ) < radius ) {
      fill( 255, 0, 0 );
    } else {
      fill( 0, 255, 0 );
    }

    // 6. Draw the shape using the current fill color
    ellipse( x, y, 2*radius, 2*radius);
  }
}
```

mouseOver1b.pde



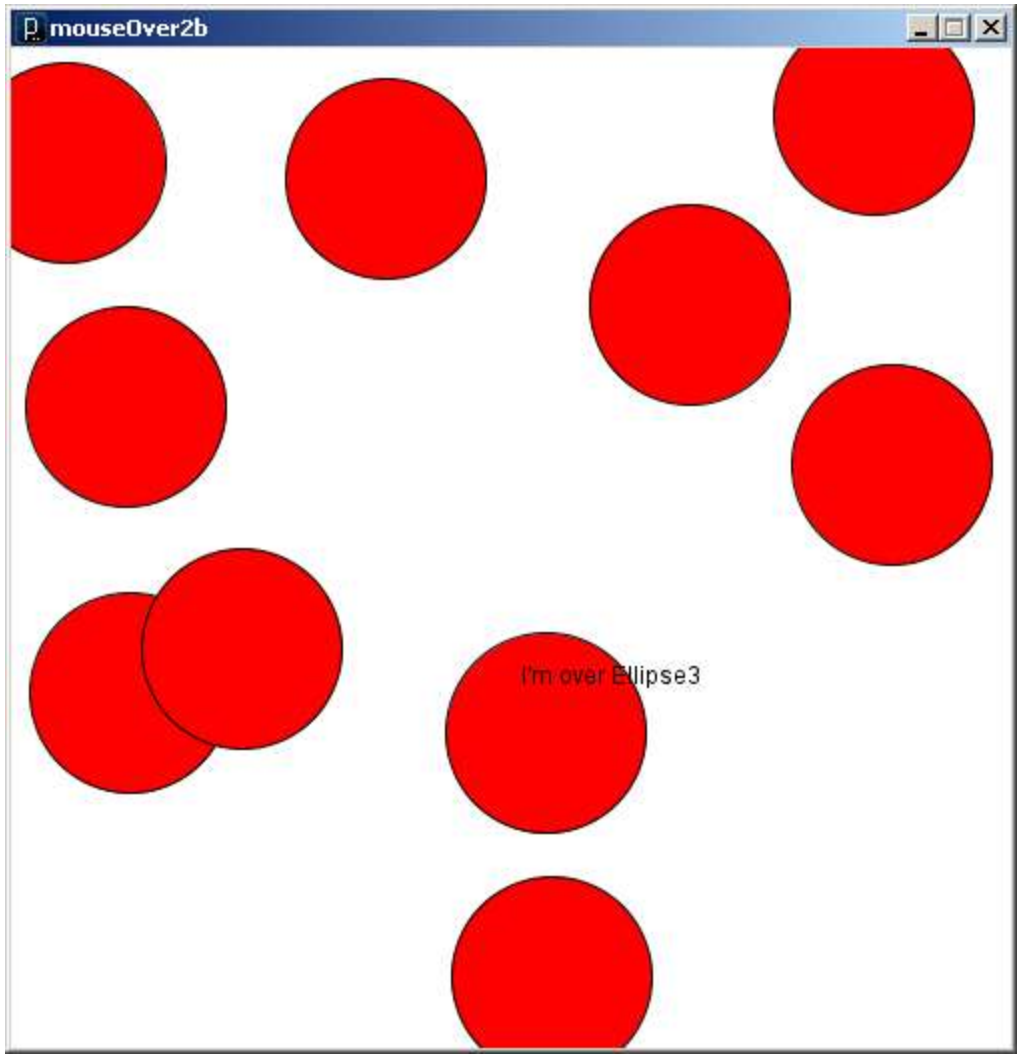
mouseOver1b.pde

Expand Circle Class with different mouse-over behavior

```
// Circle class
class Circle {

  // stuff removed
  …

void draw() {

    // 5. Draw the shape using the current fill color
    fill(255, 0, 0);
    ellipse( x, y, 2*radius, 2*radius);

