# Interfaces

CSE 142, Summer 2003

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/03su/



## Readings and References

- References
  - » The Java Tutorial on Interfaces
    **http://java.sun.com/docs/books/tutorial/java/interpack/interfaces.html**



## Classes Reviewed

- The basic unit of programming in Java is a *class definition*
  - » The *class* specifies properties and responsibilities
  - » Individual *objects* are created as needed
  - » All objects of the same class have the same list of properties and responsibilities
  - » Properties can contain simple values or be references to other objects
- Every object is an *instance* of some class
- Each class defines a new *type*



## Recall the specification of an Acrobat

You are an Acrobat

When you are asked to **Clap**, you will be given a number. Clap your hands that many times.

When you are asked to **Twirl**, you will be given a number. Turn completely around that many times.

When you are asked to **Count**, announce how many actions you have performed. This is the sum of the numbers you have been given to date.

## Can Dogs be Acrobats too?

- The essence of an Acrobat being is that it can Clap, Twirl, and Count
- *Any* class of beings that can Clap, Twirl, and Count could satisfy our needs
- We might want to add these capabilities to a class like Dog (probably not Cat, I would say)
  - » It's still a Dog, but we want it be a performer too



## Responsibilities

- The problem is that we have two lists of responsibilities
  - » Dog : eat, getMealSize, getCurrentWeight
  - » Acrobat : clap, twirl, getEventCount
- We can add Acrobat methods to the Dog class
  - » Does that make a Dog into an Acrobat in the eyes of the Ringmaster?



## "I can do the Acrobat thing"

- What we need is something that is true of any class that can do the right things
- Every Acrobat can clap, twirl, and getEventCount
- So if a class could promise:
  - » *I can clap, twirl, and keep track of how many times I've done something*
- then we would be able to tell other classes
  - » *this class can do the Acrobat thing for you*



## public interface

- The Java *interface* is a very nice way to tell other classes exactly what your class can do
  - » "these are the responsibilities I have implemented"
- The class is saying "I can do Acrobat functions" as opposed to saying "I am an Acrobat"
- Any class that implements the Acrobat interface guarantees that it has methods for all the things that any Acrobat must do

# public interface Acrobat

```
/**
 * This interface defines the methods that a class must implement
 * in order to be considered an Acrobat.
 */
public interface Acrobat {
  /**
   * Twirl around as instructed.
   * @param k the number of times to twirl
   */
  public void twirl(int k);
  /**
   * Clap as instructed.
   * @param k the number of times to clap
   */
  public void clap(int k);
  /**
   * Tell the caller how many things we've done so far.
   * @return the number of claps and twirls to date
   */
  public int getActionCount();
}
```



# using an interface in a class definition

- Each of the classes that wants to be considered for an Acrobat role must say so at the very beginning of the class definition

```
public class Student implements Acrobat {...}
public class Dog implements Acrobat {...}
```

- You are telling the compiler that this class guarantees that it will implement all the methods that are required in the interface



# What is the guarantee?

```
/**
 * This interface defines the methods that a class must implement
 * in order to be considered an Acrobat.
 */
public interface Acrobat {
  /**
   * Twirl around as instructed.
   * @param k the number of times to twirl
   */
  public void twirl(int k);
  /**
   * Clap as instructed.
   * @param k the number of times to clap
   */
  public void clap(int k);
  /**
   * Tell the caller how many things we've done so far.
   * @return the number of claps and twirls to date
   */
  public int getActionCount();
}
```

Each of these methods is available in the implementing class.



# Anybody can be an Acrobat

- Note that we no longer have an Acrobat *class*
- Instead we have an Acrobat *interface*, that <u>any</u> class can implement
  - » **`public class Student implements Acrobat`**
    - re-titled version of the old Acrobat
    - same methods as before
  - » **`public class Dog implements Acrobat`**
    - updated version of the old Dog
    - same methods as before, plus the Acrobat methods

# Using Acrobat interface in Ringmaster

- Now we can use objects of several different classes to do Acrobatic things, not just one class
- So the Ringmaster can create a whole crowd of objects of different types, and ask each of them to do Acrobat things no matter who they are
  - » Each class to be used this way must say that it

```
implements Acrobat
```



# We're all Acrobats on this bus

```
Acrobat actA = new Student("Alpha","Broti");
Acrobat actB = new Student("Cheri","Delay");
Acrobat actC = new Dog("Jessie");

actA.clap(3);
actB.twirl(4);
actC.twirl(7);
actA.clap(8);
```

from Ringmaster.java



# But I'm more than just an Acrobat ...

- Sometimes we want to access some of the special skills of the object, because we know that there are hidden talents
- Java lets us *cast* the object to another type

```
double weight = ((Dog)actC).getCurrentWeight();
```

- The programmer (you) is telling the compiler
  - » *trust me, it really is a Dog and it can do more than just the Acrobat responsibilities*



# Cast to Dog

- Tell the compiler that this reference is to a Dog
  - » a Dog can do much more than just be an Acrobat

good idea, actC actually is a Dog

```
double weight = ((Dog)actC).getCurrentWeight();
```

- If you tell the compiler that the object is a Dog, but then it actually turns out to be a Student, the program will stop with an error

bad idea, actA is not a Dog

```
double weight = ((Dog)actA).getCurrentWeight();
```