Derived Classes and Inheritance

Derived Classes

When designing your programs, some objects may be mostly the same but have some variations. You may want to put every variable and method you may ever need all into one large class. (don't do this)

```
public class Shape {
    String name;
    int length;
    int width;
    int height;
    int radius;
    int diameter;
    int sides;

    // ...
}
```

Derived Classes

Instead, think about what many of your objects share and then you can derive a new class from your base class.

```
public class Shape {
    String name;
    int sides;

    // ...
}
```

```
public class Square extends Shape {
    int length;

    // ...
}

public class Rectangle extends Shape {
    int length;
    int width;

    // ...
}
```

Derived Classes

Subclasses can access public variables and methods in the superclass.

```
public class Shape {
    public String name;

    public void printName() {
        System.out.println(name);
    }
}
```

```
public class Square extends Shape
{

    public Square() {
        name = "square";
    }
}
```

```
Square sq = new Square();
sq.printName();
```

```
Output:
square
```

Derived Classes

You can declare a variable as protected to make it only available within a class and its subclasses.

```
// Will not compile!

public class Shape {
    private String name;
}

public class Square extends Shape {

    public Square() {
        name = "square";
    }
}
```

```
// Works!

public class Shape {
    protected String name;
}

public class Square extends Shape {

    public Square() {
        name = "square";
    }
}
```

Overriding

Overriding

You can override a method in the base class by using the @Override annotation.

```
public class Shape {
    public void draw() {
    }
}
```

```
public class Square extends Shape {

    @Override
    public void draw() {
        // Draws a square
    }
}

public class Circle extends Shape {

    @Override
    public void draw() {
        // Draws a Circle
    }
}
```

Overriding

The power in this is being able to have an array of the base class and then have the related method be called.

```
Shape shapes[] = new Shape[4];

shapes[0] = new Circle();
shapes[1] = new Square();
shapes[2] = new Rectangle();
shapes[3] = new Rectangle();



for (int i = 0; i < shapes.length; i++) {
    shapes[i].draw();    // Check this out!
}
```

Let’s Code

Don’t Forget!

Check the syllabus / schedule for reading assignments and due dates!