Processing/Java Syntax

Classes, objects, arrays, and ArrayLists
Processing and Java

• Java is relentlessly object-oriented
  • Among other things, this means that all methods must occur within classes
• Processing puts your declarations and methods inside an “invisible” Java class
• You can create additional classes to use alongside this provided “invisible” class
• All the syntax in this lecture can be used in either Processing or in Java
• Arrays and ArrayLists are from Java, and are therefore available in Processing
• The drawing methods, however, are only available in Processing
Classes and objects

- A class defines objects
  - If a class is a cookie cutter, objects are cookies
  - If a class is a housing blueprint, objects are houses
- The purpose of an object is to hold information and methods for manipulating that information
  - **Example:** A Customer class with fields customerId, name, address, orders, etc. and methods fillOrder, sendBill, etc.
  - **Example:** An Order class with fields itemId, description, cost, numberInStock, etc., and methods updateNumberInStock, orderMore, etc.
  - **Example:** A String class with methods equals, beginsWith, toLowerCase, etc.
Simple classes

- You define a class with the syntax
  ```java
  class NameOfClass {
      // Fields (variables) of class
      // Methods of class
  }
  ```
  - Class names should always begin with a capital letter

- You create an object of that class and assign it to a variable with the syntax
  ```java
  NameOfClass nameOfObject = new NameOfClass();
  ```
  - Objects created in this way are all identical, until you call a `setXXX` method to change the values of the fields of an object
  - Later in this course we will see how to create non-identical objects

- You talk to the object with the syntax
  ```java
  nameOfObject.nameOfMethod(parameters);
  ```
Example class definition

```
class SmileyFace {
    private float size = 30.0; // this is a “field”

    void setSize(float s) { // method to change a field’s value
        size = s;
    }

    void drawMe(float x, float y) { // an ordinary method
        stroke(0);
        fill(255, 255, 0);
        ellipse(x, y, size, size); // some things depend on “size”
        fill(0);
        ellipse(x - size / 5, y - 4, 5, 7); // some don’t
        ellipse(x + size / 5, y - 4, 5, 7);
        noFill();
        arc(x, y, 0.625 * size, 0.625 * size,
            radians(30), radians(150));
    }
}
```
Example object creation and use

```java
void setup() {
    size(200, 100);
    background(255);
}

void draw() {
    SmileyFace face1 = new SmileyFace();
    SmileyFace face2 = new SmileyFace();
    face1.drawMe(50, 50);
    face2.setSize(60);
    face2.drawMe(100, 50);
    face1.drawMe(150, 50);
}
```
Dot notation

- Once we have created some objects, we “talk to” them using **dot notation**
  - **Syntax:** `theObject.methodName(parameters);`
  - **Example:** `face1.drawMe(50, 50);`
    - Here we are telling the object `face1` to draw itself, and using the parameters to tell it *where* to draw itself
    - Similarly, the call `face2.setSize(60);` tells the object `face2` to change the value in its *size* field
      - This does not affect the *size* field of object `face1`
  - If a method in a class returns a value, we can use that method to ask an object for that value
    - **Example method:** `float getSize() { return size; }`
    - **Example use:** `if (face1.getSize() > 10) {...}`
  - If a field in a class is not marked **private**, we can use dot notation to access and/or change the value of that field
    - This is *poor style*—an object should control its own state
The preceding example of objects is simple and kind of silly

- You could just write a `drawSmiley(float x, float y, float size)` method

- The `size` field is marked `private`:
  ```java
  private float size = 30.0;
  ```
  This means that it can only be changed by methods `within this class`

For a more convincing use of objects, you could create a whole lot of smiley face objects that move around the screen, turn red, “eat” other faces, grow bigger or smaller, explode, etc.

