Extending Interfaces

CSE 142, Summer 2002

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/02su/

Readings and References

- Reading

- Other References
  - » Chapter 6, Interfaces, in *Core Java volume 1*, by Horstmann and Cornell

Using an interface

- By specifying an interface, you can guarantee to any caller that the implementing class contains a particular set of methods
- The definition of the interface shows exactly what the methods must look like to the public
- All methods in the interface must be implemented in order to make this guarantee
  - » an animal that can eat and make noise, but cannot sleep, is not a real Animal

Recall the Actor interface

- We are using the Actor interface in our code

```
public class OvalSlider implements Actor {...
```

- OvalSlider guarantees that it has implemented the methods in the Actor interface

`void addTo(uwcse.graphics.GWindow w)`
Every Actor must be able to draw itself on a GWindow.

`void doAction(Stage stage)`
Every Actor must implement some fundamental action.

`void removeFromWindow()`
Every Actor must be able to remove itself from its GWindow.

Definition of the Actor interface

- The Actor interface is defined in the UWCSE library
- It has been defined the same way for over a year and much code has been written using it
- What if we want to add some capabilities that we now expect Actors to be able to do?
  - » If we change Actor we will break existing code because the new methods that we want to define don't exist in the old code

keyword **`extends`**

- We can define a new interface that extends another interface
- This means just what you would think it means
  - » interface Voter extends Citizen

    I know how to do the things that every Citizen can do, plus I know how to do the things that a Voter can do
  - » interface List extends Collection

    I know how to do everything that a Collection can do, plus I know how to do the special things that a List can do

interface ClickableActor extends Actor

```
package skel;

import uwcse.graphics.*;
import uwcse.animation.*;

/**
 * Extend the basic interface for an actor in an animation
 * so that it can help recognize that it has been clicked.
 */
public interface ClickableActor extends Actor {
    /**
     * Decide if the given Shape intersects any of the Shapes that
     * this object is displaying.
     * @param other the other Shape that we might intersect with
     */
    public boolean intersects(Shape other);
    /**
     * This method is called whenever this Actor was the one clicked on.
     * @param stage the Stage that we are being displayed on
     */
    public void doClickAction(Stage stage);
}
```

added methods in ClickableActor

- By extending the Actor interface, the ClickableActor interface adds two new methods

```
public boolean intersects(Shape other);
public void doClickAction(Stage stage);
```

- Any class that implements ClickableActor is saying:
  - » it implements all the methods of Actor
  - » *and* it implements “intersects” and “doClickAction”

ShapeSlider is a ClickableActor

- We implemented the Actor interface in OvalSlider

```
public class OvalSlider implements Actor {...
```

- I've written ShapeSlider to handle more Shapes and to be clickable

```
public class ShapeSlider implements ClickableActor {…

    public boolean intersects(Shape other) {…

    public void doClickAction(Stage stage) {
```

homework 4 with frogCast

intersects(Shape other)

```
/**
 * Decide if the other Shape intersects the Shape that
 * this object is displaying.  This method uses the
 * intersects(other) method to decide if there is an
 * intersection.
 * @param other the other Shape that we might intersect with
 */
 public boolean intersects(Shape other) {
   return theShape.intersects(other);
 }
```

doClickAction(Stage stage)

```
/**
  * This method is called whenever this Actor is
  * clicked on.  The method does whatever is needed to
  * implement the clicked-on behavior for this class of
  * objects.
  * @param stage the Stage that we are being displayed on
  */
 public void doClickAction(Stage stage) {
//  System.out.println("I was clicked, said "+this);
    deltaX = -deltaX;
    deltaY = -deltaY;
    if (clickSound != null) {
        clickSound.play();
    }
 }
```

