Polymorphism

- Polymorphism means *many* (poly) *shapes* (morph)
- In Java, *polymorphism* refers to the fact that you can have multiple methods with the same name in the same class
- There are two kinds of polymorphism:
  - Overloading
    - Two or more methods with *different signatures*
  - Overriding
    - A method in a subclass to "override" a method in the superclass that has the *same signature*
- We’ve already seen Overloading scenario with Constructors
  E.g. public PersonDB() {}
  public PersonDB(Person []){}
Cat's `eat()` method overrides Animal's `eat()` method.

### Abstract Classes

- An abstract class is a class
  - With keyword `abstract` placed before declaring a class
    - E.g. public `abstract` class Animal { .. }
  - It cannot be instantiated
    - Illegal: Animal `a` = new Animal()
  - It may have "abstract methods" i.e. methods with keyword `abstract`
    - Abstract methods are body-less i.e. no code within them
  - They also can have regular/concrete methods
    - Methods with code in them

### Setting up Inheritance with an Abstract Class

```java
public abstract class Animal{
    private double hunger;
    private boolean isAwake;

    public void eat(){
        hunger = 0;
    }
    public abstract String makeNoise();
}

/* Dog class */
public class Dog extends Animal {
    // The Dog class must have a concrete makeNoise method.
    // Otherwise, it won't compile.
    public String makeNoise(){
        return "woof!";
    }
}
```
Creating an Interface

public interface Trainable{
    public void sit();
    public void stand();
}

- An interface is like a class except:
  - The keyword interface is used instead of class
  - All of its methods are body-less
  - It has no instance variables
  - It can not be instantiated (Illegal: new Trainable())
- An interface is like a contract, protocol, role, or point of view
- Code written for an interface type works with any object whose class implements it
  - It can assume that all of the subtypes have the methods listed in the interface (e.g. sit and stand)

A Class implements an Interface

public class Dog implements Trainable{
    public void sit(){ // code for sit method }
    public void stand(){ // code for stand method }
}

Example: Trainable Interface

> Lion lion = new Lion();
> Dog dog = new Dog();
> Trainable beast;
> beast = lion; // static type is Trainable; dynamic type is Lion
> beast.sit();
> beast = dog;
> beast.sit(); // static type is Trainable; dynamic type is Dog
Examples of Interfaces

- Interfaces are typically “lightweight”, with only a few method headers

```java
public interface Displayable {
    public void display();
}

public interface XYZProtocol {
    public void connect();
    public void disconnect();
    public Object receive();
}
```

Summary

<table>
<thead>
<tr>
<th></th>
<th>Regular Class</th>
<th>Abstract Class</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td>all concrete</td>
<td>concrete and/or abstract</td>
<td>all abstract</td>
</tr>
<tr>
<td>May have instance variables</td>
<td>yes</td>
<td>yes</td>
<td>no*</td>
</tr>
<tr>
<td>May be instantiated</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

*Except they may have constants*

instanceof Operator with Interfaces

- Usage: `lvalue instanceof T`
- The expression is true if `lvalue` has type `T` or is a subtype of `T`.
- Example:
  ```java
  public class Rectangle implements Displayable { .. }
  public class Square extends Rectangle { .. }
  > Rectangle r = new Rectangle(2,3);
  > r instanceof Rectangle true
  > r instanceof Displayable true
  > r instanceof Square false
  > r = new Square(10);
  > r instanceof Square true
  > Square s = new Square(8);
  > s instanceof Square true
  > s instanceof Rectangle true
  > s instanceof Displayable true
  ```

A Design Problem

- What if we want to write graphics code for all the animals, and want to require that they all have a display(..) method? (There are many ways to accomplish this.)
Syntax for Inheriting a class & Implementing a class

```java
public class SubclassName extends SuperclassName implements Interface1, Interface2 {
    // class body
}
```