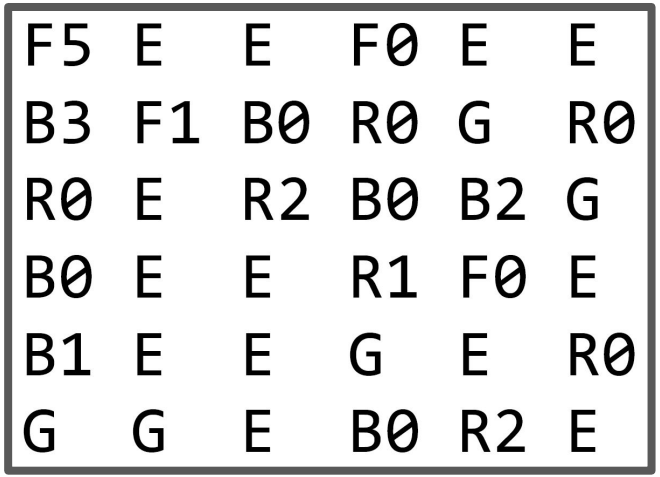
Week 2

Object Orientation basics, Equals(), Clone()

Project 1 is out!

Wildlife of the Plain

- Due Saturday, February 9th
- Focuses on object orientation principles
- Must write test cases

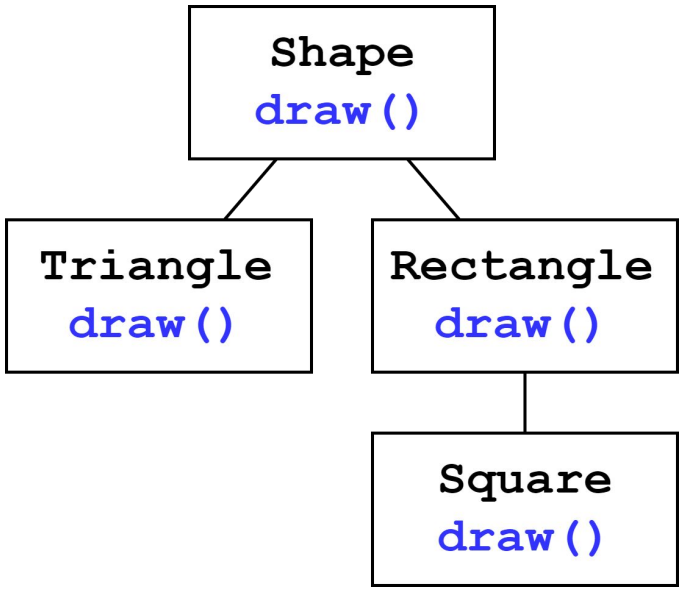


Demo on how to package your zip file submission

Object Orientation in Java

What is Polymorphism?

- Ability to have one type represent many different types
- One of three OOP principles



Why use Polymorphism?

To turn this monster...

```
private ArrayList<Triangle> triangles;
private ArrayList<Rectangle> rectangles;
private ArrayList<Square> squares;

public void drawAll() {
    for(Triangle t : triangles) {
        t.draw();
    }
    for(Rectangle r : rectangles) {
        r.draw();
    }
    for(Square s : squares) {
        s.draw();
    }
}
```

Into this!

```
private ArrayList<Shape> shapes;

public void drawAll() {
    for(Shape s : shapes) {
        s.draw();
    }
}
```

Why use Polymorphism?

