Week 2 - Wednesday

COMP 2000

Last time

- What did we talk about last time?
- Objects
- Classes
- Enums
- Packages

Questions?

Project 1

Interfaces

Interface basics

- An **interface** is a set of methods which a class must have
- **Implementing** an interface means making a promise to define each of the listed methods
- It can do what it wants inside the body of each method, but it **must** have them to compile
- A class can implement as many interfaces as it wants

Interface definition

- An interface looks a lot like a class, but all its methods are empty
  - In Java 8 and higher, default implementations can be given, but never mind that now
- Interfaces have no members except for (**`static final`**) constants

```
public interface Guitarist {
    void strumChord(Chord chord);
    void playMelody(Melody notes);
}
```

Interface use

```
public class RockGuitarist extends RockMusician
     implements Guitarist {

     public void strumChord( Chord chord ) {
          System.out.print("Totally wails on that " +
          chord.getName() + " chord!");
     }

     public void playMelody( Melody notes ) {
          System.out.print("Burns through the notes " +
          notes.toString() + " like Jimmy Page!" );
     }
}
```

Usefulness

- A class has an **is-a** relationship with interfaces it implements, just like a superclass it extends
- Code that specifies a particular interface can use any class that implements it

```
public static void perform(Guitarist guitarist,
    Chord chord, Melody notes) {
    System.out.println("Give it up " +
    "for the next guitarist!");
    guitarist.strumChord( chord );
    guitarist.playMelody( notes );
}
```

Interface example

- Let's look at the **`List<E>`** interface
- Some of its methods:
  - **`boolean add(E element)`**
  - **`void add(int index, E element)`**
  - **`void clear()`**
  - **`E get(int index)`**
  - **`int size()`**
  - **`boolean remove(Object o)`**

List implementations

- There are lots of different ways of keeping a list of data
- The **`List`** interface doesn't care how we do it
- And there are lots of implementations that Java provides:
  - **`ArrayList`**
  - **`LinkedList`**
  - **`Stack`**
  - **`Vector`**
- You can use whichever you think best suits your task in terms of efficiency

Implementing Interfaces

Interfaces

- Many interfaces only have a single method
- Consider the following example:

```
public interface NoiseMaker {
    String makeNoise();
}
```

- To implement this interface, a class must:
  - State that it implements the interface
  - Have a public, non-static method called **`makeNoise()`** that takes no parameters and returns a **`String`**

Example classes

- Here are classes that implement **`NoiseMaker`**:

```
public class Pig implements NoiseMaker {
	public String makeNoise() {
		return "Grunt!";
	}
}
public class Explosion implements NoiseMaker {
	public String makeNoise() {
		return "BOOM!";
	}
}
public class Wind implements NoiseMaker {
	public String makeNoise() {
		return "Woosh!";
	}
}
```

Using such classes

- If an object implements an interface, it can be used wherever that interface is needed because we know it can do the job

```
NoiseMaker[] noiseMakers = new NoiseMaker[100];
noiseMakers[0] = new Wind();
```

- We could pass a **Pig**, **Explosion**, or **Wind** object to the **scareChildren()** method

```
public static void scareChildren(NoiseMaker maker) {
    System.out.println("Hey, kids!");
    System.out.println(maker.makeNoise());
}
```

Capabilities

- When a child class extends a parent class, it gains all of its members and methods
- When a class implements an interface, it's promising to have all of the capabilities required by that interface
- In Java, it's impossible to extend more than one parent class
- However, a single class can implement an unlimited number of interfaces
  - It has the capabilities required by each of the interfaces
  - Many unrelated classes could implement the same interface

Implementing more than one interface

- In addition to the **`NoiseMaker`** interface, consider the following interfaces:

```
public interface Colored {
    Color getColor();
}

public interface Operation {
    int process(int a, int b);
}
```

Implementing more than one interface

- Here is a class that implements all three interfaces:

```
public class LoudPinkAdder
     implements NoiseMaker, Colored, Operation {
     public String makeNoise() {
          return "Bang!";
     }
     public Color getColor() {
          return Color.PINK;
     }
     public int process(int a, int b) {
          return a + b;
     }
}
```

Which Interfaces Have You Got?

Testing for an interface

- Given an arbitrary object, it's possible to tell if it implements a specific interface
- As with extending classes, you can use the **`instanceof`** keyword to see if an object is an instance of an interface
  - What an ugly keyword!
- After you discover that an object implements an interface, you can cast it to that interface and call methods defined by that interface
- Don't cast something to an interface it doesn't have or you'll get a **`ClassCastException`** and crash your program!

Testing for an interface example

```
public void tryToMakeNoise(Object object) {
    // Check for interface
    if(object instanceof NoiseMaker) {
        // Perform cast
        NoiseMaker maker = (NoiseMaker)object;
        System.out.println(maker.makeNoise());
    }
    else
        System.out.println("Can't make noise.");
}
```

Quiz

Upcoming

Next time…

- Lab 2 tomorrow
- On Friday, we'll talk about defining and extending interfaces

Reminders

- Keep reading Chapter 10
- Get started on Project 1