Inheritance
Inheritance

- Classes that we design in Java can be used to model some concept in our program. For example:

  ```java
  Pokemon a = new Pokemon();
  Pokemon b = new Pokemon();
  ```

- Sometimes we need to create new more “specialized” types that are similar to types we have already created. For example:

  ```java
  Pikachu c = new Pikachu();
  Charmander d = new Charmander();
  ```

- In this case we would expect the Pikachu and Charmander types to have access to everything in the Pokemon type, but each one might have specialized functionality that goes above and beyond what the general Pokemon type can do.
Inheritance

- Object Oriented Programming lets us define new classes from existing classes.

- This technique is called “inheritance”

- Inheritance can be used to help you:
  - Make your code more re-usable
  - Avoid redundancy
  - Simplify maintenance of your software
Inheritance

class Pokemon {
    String name;
    char type;
    int hitPoints;

    public void announce();
}

class Pikachu extends Pokemon {
    int power;

    public void shock();
}

Superclass

Subclass
More on Superclasses & Subclasses

- In an inheritance hierarchy, classes are known as “superclasses” and “subclasses”

- For some class B that extends A
  - A is a superclass of B
  - B is a subclass of A
  - A is a subclass of Object
  - B is also a subclass of Object (via A)

- Sometimes also called ‘parent’ and ‘child’ classes
More on Superclasses & Subclasses

- A subclass inherits certain properties (data) and methods from the superclass.

- Moreover, it has all the non-private behaviors and data of the superclass.

- But the subclass can also …
  - Add new instance variables
  - Add new methods
  - *Override* the methods of the superclass
A Quick Word about ‘protected’

- The protected keyword is an access modifier for methods and variables of a class.

- When a method or a variable is marked as protected, it can be accessed from:
  - Within the class itself
  - Within a subclass

- If you wish to leave your class “open to extension” then you may want to mark variables and methods as protected, rather than private.
Implement a general “Animal” class that contains the following…
- String name
- Double weight
- Double height

All Animals should also be able to do the following:
- Say their name
- Report their height and weight
- Set and get their weight and height
Programming Example *con’t*

- Extends this class into two new classes:
  - Bird
    - Should be able to sing
    - Boolean variable: flight
  - Fish
    - Should be able to swim
    - Double value: maxSwimSpeed
Method Overriding
More on Superclasses & Subclasses

- A subclass inherits certain properties (data) and methods from the superclass.

- Moreover, it has all the non-private behaviors and data of the superclass.

- But the subclass can also …
  - Add new instance variables
  - Add new methods
  - Override the methods of the superclass
Overriding

- Suppose we extend a class and its method does not do exactly what we need?

- For example, suppose we extended Bicycle and created MotorizedBicycle

- Conceptually, a ‘pedal’ method for Bicycle would not work properly for a MotorizedBicycle.
Overriding

- This is called method overriding.
- We declare a pedal method in MotorizedBicycled that knows how to function for a bike with a motor.
- When users call the pedal method on our MotorizedBicycled, it calls our special “overridden” implementation.
- We are effectively ‘hiding’ Bicycle’s pedal method with our own.
Overriding

- We’ve seen an example of a method override in the Snape exercise with `toString()`
- Potion had a `toString()` method
- `java.lang.Object` is Potion’s superclass, it has a `toString()` method
- And since the one for Object is useful to exactly nobody ever in the history of programming we implemented our own.
- This is an example of overriding.
Overriding: Main Idea

- Overriding is a technique where we replace a method of a superclass with one in a subclass.
With overriding we are providing a new implementation of an existing method in the superclass.

The @Override directive can be placed before you override a method in a subclass, but it technically isn’t required.
Programming Example

- Write a class called Shape that models a generic shape
  - Stores a location (x & y)

- Add a method named printInfo to the Shape class that prints out all basic info on the shape (X & Y position)

- Write two new classes called Rectangle and Circle. Rectangles store a length & width, and Circles store a radius.