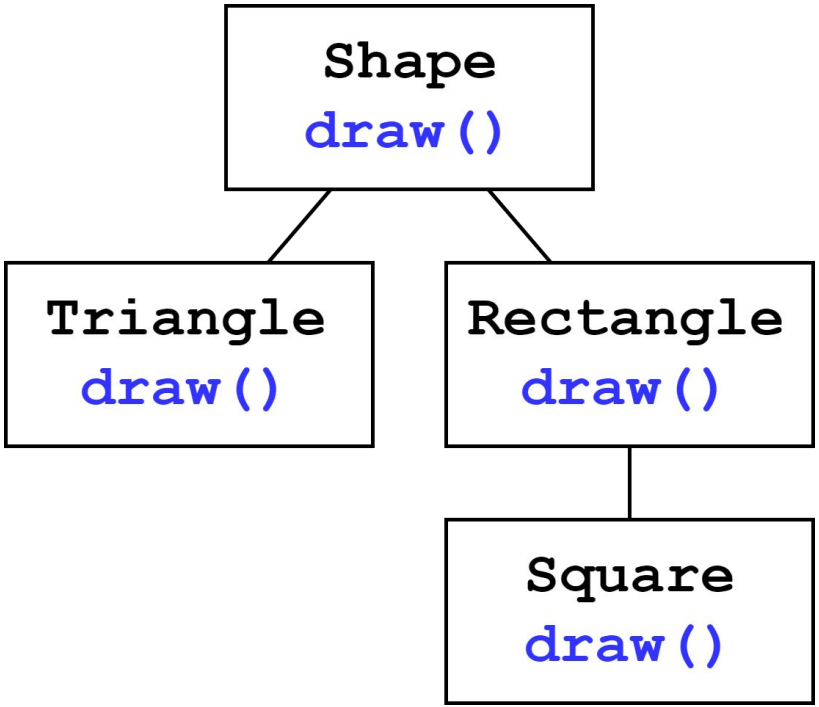
Or this...

```
private ArrayList<Triangle> triangles;
private ArrayList<Rectangle> rectangles;
private ArrayList<Square> squares;

public void addTriangle(Triangle t) {
    triangles.add(t);
}

public void addRectangle(Rectangle r) {
    rectangles.add(r);
}

public void addSquare(Square s) {
    squares.add(s);
}
```

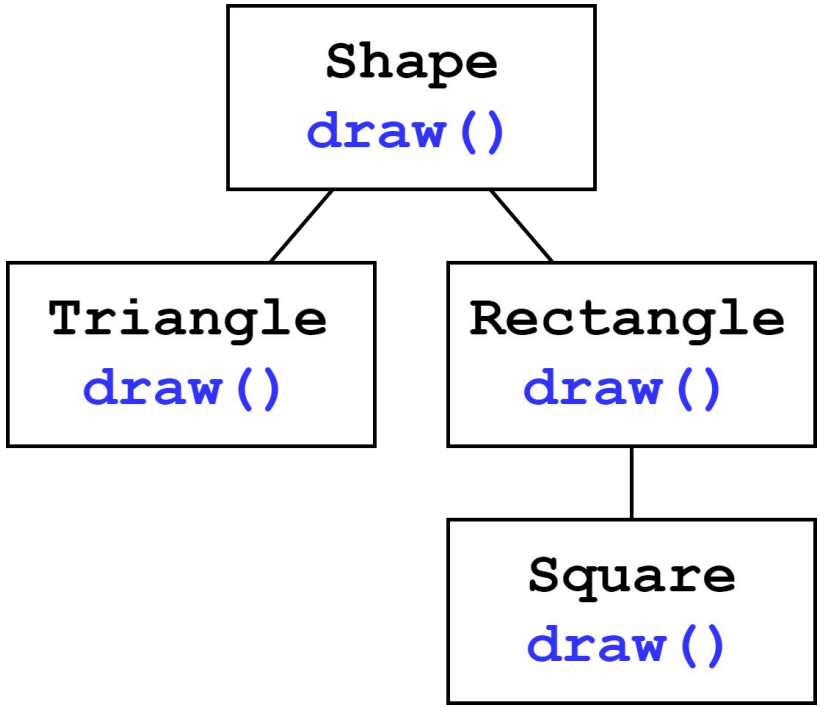
Into this!

```
private ArrayList<Shape> shapes;

public void addShape(Shape s) {
    shapes.add(s);
}
```

How do we create type hierarchies in Java?