- Each object would have its own fields
- The fields of each object would determine its current `state`

To manage a large number of objects, you also need some way to keep track of them all
Arrays and ArrayLists

- Java provides a large number of ways to keep track of a collection of objects
- The two most commonly used are arrays and ArrayLists

Arrays:
- Are easier to use
- Are much more efficient (in general)
- Can hold primitives (floats, integers, booleans, chars) or objects
- Use a simpler, special-purpose syntax
- The size is specified when the array is created, and cannot be changed

ArrayLists:
- Use the more complicated object syntax
- Require more space and more computer time
- Can only hold objects, not primitives
  - (Java tries to hide this limitation, but it doesn’t always work)
- The size can be changed, by adding or removing objects
ArrayList is a built-in class

- You have to import java.util.ArrayList;
- Create an ArrayList with
  ArrayList<\texttt{ObjectType}> \texttt{variable} = new ArrayList<\texttt{ObjectType}>();
  - The \texttt{ObjectType} is the type of thing you want to put into the ArrayList, for example, Customer or String
  - In Java 8 (but not yet in Processing), you can reduce the redundancy somewhat by saying
    ArrayList<\texttt{ObjectType}> \texttt{variable} = new ArrayList<>();
- Some methods (assuming \texttt{al} is an ArrayList):
  - \texttt{al.add(\texttt{object})}; adds \texttt{object} (which must be of the correct type) to the end of the list \texttt{al}
  - \texttt{al.add(index, \texttt{object})}; adds the \texttt{object} to position \texttt{index} (0 based) of \texttt{al}
  - \texttt{al.get(index)} returns the object at location \texttt{index} of \texttt{al}
  - \texttt{al.set(index, \texttt{object})}; replaces the object at \texttt{index} in \texttt{al} with \texttt{object}
  - \texttt{al.remove(\texttt{object})}; removes \texttt{object} from \texttt{al}
  - \texttt{al.remove(index)}; removes the object at location \texttt{index} from \texttt{al}
  - \texttt{al.contains(\texttt{object})} returns true if \texttt{object} is in the list \texttt{al}
class Ball {
    float x, y, dx, dy, radius = 15; // not great style
    color ballColor;

    void setAll(float x, float y, color c) {
        this.x = x;
        this.y = y;
        dx = random(1, 5);
        dy = random(1, 5);
        ballColor = c;
    }

    void draw() {
        if (x < radius || x > width - radius) dx = -dx;
        if (y < radius || y > height - radius) dy = -dy;
        fill(ballColor);
        ellipse(x, y, 2 * radius, 2 * radius);
        x += dx;
        y += dy;
    }
}
Using an `ArrayList` of `Ball`

- `ArrayList<Ball> balls = new ArrayList<Ball>();`

```java
void setup() {
    size(511, 255);
    background(255);
    noLoop(); // comment this line out to get animation
    for (int row = 15; row < 255; row += 20) {
        for (int col = 15; col < 511; col +=20) {
            Ball ball = new Ball();
            if (col > 255) ball.setAll(col, row, color(row, 0, col - 255));
            else ball.setAll(col, row, color(row, 255 - col, 0));
            balls.add(ball);
        }
    }
}

void draw() {
    background(255);
    for (Ball ball : balls) {
        ball.draw(); // Notice form
    }
}
```
Using an `ArrayList` of `Ball`

- `ArrayList<Ball> balls = new ArrayList<Ball>();`

```java
void setup() {
    size(511, 255);
    background(255);
    // noLoop(); // comment this line out to get animation
    for (int row = 15; row < 255; row += 20) {
        for (int col = 15; col < 511; col += 20) {
            Ball ball = new Ball();
            if (col > 255) ball.setAll(col, row, color(row, 0, col - 255));
            else           ball.setAll(col, row, color(row, 255 - col, 0));
            balls.add(ball);
        }
    }
}

void draw() {
    background(255);
    for (Ball ball : balls) {
        ball.draw(); // Notice form
    }
}
```
Arrays