- Override the printInfo method in the Rectangle and Circle classes so that it provides more detailed information about those objects.
The Object class
If you do not specify an ‘extends’ declaration in your code the compiler will add one for you.

By default, all objects extend `java.lang.Object`.

All classes extend Object directly or indirectly.

The `java.lang.Object` class is the ‘root of the class hierarchy’ in Java.
java.lang.Object

class A {}

public class SimpleClass {
    public SimpleClass() {
        A a = new A();
        a.
    }
}
The Object class

- If a class definition does not specifically include an ‘extends’ class then the class will inherit from the “root” class of Java, which is “Object”.
- “Object” is parent class of all classes, and contains basic methods that are applicable to any class in Java.
- Docs for the Object class:

  http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html
The Object class

- Object contains a method called toString which returns a String version of the name of the class along with its memory address.

- You can override the toString() method in a subclass to provide more information about that class to the user, like this:

```java
@Override
public String toString()
{
    // implement
}
```
Programming Example

- Revisit the geometry package

- Implement a `toString()` method on the `Shape` class that returns the area of the object

- Override this behavior in both the `Rectangle` and `Circle` classes
The “super” keyword
“super” & “this”

- The **this** keyword can be used within a class to refer to an instance variable of that class or to call another constructor.

- The **super** keyword can be used within a class to refer to an instance variable of its superclass or a super class constructor.
“super” & “this”

Using the super keyword

```java
public class A extends B {
    public m() {
        super.xPosition;
        super.yPosition;
        super.zPosition;
        super.someMethod();
    }
}
```

Functionally Equivalent

```java
public class A extends B {
    public m() {
        xPosition;
        yPosition;
        zPosition;
        someMethod();
    }
}
```
Constructor chaining

- You can also use the “super” keyword to call the constructor method on a superclass.

- This will invoke the constructor on the superclass and then return control back to the subclass once it has completed.

- **You must invoke the superclass’s constructor prior to doing anything else in the subclass’s constructor method. (Same as ‘this’ for constructors).**
public class A {
    protected int x;
    public A(int x) {
        this.x = x;
    }
}

public class B extends A {
    protected int d;
    public B(int d) {
        super(d);
        this.d = d + 1;
    }
}
Programming Example

- Add a constructor to your Shape class that accepts an x & y position
- Then add constructors to your Rectangle and Circle classes that accept a width & height (for Rectangles) and a radius (for Circles)
- Use the Shape constructor from the Rectangle and Circle classes with the ‘super’ keyword.
Processing
What is Processing?

- A ‘library’ of code that can be used create graphical programs.
- A teaching tool for programming.
- Developed in part here at NYU’s ITP program.
  - [https://processing.org/overview/](https://processing.org/overview/)
  - [https://processing.org/tutorials/gettingstarted/](https://processing.org/tutorials/gettingstarted/)
- The documentation is targeted at the beginner
How to Install Processing?

- You need to set it up in each Eclipse project you want to use it in.
- Instructions are listed on the site.
We are going to be writing all of our graphical code inside of our own subclasses of a class called “PApplet” that is part of the Processing graphics helper library.

“Processing” is a well documented library that was designed for artists and media developers to rapidly prototype graphical applications.

Easy for beginners to use.
How to Use Processing

- ‘import’ the PApplet
- Just like importing Scanner.
- Declares your intention to use something from the Processing library.

```java
import processing.core.PApplet;

public class ProcessingExample00 extends PApplet {

    public void draw() {
        rect(random(0, 500), random(0, 500), 10, 10);
    }
}
```
How to Use Processing

- ‘extend’ the PApplet
- Thus inheriting all of PApplet’s methods (including a main method)

```java
import processing.core.PApplet;

public class ProcessingExample00 extends PApplet {

    public void draw() {
        rect(random(0, 500), random(0, 500), 10, 10);
    }
}
```
How to Use Processing

- Then we ‘override’ its `draw()` method.

- We are going to ‘draw’ to the ‘canvas’ whatever we would like.

```java
import processing.core.PApplet;

public class ProcessingExample00 extends PApplet {

    public void draw() {
        rect(random(0, 500), random(0, 500), 10, 10);
    }
}
```
How to Use Processing

- The in our draw() implementation we use other methods of PApplet to write things to the canvas.