How do we create type hierarchies in Java?



By using inheritance!

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

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

```
public class Square extends Rectangle {
    //Already implemented in Rectangle!
}
```

Class vs. Abstract Class vs. Interface

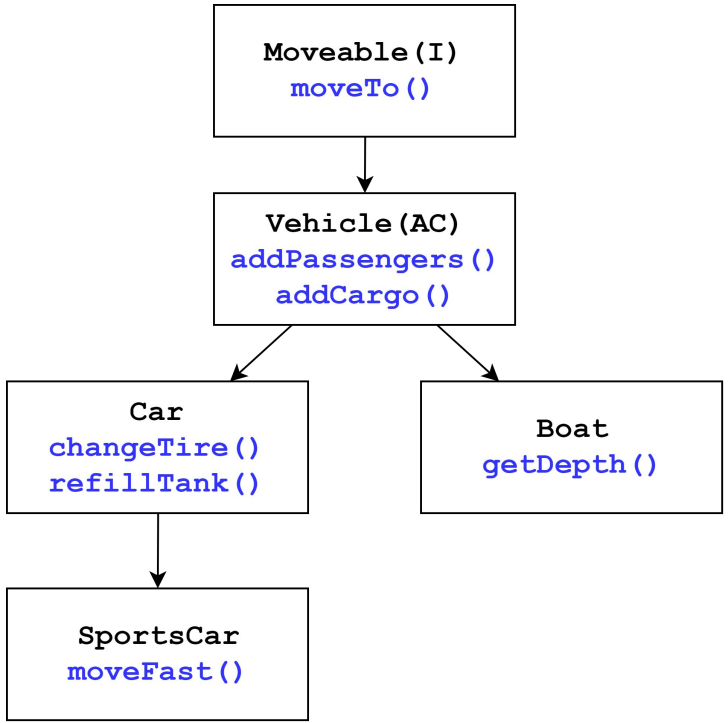
What are the differences between the three?

Class vs. Abstract Class vs. Interface

Class: The specific implementation of a class

Abstract Class: The core idea of all subclasses, defines and implements some methods

Interface: Some ability of an object, defines but doesn't implement any methods



Static type vs. Dynamic type

Vehicle methods:

- moveTo(Location l)

Car methods:

- moveTo(Location l)
- changeTire(Tire t)
- refillTank(int gallons)

```
Vehicle v = new Car();
v.moveTo(iowa);
v.refillTank(10);
```

Casting

Upcasting (Always safe)

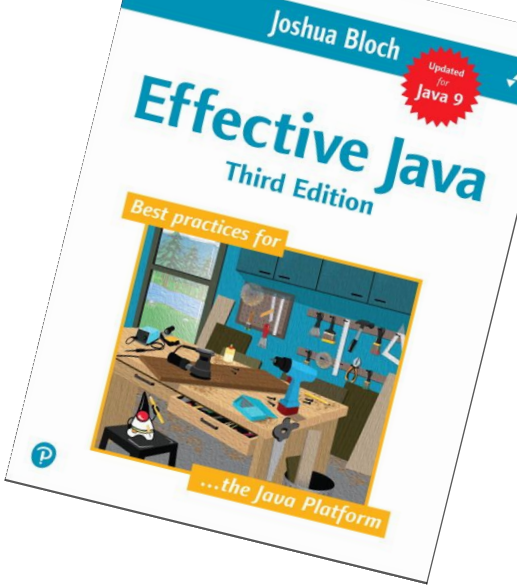
```
Car c = new Car();
Vehicle v = c;
```

Downcasting (Can be unsafe)

```
//Safe, dynamic type is a car
Vehicle v = new Car();
Car c = (Car) v;
```

```
//Unsafe, dynamic type is a car,
//doesn't have SportsCar methods
Vehicle v = new Car();
SportsCar sc = (SportsCar) v;
```