    // 6. Add tooltip if the mouse is over ellipse
    if ( dist( mouseX, mouseY, x, y ) < radius ) {
      fill(0);
      text("I'm over " + name, mouseX, mouseY);
    }
  }
}
```

The main program is identical to mouseOver1b.pde

mouseOver2b.pde

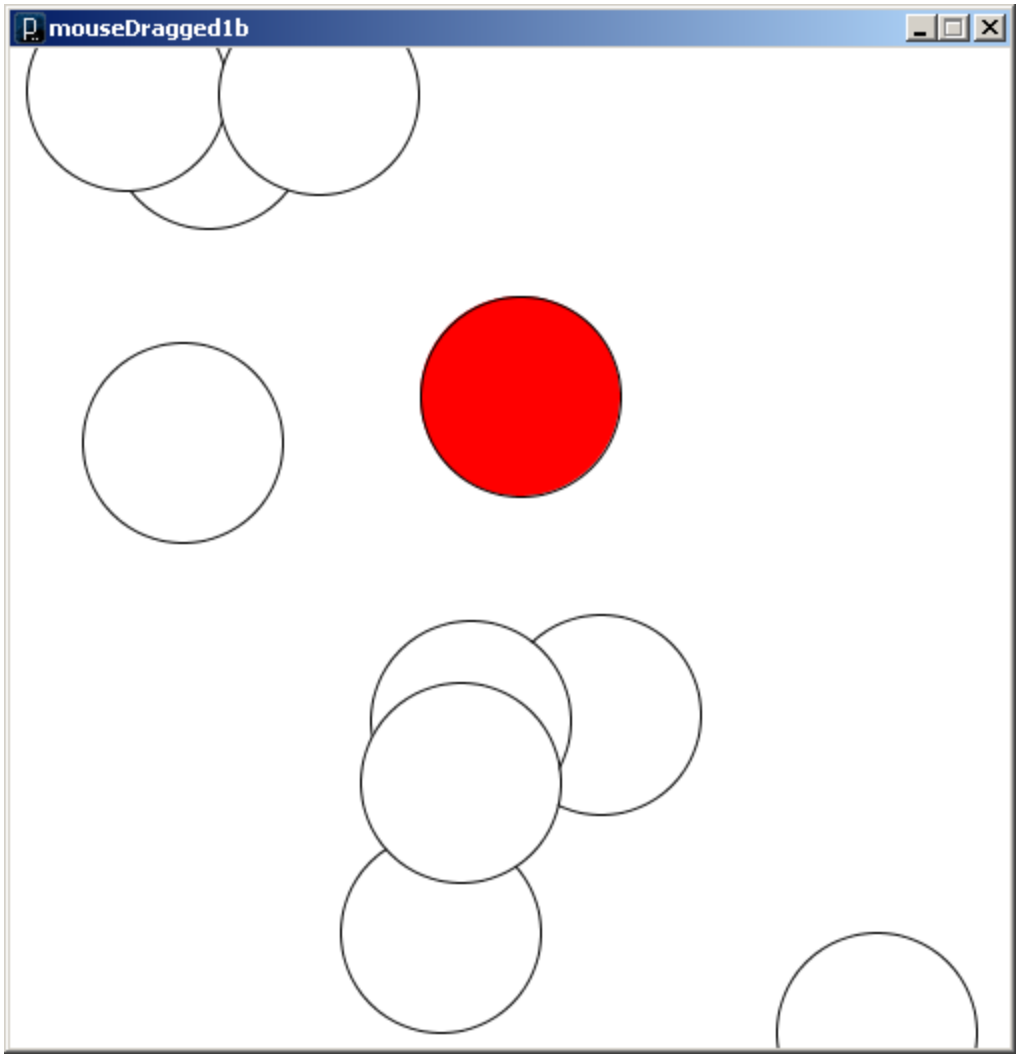


mouseOver2b.pde

Following the same method for expansion…

