Introduction to Objects

CS 110

Object Oriented Programming

- Objects are software bundles that wrap up all semantically related variables and functions.
    - Object variables are called fields
    - Object functions are called methods
- Objects are said to Encapsulate (hide) its detail
    - How an object method is implemented is not important
    - What it does is important
- Objects can be created, named and referenced with variables
    - Very similar to standard data types
- An object's individual fields and methods are accessed using syntax called dot-notation

Recall … Images

`loadImage(`*`filename`*`);`

- Loads an image from a file in the *data* folder in sketch folder.
- Must be assigned to a variable of type PImage.

`image(`*`img, X, Y, [X2, Y2]`*`);`

- Draws the image *img* on the canvas at X, Y
- Optionally fits image into box X,Y and X2,Y2

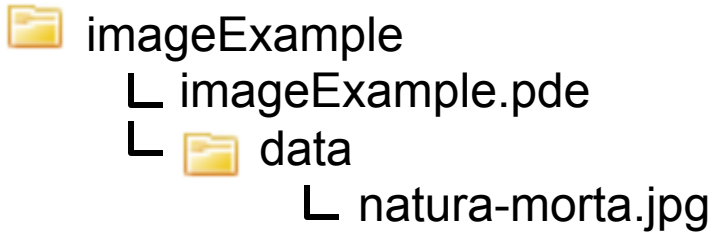
`imageMode(CORNER);`

- X2 and Y2 define width and height.

`imageMode(CORNERS);`

- X2 and Y2 define opposite corner.

Image Example




```
PImage img;

void setup()
{
  size(500, 400);
  img = loadImage("natura-morta.jpg");
  image(img, 50, 40);
}
```

loadImage is a function that reads image data from a file, stores it in a new PImage object, and returns the new PImage object.

The image function takes a variable of type PImage as its first argument and renders it on your sketch.

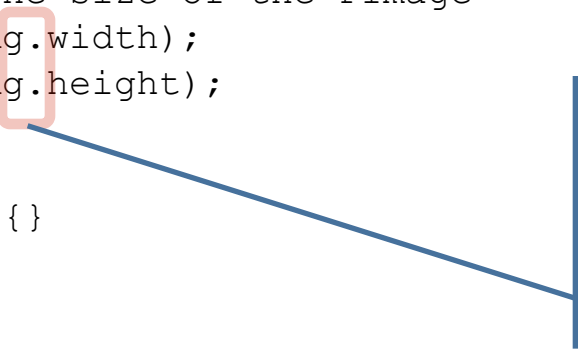
The PImage Object

- Fields
  - width — *image width*
  - height — *image height*
  - pixels[] — *1D array holding all image pixels*
- Methods
  - loadPixels() — *fill the pixels[] array with image pixels*
  - updatePixels() — *copy pixels in pixels[] array back to image*
  - get(*x, y*) — *reads a pixel at position x, y*
  - set(*x, y, color*) — *set the color at position x, y*
  - save(*path*) — *saved an image to a file*
  - …
- Related Functions
  - loadImage(*path*) — *create a new PImage and init with image file*
  - createImage(*w, h, form*) — *create a new empty Pimage object*
  - image(*img, x, y*) — *draw a PImage to a sketch*

http://processing.googlecode.com/svn/trunk/processing/build/javadoc/core/index.html

Image Example

```
// imageExample2

PImage img;

void setup()
{
  size(500, 400);
  img = loadImage("natura-morta.jpg");
  image(img, 50, 40);
}

void mousePressed() {
  // Print the size of the PImage
  println(img.width);
  println(img.height);
}

void draw() {}
```



Dot-notation …
To access the fields and methods within an object, join the object and field/method using a dot.

The String Object

- Fields
  - …
- Methods
  - equals(*anotherString*)
  - length()
  - substring()
  - toLowerCase()
  - toUpperCase()

http://download.oracle.com/javase/1.4.2/docs/api/

String Method Examples

```
String s;

s = "BrynMawr";
println(s);
println( s.length() );

println( s.substring(4) );
println( s.substring(3,7) );

println( s.toUpperCase() );
println( s.toLowerCase() );
```

```
BrynMawr
8
Mawr
nMaw
BRYNMAWR
brynmawr
```

Defining Your Own Object with Classes

- Classes are blueprints or prototypes for new objects
- Classes define all field and method declarations

    … which are repeated for each new object created

- Classes DO NOT set the data values stored in fields

    … but they likely determine how

- Using a class to create a new object is called *instantiating* an object

    … creating a new object instance of the class

- Classes often model real-world items

Defining Your Own Objects with Classes

```
// Defining a new class of object

class MyObjectName {

  // All field variable declarations go here;

  // Define a special function-like statement called
  // the class's Constructor.
  // It's name is same as object class name,
  // with no return value.

  MyObjectName( optional arguments ) {

    // Perform all initialization here

  }

  // Declare all method functions here.
}
```

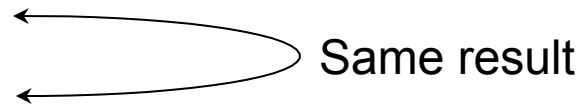
```
// A Ball Class
class Ball {
  // Fields
  float ay = 0.2;     // y acceleration (gravity)
  float sx;    // x position
  float sy;    // y position
  float vx;    // x velocity
  float vy;    // y velocity

  // Constructor
  Ball() {
    sx = random(0.0, width);
    sy = random(0.0, 10.0);
    vx = random(-3.0, 3.0);
    vy = random(0.0, 5.0);
  }

  // Methods
  void update() {
    // Move ball
    sx += vx;
    sy += vy;
    vy += ay;

    // Bounce off walls and floor
    if (sx <= 10.0 || sx >= (width-10.0)) vx = -vx;
    if (sy >= (height-10.0) && vy > 0.0) vy = -0.9*vy;
  }

  void draw() {
    ellipse( sx, sy, 20, 20);
  }
}
```

Creating New Objects with Classes

- To create a new instance of an object, use the ***new*** keyword and call the object Constructor

```
MyObjectName ob = new MyObjectName(42);
```

```
String s = new String("Blah");
String s = "Blah";
```




```
Ball b = new Ball();
```

Use the Ball class

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

```
// bounce4
Ball[] balls = new Ball[20];

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();
  }
}

void draw() {
  background(255);

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

Declare an array of Balls.

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

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

Comparing Declarations and Initializers

```
int       i;
int       j   = 3;
float     f   = 0.1;
float[]   f2  = new float[20];
String    s1  = "abc";
String    s2  = new String("abc");
Ball      b   = new Ball();
Ball[]    b2  = new Ball[20];
```