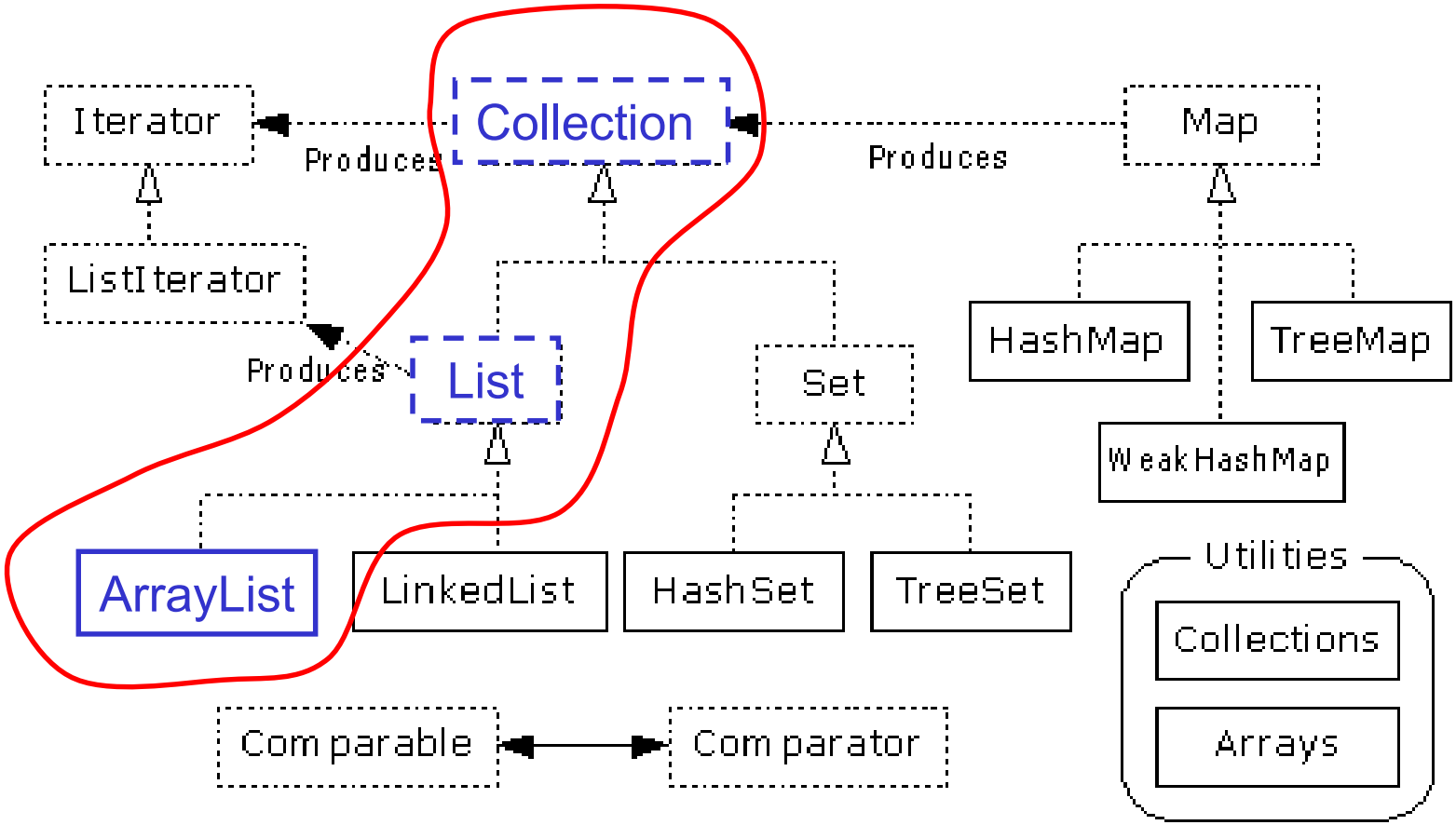
Use the simplest interface you can

- All of the cast members are Actors
  - » Some of the cast members are ClickableActors
- So if a class doesn't care about the snazzy clickable features (intersects, doClickAction) it can just deal with the objects as Actors

```
/*  Add all the actors to the stage */
for (int i=0; i<theCast.size(); i++) {
    Actor stiff = (Actor)theCast.get(i);
    System.out.println("Adding "+stiff);
    theStage.addActor(stiff);
}
```

from skel.AssistantDirector

Collections Framework Diagram



- Interfaces, Implementations, and Algorithms
- From Thinking in Java, page 462

ArrayList implements List

- The ArrayList class is an implementation of the List interface

```
public class ArrayList extends AbstractList
        implements List, RandomAccess, Cloneable, java.io.Serializable
{...
```

- Therefore, all the methods that are expected for Lists are implemented one way or another by ArrayList

ArrayList.java

List interface extends Collection interface

- The Collection interface is the basic set of things you can do with a bunch of objects
- The List interface adds the idea of the objects being in a particular sequential order

- Every class that implements the List interface
  - » has all the methods in the Collection interface
  - » plus some more for the List interface

Collection and List

- Collection
  - » The root interface in the collection hierarchy. A collection represents a group of objects, known as its elements. Some collections allow duplicate elements and others do not. Some are ordered and others unordered.
- List
  - » An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.