- Delegate behavior to all object event handlers
  - draw()
  - mousePressed()
  - mouseReleased()
  - mouseDragged()

mouseDragged1b.pde



mouseDragged1b.pde

Object Oriented Programming – More than Classes

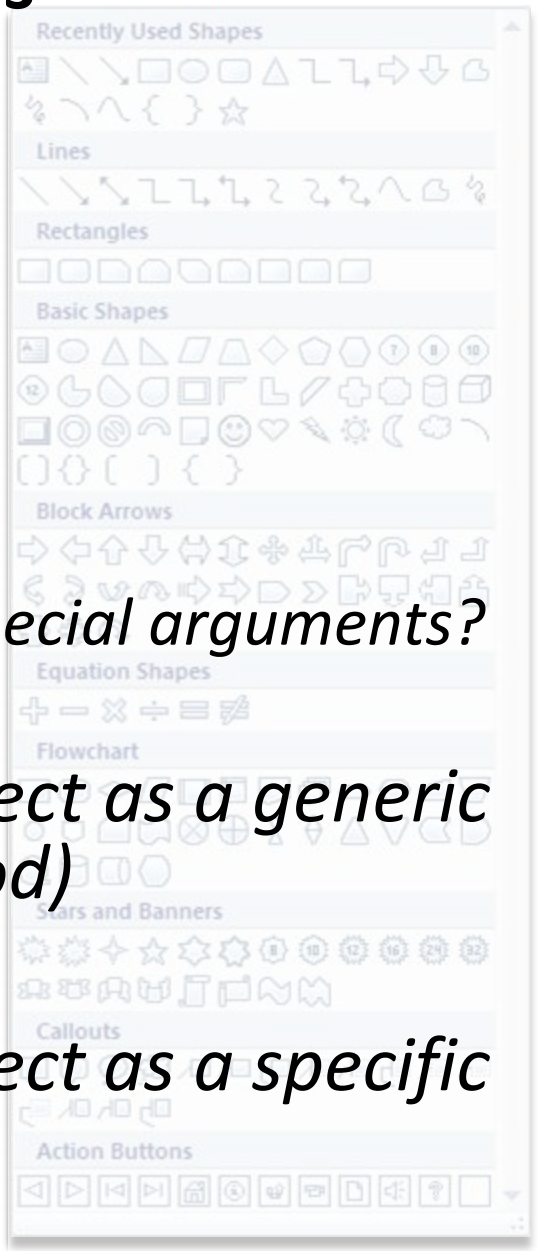
- Encapsulation
  - Classes encapsulate **state** (fields) and **behavior** (methods)

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