- Here’s how to declare an array (of Balls): `Ball[] balls;`
- Here’s how to create an array: `balls = new Ball[300];`
- Here’s how to do both at once: `Ball[] balls = new Ball[300];`
- Notice:
  - The size of the array is not part of its type
    - This means you can assign an array of a different size to the same variable
    - When you create an array (and only then), you specify its size
  - You access an element of an array by putting the index of the element in brackets, for example, `balls[5]` or `balls[n - 1]`
    - The first index is zero; the last is the array size minus one
    - To find the size of the `balls` array, say `balls.length`
      - `length` isn’t a method; it’s a field of the array object
Two ways to declare arrays

- You can declare more than one variable in the same declaration:
  ```java
  int a[], b, c[], d;  // notice position of brackets
  - a and c are int arrays
  - b and d are just ints
  ```

- Another syntax:
  ```java
  int [ ] a, b, c, d;  // notice position of brackets
  - a, b, c and d are int arrays
  - When the brackets come before the first variable, they apply to all variables in the list
  ```

- But...
  - In Java, we typically declare each variable separately
Using a Ball[] array

```java
Ball[] balls = new Ball[300];

void setup() {
    size(511, 255);
    background(255);
    noLoop(); // comment this line out to get animation
    int index = 0;
    for (int row = 15; row < 255; row += 20) {
        for (int col = 15; col < 511; col +=20) {
            Ball ball = new Ball();
            if (col > 255) ball.setAll(col, row, color(row, 0, col - 255));
            else ball.setAll(col, row, color(row, 255 - col, 0));
            balls[index] = ball;
            index += 1;
        }
    }
}

void draw() {
    background(255);
    for (Ball ball : balls) { // works for both arrays and ArrayLists
        ball.draw(); // Notice form of call
    }
}
```
Two-dimensional arrays

- Java does not actually have “two dimensional arrays,” that is, arrays with both rows and columns
- Instead, the elements of an array may be themselves arrays
- Example syntax:
  ```java
  Ball[][] balls = new Ball[12][25];
  ```
  - Informally, this is an array of 12 rows and 25 columns
  - More precisely, it’s an array containing 12 arrays, where each contained array has 25 elements
  - Reference the elements with `balls[row][column]`
    - In this example, \(0 \leq row < 12\) and \(0 \leq column < 25\)
  - Or get a single (one-dimensional) array with `balls[row]`
Using a Ball[][][] array

- Ball[][][] balls = new Ball[12][25];

void setup() {
    size(511, 255);
    background(255);
    noLoop(); // comment this line out to get animation
    int index = 0;
    for (int i = 0; i < balls.length; i += 1) {
        for (int j = 0; j < balls[0].length; j +=1) {
            Ball ball = new Ball();
            int x = 20 * j + 15;
            int y = 20 * i + 15;
            if (x > 255) ball.setAll(x, y, color(y, 0, x - 255));
            else         ball.setAll(x, y, color(y, 255 - x, 0));
            balls[i][j] = ball;
            index += 1;
        }
    }
}

- In this version, it was easier to compute the ball x,y locations from the array i, j indices (changes in black text above), rather than the other way around
void draw() {
    background(255);
    for (Ball[] row : balls) {
        for (Ball ball : row) {
            ball.draw();
        }
    }
}

- The outer for loop gives us a single row of the array, which is itself an array
- The inner for loop gives the element of the row

Alternatively, we can use indices to get each element of balls

void draw() {
    background(255);
    for (int i = 0; i < balls.length; i += 1) {
        for (int j = 0; j < balls[0].length; j += 1) {
            balls[i][j].draw();
        }
    }
}
From Processing to Java

- Processing is simply Java **plus** a bunch of methods, and **minus** the requirement to understand classes
  - All of the methods in the Processing reference page go away
  - Everything you do in Processing you can do in Java, but it’s a lot more complex
- There are over 4000 classes in Java, and most of them have dozens of methods
  - All these are available in both Processing and Java
  - Google for “Java API”
The End