CSE 142

Interfaces

Outline for Today

- Review: specification vs implementation
- Java interfaces – specifying behavior common to several classes
- Implementing interfaces in classes
- Interface types and class types
- Interface types and collections

Specification vs Implementation – Review

- Two different perspectives
  - Client – what is publicly available to users of a class
  - Implementer – public interface + private implementation details
- Function headings and comments (JavaDoc) give us a way to record what is available to the client – they *specify* the class
  - Often informally thought of as the class's *interface*
  - However, the class combines both specification and implementation
- There are many cases where we would like to be able to give a pure specification – no implementation details at all

Java Interfaces

- A new Java construct
- Looks much like a class definition

```
/** description of this interface */
public interface name {
    /** JavaDoc comments */
    specifications (only) of methods and constants that belong to the interface
    // regular comments
    /* are also allowed */
}
```

- Pure specification – no implementation

Recall: Performer Role-Playing

- We had Performer objects that knew how to:
  - Clap
  - Twirl
  - TellCount
- We had different types of Performer objects:
  - Acrobat, Choreographer, AcrobatWithBuddy, Actor, Curmudgeon
- Let's implement a simulation in Java

Performer Interface

- File Performer.java (comments abbreviated)

```
/** Interface to Performer objects. ... */
public interface Performer {
  /** Clap nTimes ...   */
  public void clap(int nTimes);
  /** Twirl nTimes... */
  public void twirl(int nTimes);
  /** Report how much this performer has clapped and twirled ... */
  public int tellCount();
}
```

Notes

- Bodies of methods { ... } replaced by ;
- Besides method headings, interfaces can contain constants (later), but essentially nothing else
- An interface declares a type (here Performer) just like a class definition
  - Can have variables and parameters with the type (more below)

    Performer bozo;
- But an interface does not contain *any* implementation
  - Corollary: cannot create an instance of an interface (can't use new) (Why?)

    Performer clarabelle = new Performer();   // can't do this

Implementing Interfaces

- Any class can implement an interface by naming it in an *implements* clause

  public class Acrobat implements Performer { ... }
- Meaning
  - The class *must* provide implementations of *all* of the methods declared in the interface
  - The class can contain any additional methods or instance variables desired
  - Instances of the class can be used as if they had either the class type or the interface type

    [An instance of Acrobat has type Acrobat and also has type Performer]

Examples

```
/** Acrobat - an implementation of Performer.*/
public class Acrobat implements Performer {

  /**  Twirl the specified number of times */
  public void twirl(int n) {  … }

  /** Clap the specified number of times */
  public void clap(int n) {  …  }

  /** Report the total number of claps and twirls*/
  public int tellCount() { … }

  [Other methods and instance variables as
needed]
}
```

```
/** Crumudgeon- an implementation of Performer.*/
public class Crumudgeon implements Performer {

  /**  Twirl the specified number of times */
  public void twirl(int n) {  … }

  /** Clap the specified number of times */
  public void clap(int n) {  …  }

  /** Report the total number of claps and twirls*/
  public int tellCount() { … }

  [Other methods and instance variables as needed]
}
```

What Does This Buy Us?

- **Answer – can now write code that works with any sort of Performer, regardless of the actual kind(!)**

```
/** Make a performer twirl and then report its count
 * @param p  a Performer object
 * @param  n   number of times to twirl
 * @return the performer's current count */
public int twirlAndCount(Performer p, int n) {
    p.twirl(n);
    return p.tellCount();
}
```

  - **When this method is called, the first argument can be an instance of *any* class that implements Performer**

    Because the types match: instances of a class that implements Performer have type Performer, in addition to their class type

Type Compatibility

- **If a parameter or instance variable has a type T, then it can refer to any object that has type T**
  - **If T is a class type, any instance of T**
  - **If T is an interface type, any object whose class implements T**
  - **If T is Object, it can refer to any object**
- **Legal examples**

```
Acrobat one = new Acrobat();
Performer p = one ;      // one and p refer to the same object
```

- **Not legal**

```
Acrobat two = p;         // error – p might refer to an Acrobat, but it might
                         // refer to a different kind of Performer, not an Acrobat
                         // [Can use a cast if it really is an Acrobat]
```