Creating a set of general Graphic Object Classes

- All can 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

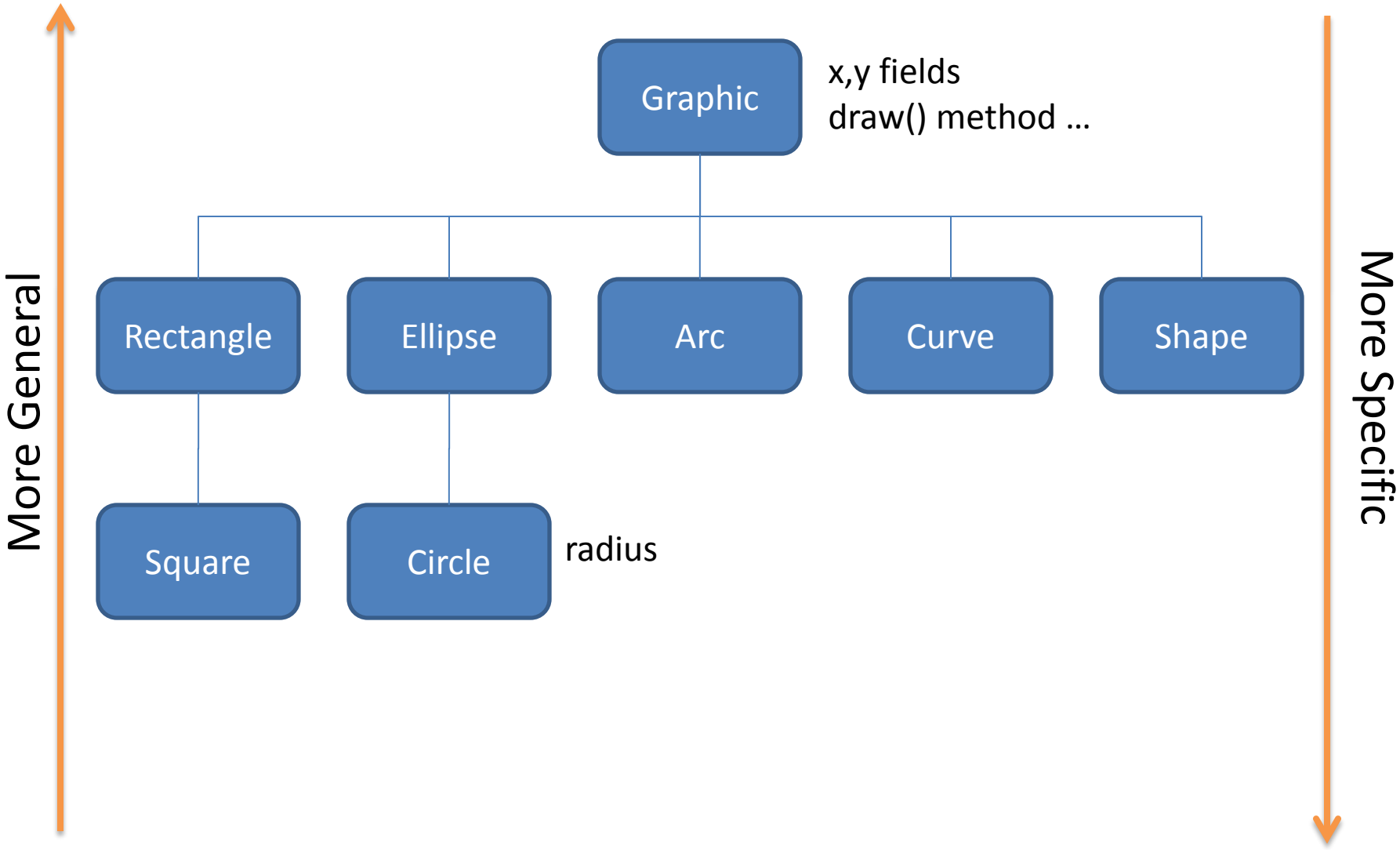
  How would you hold all your objects?
  - Array?
  - ArrayList or HashMap?

  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

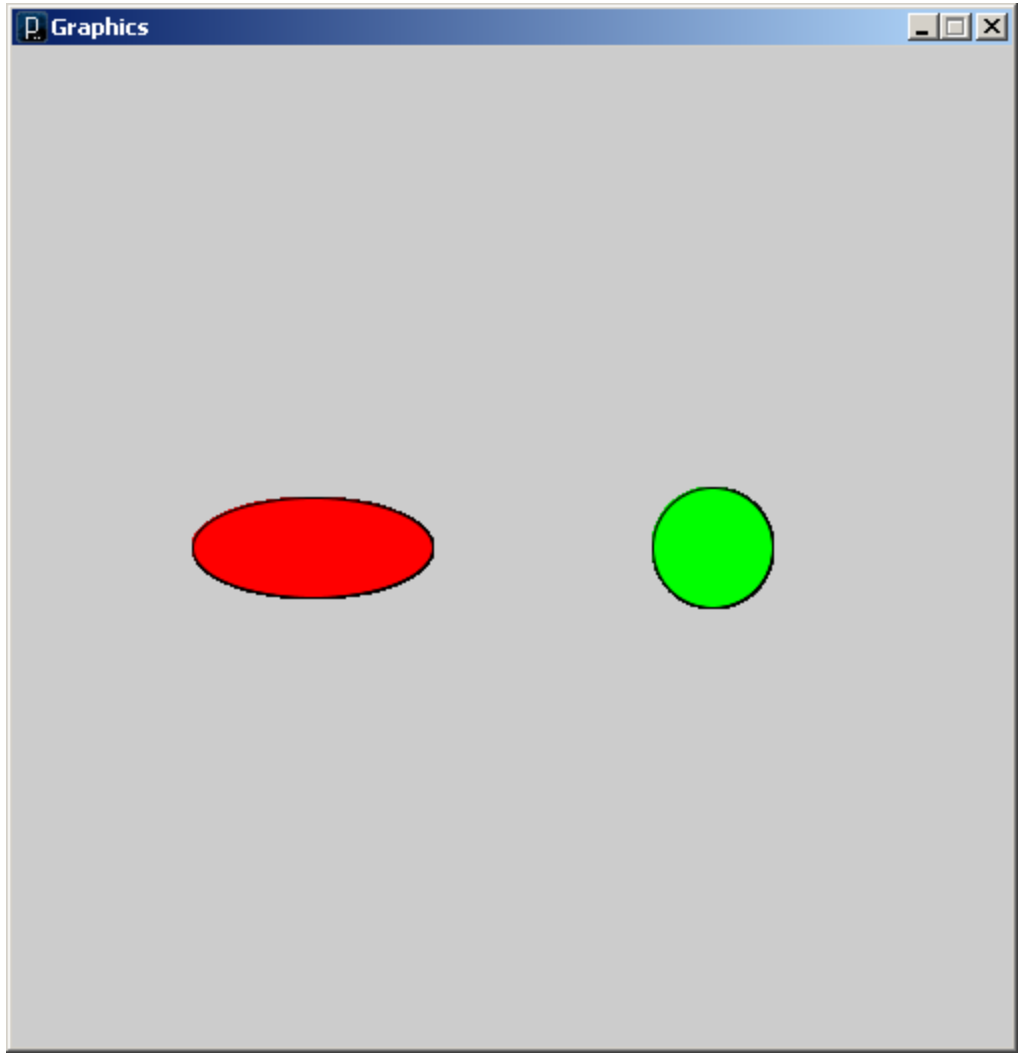
```
// Ellipse base class
class Ellipse {