Equals Methods



==

- Can be used for primitives and objects
- Compares primitives values
- Compares object locations

equals()

- Can only be used for objects
- All objects implement it (Default: compare locations)
- **Can be overwritten for a new class to compare fields**

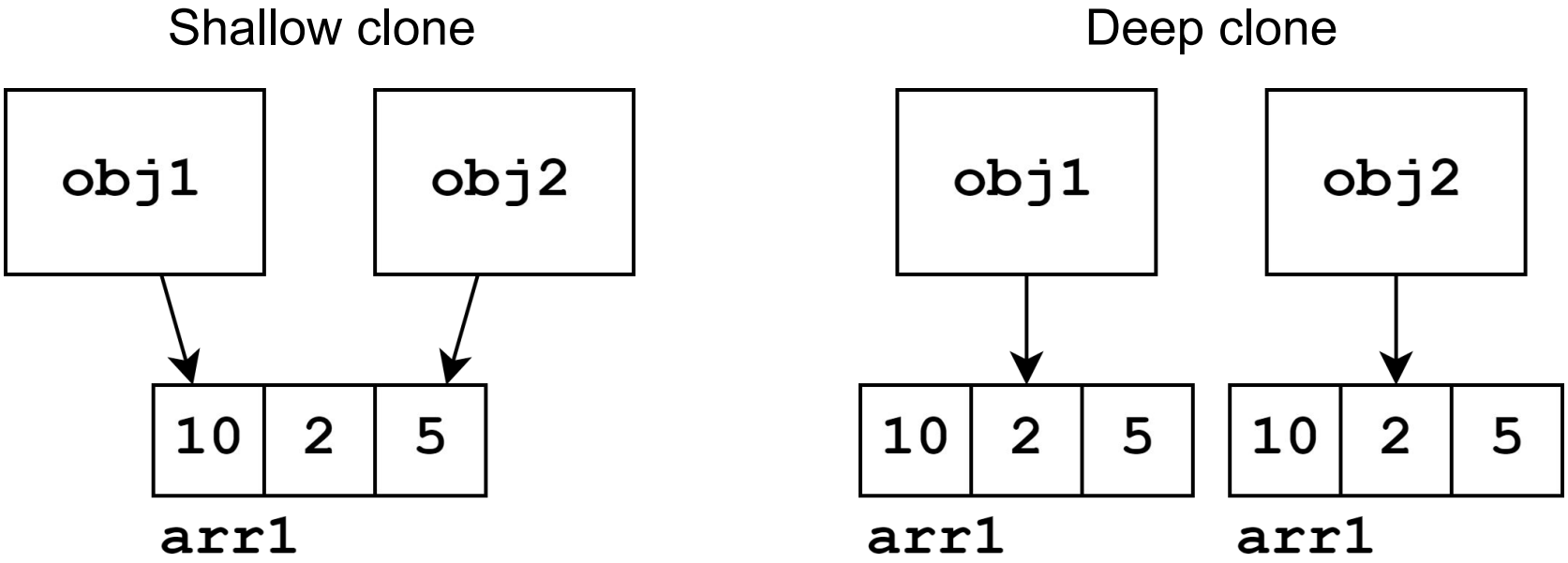
Principles of equals()

- **Reflexive**: x.equals(x) is true
- **Symmetric**: x.equals(y) returns same as y.equals(x)
- **Transitive**: If x.equals(y), and y.equals(z), then x.equals(z)
- **Consistent**: Repeated calls to x.equals(y) yield same output
- x.equals(null) always returns false

This won’t be on exams, don’t worry

Clone Method

Shallow vs. Deep Clone



Equals() and Clone() Exercise

Bonus exercise!

Create a type hierarchy for the following types, and label each type as a class, abstract class, or interface. There is no “correct” answer here!