Intro to Programming II

Interfaces

Chris Brooks

Department of Computer Science
University of San Francisco
Inheritance review

- Inheritance allows us to reuse existing code.
- Allows us to define a hierarchy of classes.
- *Base class* has the most general behavior.
- *Derived classes* have more specific behavior.
public class Person
{
    public String lastName;
    public String id;
    public void eat() {}
    public void sleep() {}
}
public class Professor extends Person
{
    public String officeNum;
    public void teach() {}
    public void grade() {}
    public void forget() {}
}
Suppose that the Person constructor looks like this:

```java
public Person(String lname) {
    lastName = lname;
}
```

How can we still set the last name in derived class constructors?
We could do something like this:

```java
public Professor(String lname, String office) {
    lastName = lname;
    officeNum = office;
}
```

Except:

- We just had to cut and paste code. (we hate that!)
- If the base class changes, all derived classes need to change.
Instead, let’s just indicate that the parent class’ constructor should be called.

We do this with super()

```java
public Professor(String lname, String office) {
    super(lname);
    officeNum = office;
}
```

Now we don’t need to worry about what the base class’ constructor does anymore.
We can also use super to explicitly call a superclass’ method.

This lets us *extend* a method, rather than overriding it.

For example, let’s say that Person has the following method:

```java
/* in Person.java */
public void greet() {
    System.out.println(``Nice to meet you'');
}
```
6. We’d like for professors to do this greeting, plus a little extra. (They’re long-winded.) So we can do this:

```java
/* in Professor.java */
public void greet() {
    System.out.print("I do say, old chap, ");
    super.greet();
}
```
Write a base class called Animal. Give it two instance variables (name and furColor). Give it a constructor that sets both of these.

Give Animal a printSelf method that prints out the following:
- My name is (name). My fur is (furColor).

Now create a class called Cat that inherits from Animal. Cat should have one instance variable: age.

Write a constructor for Cat that takes three arguments: name, furColor, and age. It should set age itself, then call super with the other two arguments.

Write a method in Cat called printSelf. It should print “I am a cat”, then call the superclass’ printSelf method.
Previously, we talked about abstract classes. They allow a superclass to specify the methods a subclass will respond to without providing an implementation.

Sometimes abstract classes can be awkward to deal with. For example, let’s say we want to create a class called Bat that inherits from Animal.

We also want to say that Bat is a FlyingThing, and that FlyingThings respond to the fly() method.

But we already inherited from Animal!
Interfaces allow us to specify methods that an object is guaranteed to respond to, without specifying an implementation.

A class can implement as many interfaces as it wants.

```java
public interface FlyingThing {
    public void fly();
}

public class Bat extends Animal implements FlyingThing {
    public void fly() {
        System.out.println("I’m flying!’");
    }
}
Interfaces let us specify which methods an object should respond to, without specifying how they should respond.

This provides polymorphism - each object responds to a method in the appropriate way.

A class can implement as many interfaces as it wants.
16-12: Interfaces in the JDK

- Comparable
- Iterable
- Iterator
- Cloneable
-Readable
- and many, many more
Comparable is a particularly useful interface.

compareTo allows you to specify a ‘less than’ relationship for arbitrary objects.
public class Cat extends Animal implements Comparable {
    public int compareTo(Object other) {
        if (!(other instanceof Cat)) {
            System.out.println("Can’t compare these!");
            return 0;
        } else {
            int age2 = ((Cat)other).age;
            if (age < age2)
                return -1;
            else if (age > age2)
                return 1;
            else
                return 0;
        }
    }
}

16-14: Comparable
6 Rewrite Shape to be an Interface that declares an area() method.

6 Rewrite Circle and Rectangle to implement the Shape interface.

6 Now have Rectangle and Circle also implement the Comparable interface.
   △ They will need to implement the compareTo method. Have them compare areas.