  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 r) {
    super(x, y, 2*r, 2*r);

    // Circles are always blue
    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.

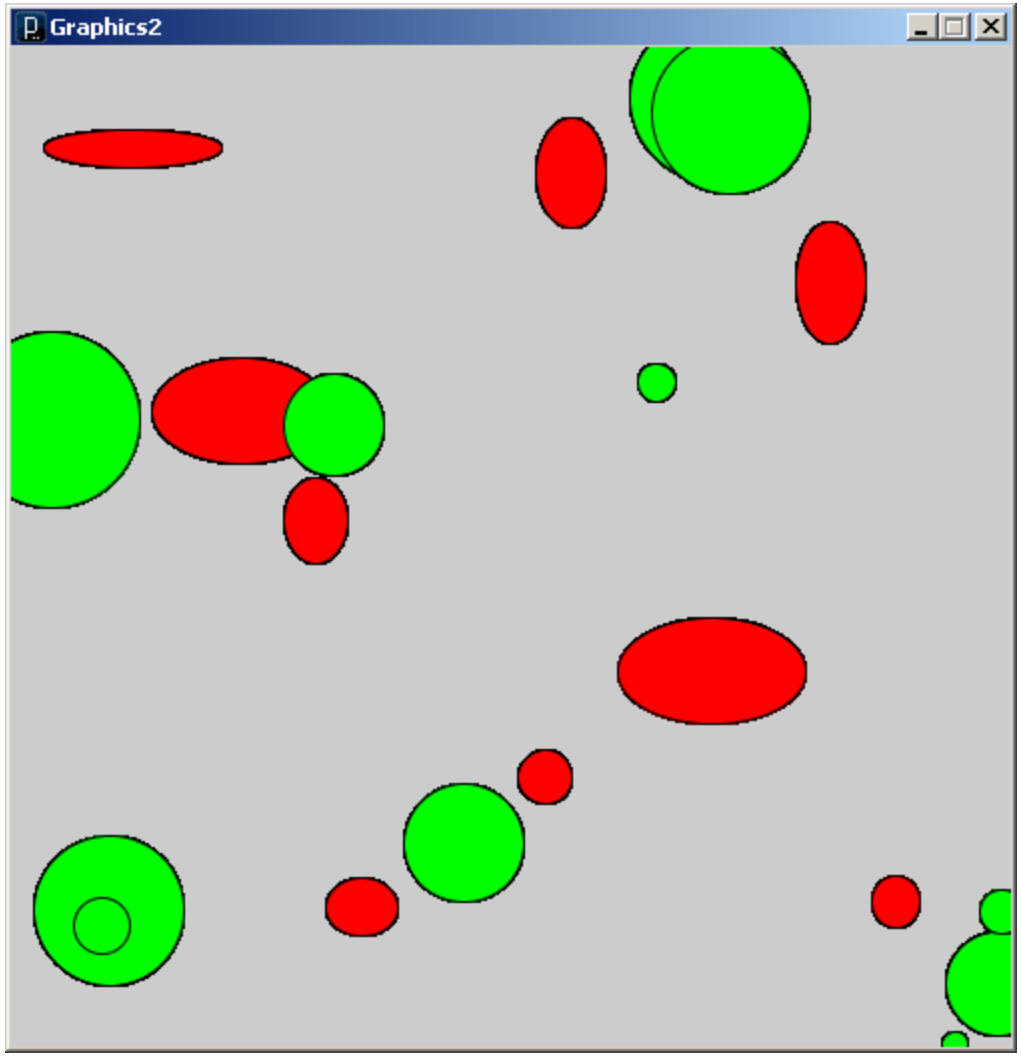
Graphics.pde

```
// Graphics
Ellipse e = new Ellipse(150, 250, 120, 50);
Circle  c = new Circle(350, 250, 30);

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, 0.5*w);