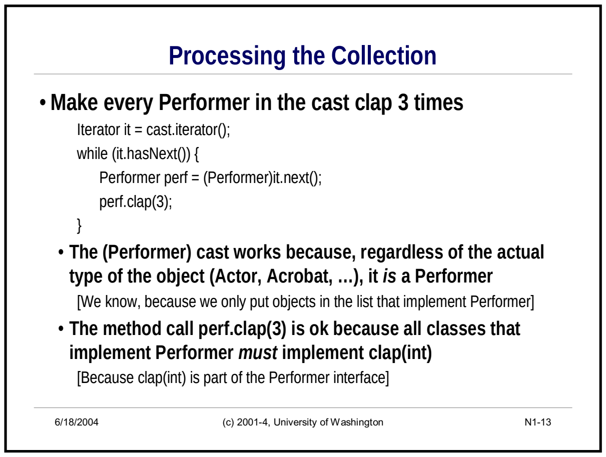
What Else Does This Buy Us?

- **Collections!**
- **Suppose we have a collection**

```
ArrayList cast = new ArrayList();
```

and we add a bunch of Acrobats, Choreographers, Actors, and Curmudgeons to this collection

```
Acrobat tarzan = new Acrobat();
Actor jane = new Actor();
Actor chetah = new Actor();
cast.add(tarzan);
cast.add(chetah);
cast.add(jane);
```



Processing the Collection

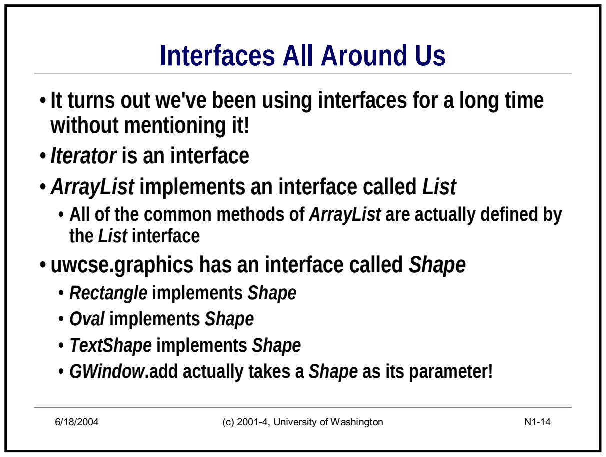
- **Make every Performer in the cast clap 3 times**

```
Iterator it = cast.iterator();
while (it.hasNext()) {
    Performer perf = (Performer)it.next();
    perf.clap(3);
}
```

  - **The (Performer) cast works because, regardless of the actual type of the object (Actor, Acrobat, ...), it *is* a Performer**

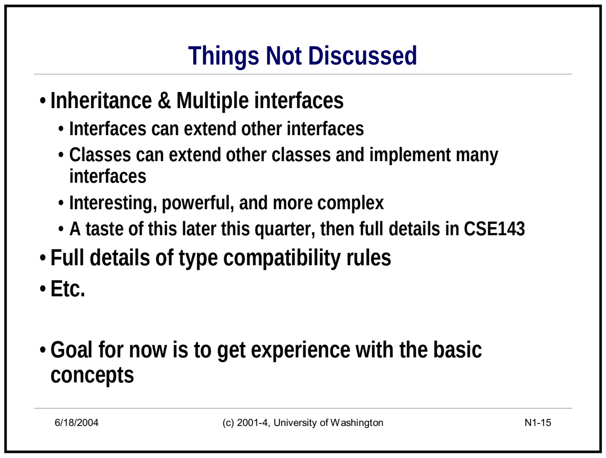
    [We know, because we only put objects in the list that implement Performer]
  - **The method call perf.clap(3) is ok because all classes that implement Performer *must* implement clap(int)**

    [Because clap(int) is part of the Performer interface]



Interfaces All Around Us

- **It turns out we've been using interfaces for a long time without mentioning it!**
- ***Iterator* is an interface**
- ***ArrayList* implements an interface called *List***
  - **All of the common methods of *ArrayList* are actually defined by the *List* interface**
- **uwcse.graphics has an interface called *Shape***
  - ***Rectangle* implements *Shape***
  - ***Oval* implements *Shape***
  - ***TextShape* implements *Shape***
  - ***GWindow*.add actually takes a *Shape* as its parameter!**



Things Not Discussed

- **Inheritance & Multiple interfaces**
  - **Interfaces can extend other interfaces**
  - **Classes can extend other classes and implement many interfaces**
  - **Interesting, powerful, and more complex**
  - **A taste of this later this quarter, then full details in CSE143**
- **Full details of type compatibility rules**
- **Etc.**

- **Goal for now is to get experience with the basic concepts**