- Note that draw gets invoked in a loop by the PApplet class.

```java
import processing.core.PApplet;

public class ProcessingExample00 extends PApplet {
    public void draw() {
        rect(random(0, 500), random(0, 500), 10, 10);
    }
}
```
The class you are writing will “extend” PApplet. This means that you are building additional functionality into an already written class, just like how we extended the Shape class to create the Rectangle class.

You have access to all of the instance variables and methods that have already been written for the PApplet class.

PApplet provides a “main” method which instantiates your class when you choose to run your program. You do not need to implement a main method.

```java
import processing.core.PApplet;

public class HelloGraphics extends PApplet {
    // runs one time
    public void setup()
    {
        size(500,500);
    }

    // infinite loop - runs 60 times per second
    public void draw()
    {
    }
}
```
You are programming inside an instance of the PApplet Class.

**Main Program (hidden)**

```java
public static void main()
{
    PApplet program = new PApplet();
    program.start();
}
```

**Instance of PApplet**

```java
public void setup() {}

public void draw() {}
PApplet and You

- PApplet is a well documented class and is designed to be very easy to use. You can refer to the full documentation for the class by visiting this link: http://bit.ly/1L6l0Jj

- We will cover a subset of this documentation but feel free to experiment and play with methods that we don’t cover in class. It’s highly encouraged – be creative! 😊
Anatomy of a PApplet class

- There are two primary methods that you need to override when building a graphics project via PApplet:
  - **public void setup()**
    - Runs one time at the beginning of your program
    - Think of it like a constructor for your graphics project (but it’s not really …)
    - The setup() method is where you place code that you want to run one time as your program starts up.
  - **public void draw()**
    - An infinite loop that runs based on a desired frame rate. The default is 60 times per second, but you can adjust this as necessary.
    - The draw() method is where you place code that you want to repeat over and over again.
Creating a Graphics Canvas

- You can call the `size()` method to set up a new graphics canvas.

- This method call should go in your `setup()` method.

- The `setup()` method takes two arguments – a `width` and a `height`. These values are measured in pixels.

- You can access the `width` and `height` of your project once it has been set by referring to the following instance variables:

  ```java
  this.width
  this.height
  ```

- See Ex00_SizingYourCanvas.java
Changing the background color

- You can change the background color of your project by calling the background method.

- This method is overloaded to included the following headers:

  ```
  background (float grey)
  background (float red, float green, float blue)
  background (float red, float green, float blue, float alpha)
  ```
Color Values

- Color in the PApplet class can be expressed as a byte.

- Recall that one byte contains 8 bits, which means that there are 256 possible values for one byte.
  - 0 is the lowest possible value for a byte, and 255 is the greatest

- Each color “channel” in the PApplet class can be instructed to display its value based on a single byte. For example, to set a black background you could do this:

  ```java
  // set greyscale channel to 0 (all off)
  background(0);
  ```

- And for a white background you could do this:

  ```java
  // set greyscale channel to 255 (all on)
  background(255);
  ```

- See Ex01_ColorDemonstration.java
Color Values

- Each channel can be instructed using a separate byte. For example, the following will display a red background:

  `background(255, 0, 0)`

- And this will set a blue background:

  `background(0, 0, 255)`

- You can use the digital color meter (Mac) or color picker application (other platforms) to extract colors as their RGB value.

- Here’s an RGB color picker that can help you do this via any web browser:  [http://www.rapidtables.com/web/color/RGB_Color.htm](http://www.rapidtables.com/web/color/RGB_Color.htm)
The draw() method
The draw() method

- The draw() method is where most of the action takes place in your project.

- The draw() method is called based on your frame rate. You can set up the frame rate in the setup() method by calling the frameRate() method – the frame rate is expressed as frames of animation per second:

  ```
  frameRate(30);
  ```

- With that said, you can usually leave the default value in place (you will get between 30 and 60 fps which is fine for most applications)
The draw() method

- Anything you place inside the draw() method will be called repeatedly like it is inside of an infinite loop

- For example, try putting a System.out.println() call in the draw method and see what happens.

- Next, try putting the following code inside of the draw loop:

  ```java
  background(255);
  ```

- Nothing happens! That’s because you’re doing the same thing over and over again. Try randomizing the value you’re passing to background and shield your eyes! 😊
Programming Example

- Make your program fade its background color slowly from black to white.

- Hint: use an instance variable to keep track of the current color of the background and change it over time inside your draw() method

- See `Ex02_BackgroundFading.java`
Programming Example

- Extension: When your background gets to be completely white cause it to slowly fade back to black.

- Extension: Turn this into a complete cycle, allowing your background to bounce back and forth between black and white.

- Extension: Now do this with all 3 colors (i.e. your background should look like an ever-changing rainbow)

- See Ex02_BackgroundFadingAllColors.java
The random() method

- The PApplet class contains a random() instance method that is overloaded:

  ```java
  public float random (float max)
  public float random (float min, float max)
  ```

- Note that Papplet prefers floats, not doubles. It’s recommended that you use the built in random method, its little more convenient than doing what we usually do, eg.

  ```java
  (float) (Math.random() * 255)
  ```
Programming Example

- Create a program that randomizes the color of the background one time (when the program starts)

- Next, create a program that randomizes the background color every frame (avert your eyes!)

- Extend your program to have the background only change every 30\textsuperscript{th} frame (hint: use an instance variable as a counter)

- See Ex03_BackgroundChangeEvery30Frames.java
Working with the Mouse
The PApplet class has an instance variable called `mousePressed` that keeps track of whether the mouse button is pressed or not. It is a boolean type.

You can refer to it as follows:

```
this.mousePressed
```
Programming Example

- Change the background color to a random color every time the mouse is pressed

- See Ex04_ChangeBackgroundOnMousePressed.java
public void mousePressed()

- mousePressed() is an method that you can override in your PApplet subclass

- This method will be called every time the mouse is pressed – but it will only do so once (as opposed to checking over and over in the draw() method as in the previous example)

- See Ex04_ChangeBackgroundOnMousePressedUsingMethod.java
The PApplet class has two instance variables called `mouseX` and `mouseY`.

The variables store the current position of the mouse as an integer.

These variables are updated at the beginning of each execution of the `draw()` method.

You can refer to these values by doing the following:

```java
this.mouseX;
this.mouseY;
```
Coordinates

- Processing, like most graphical frameworks, sets the origin to the top left side of the screen.
- This means that your coordinates will always be positive.
Programming Example

- Write a program that changes the background color to white when the mouse is on the left side of the screen, and to black when it is on the right side of the screen.

- Next, write a program that gradually changes the background color as you move the mouse from left to right.

- See Ex05_MousePosition.java
Drawing Shapes
Drawing Shapes

- You can draw simple shapes using a variety of built in drawing methods.
- Note that shapes drawn to the screen are additive – things drawn later will be drawn on top of things that are already there.
- *There isn’t a way to erase the screen other than to paint over it with another color.*
- Calling background(0) will wipe the screen back to black by painting over anything that is already visible.
- See Ex06_SimpleShapes.java
Ellipses

- The ellipse() method can be used to draw circular shapes to the screen. Syntax:

\[
\text{ellipse}(x, y, \text{width}, \text{height})
\]

- You can fill an ellipse by calling the fill() method before you call the ellipse() method. Fill works just like background() – it takes up to 4 color values as integers and fills all shapes drawn afterwards with those colors. Example:

```java
// draw a white ellipse at 200, 200
fill(255);
ellipse(200, 200, 10, 10);
```
Ellipses

- You can tell Java to smooth out your shapes by calling the smooth() method. smooth() takes no arguments. Call it in setup() and forget about it 😊

- You can tell Java not to fill in your shapes by calling noFill(). noFill() takes no arguments and will cause all shapes drawn afterwards to have no fill color.

- The stroke() method will let you choose the outline color for your shapes. This method works just like fill() and background()

- Finally, you can specify the weight of your outlines by using strokeWeight(). This method accepts one argument and will affect all objects drawn from that point on. You can draw a shape without a stroke by calling noStroke()
Programming Example

- Continually draw randomly placed, colored and sized ellipses on the screen.
- Extension: Change the transparency property so that the shapes appear more like a “stained glass” mosaic
- See Ex07_RandomEllipses.java
Programming Example

- Write a program that lets the user paint on the screen. You can simply draw an ellipse to the screen as the user moves their mouse.

- Extension: only paint when the mouse is down

- Extension: paint using random ellipse sizes

- Extension: paint using random colors

- See Ex08_PaintingEllipses.java
Programming Example

- Extend your program so that your ellipses pulsate (i.e. they get bigger up until a point and then start getting smaller, and then repeat)

- See Ex09_PaintingEllipsesWithPulsatingSize.java
Programming Example

- Add in the ability to clear the screen.
- You can do this by checking to see if a key is pressed on the keyboard.
- Use the instance variable “keyPressed” – this is a boolean that indicates the state of the keyboard.
- See Ex09_PaintingEllipsesWithPulsatingSizeEraseScreen.java
Programming Example

- Write a Kaleidoscope program that simultaneously draws an ellipse on both sides of the screen (mirrored across the Y axis)
- See Ex10_Kaleidoscope.java
Rectangles

- You can draw rectangles to the screen using the `rect()` method as follows:

  \[ \text{rect} (x, y, \text{width}, \text{height}) \]

- Rectangles are drawn from their top left corners. You can switch this behavior by calling the following:

  \[ \text{rectMode}(	ext{CENTER}); \]
Programming Example

- Write a painting program that uses random rectangles instead of random ellipses
- See Ex11_PaintingWithRectangles.java
Programming Example

- Write a program that generates a figure that looks like the one on the right
- See Ex12_RectangleShrinking.java
Instance Methods
Instance Methods

- You can implement methods on the PApplet class just like you would on a class that you create on your own.
- For example, you could put this a hello, world method like the one below the draw() method:

```java
public void sayHi()
{
    System.out.println("hi!");
}
```
Write an instance method called `drawRandomShape` that has the following header:

```java
public void drawRandomShape(int x, int y)
```

This method should draw a random shape at the specified x and y location. A random shape includes a random color, random size and random layout (either ellipse or rectangle).

See `Ex13_RandomShapeInstanceMethod.java`
Programming Exercise

- Hide the user’s mouse cursor using the noCursor() method
- Write an instance method called “drawCursor” that draws an ellipse in place of the mouse cursor
- Extension: pulsate the size of the cursor over a few frames.
- Extension: draw the cursor using a red color if the mouse is down, white if it is not
- See Ex14_MouseCursorSizeChange.java
Dynamics
Programming Example

- Write a program that creates a single ball in the center of the screen.
- Have the ball move to the right. If it hits the edge of the screen, stop.
- Next, instead of stopping, have the ball bounce and move to the left.
- Repeat the process for the left side of the screen.
- Extension: Update the ball so that it moves in both directions.
- Extension: The ball should bounce when it hits any edge.
- See Ex15_DynamicBall.java
Object Composition with PApplet
Object Composition

- Remember that Java classes can contain any number of instance variables
- These can be primitive types (int, double, boolean, etc) as well as reference types (Pokemon, FishTank, etc)
Programming Challenge

- Encapsulate the logic to draw and move the ball into a new class called Ball
- Extension: create an array of Ball objects and have them all move at the same time
- Extension: give each ball the ability to store its own color values (red, green and blue)
- See Ex16_DynamicBallWithClasses.java
Homework
Flappy Bird

- You’re assignment is to build a simple game.
- You can accomplish this with the tools you’ve seen today.
- There is an example of a simple game in the source posted to the site, in the flappbird folder.
- See flappybird/FlappyBirdGame.java