    }
  }
}

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



Ellipses and Circles in the same array!

Graphics2.pde

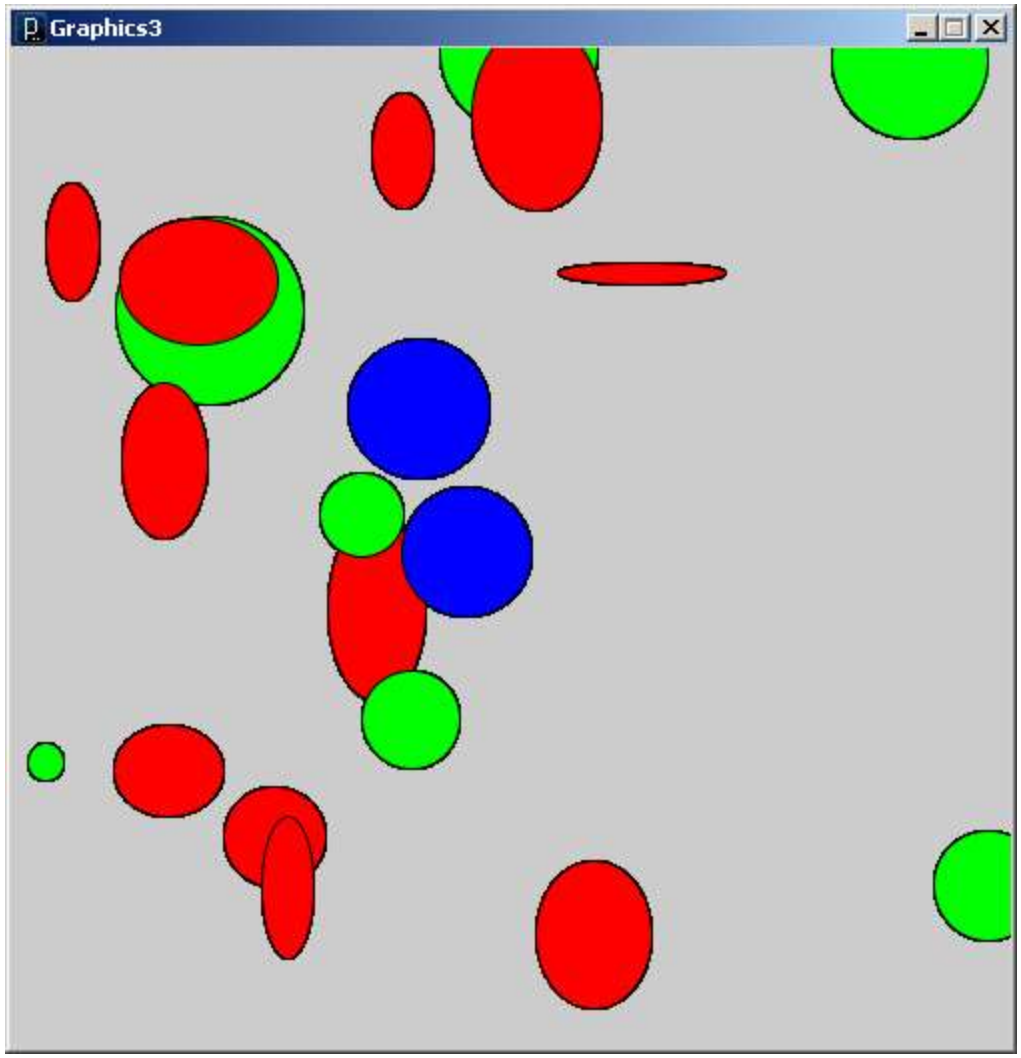
```
// Ellipse base class
class Ellipse {

  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 r) {
    super(x, y, 2*r, 2*r);

    // Circles are always blue
    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