

Interfaces & Generics

Lecture 03, Aug 28 2007

Last Time

- Testing is important!
- We discussed JUnit 3
 - JUnit 4 is quite different. RTFM
 - Use `assert(<your boolean test here>);`
 - I WILL be looking for style here. Clean code...
- DeathBox spec ambiguity...
 - “String containing an integer...”
 - Does “6abc” live or die?

The Interface worldview

- Java supports *programming by interfaces*
- Idea: don't hard-code concrete classes into your code
- Instead, refer to everything through interfaces
- Why?

The Interface worldview

Design principle: **Principle of Abstraction**

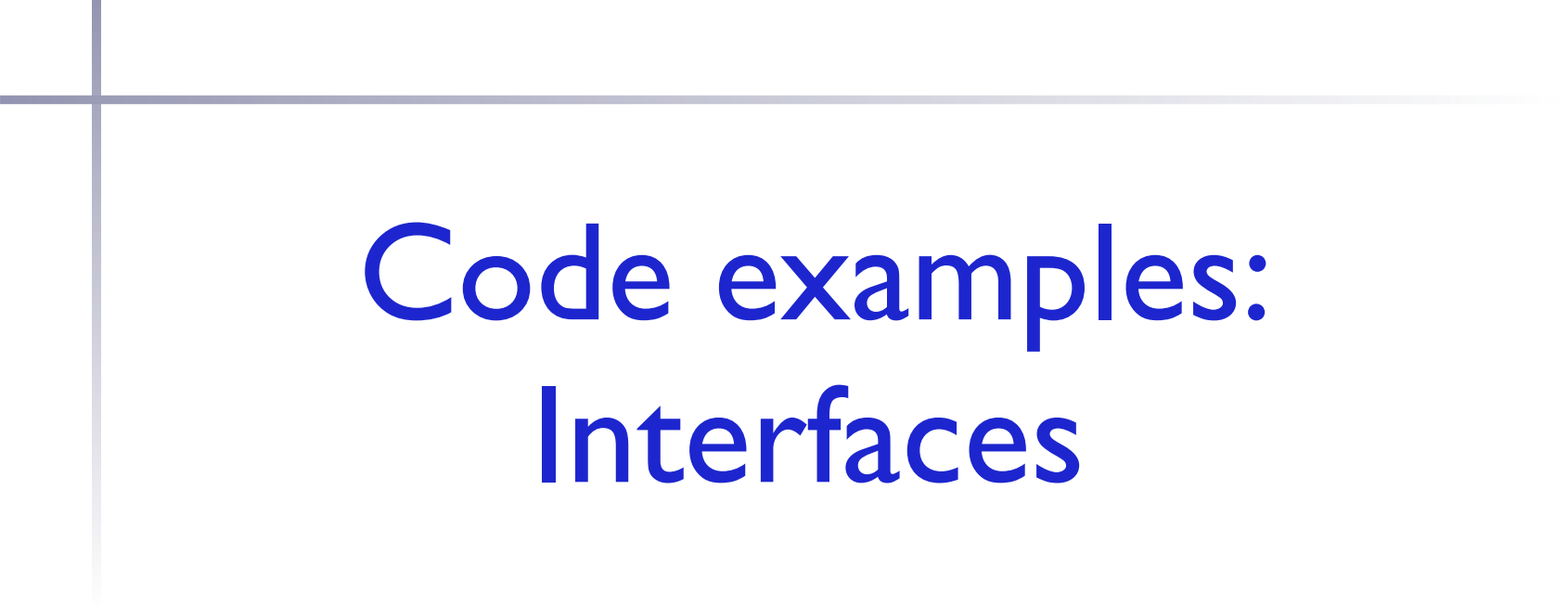
Only work with the relevant aspects of a thing;
ignore stuff that's not immediately relevant.

The Interface worldview

Design principle: **Centralized Definition**

Define a thing only once in a program; use a symbolic name to refer to it later.

A.k.a.: Don't repeat yourself...

A decorative L-shaped line consisting of a vertical line on the left and a horizontal line extending to the right, both in a light gray color.

Code examples: Interfaces

Interface for DeathBox

```
public interface DeathBox {
    /**
     * Attempt to put another object into a DeathBox.
     *
     * @param o the object to attempt to place in the DeathBox
     * @return true if o was put into the DeathBox successfully, and
     *         false if 'you died' attempting to put it in (or were
     *         already dead before the latest putInto attempt).
     */
    public boolean putInto(Object o) ;
}
```

Implementation of ...

- Implement the interface

```
public class DeathBoxKniss implements DeathBox {
    /// constructor
    public DeathBoxKniss()
    {    dead = false;    }

    /// function description here!!!
    public boolean putInto(Object o)
    {
        if( dead ) return false;
        ....
    }

    private boolean dead;
}
```

More on Interfaces

- Interfaces only “declare” functions
 - No function definitions (no body)
 - They do not allow protected/private functions
 - Only legal if the user can call it
 - They do not allow member variables
 - Interfaces themselves have no state
 - Constants ARE allowed tho (public, static, final)

Interfaces as concepts

- Multiple inheritance is illegal in java
- A class CAN implement multiple interfaces
- Abstract generic behaviors, for example:
 - Comparable
 - Additive
 - Iteration

/// Note: use of generic type “T” here

```
public interface Comparable<T> {  
    int compareTo( T obj );  
}
```

Interfaces are extensible

- You can add more functionality

```
public interface MyCompare<T> extends Comparable<T> {  
    boolean isSame(T obj );  
}
```

- You can compose interfaces
 - New interface with properties of multiple
 - Looks like multiple inheritance....

```
public interface Z extends X,Y {}
```

Accessing Constants...

- There can be ambiguity with interfaces

```
public interface A  
{ int val=1; }
```

```
public interface B extends A  
{ int val=2; }
```

```
public class C implements B {...};
```

- Which value to we access?

```
C c = new C();  
c.val == ?
```

Accessing Constants...

- There can be ambiguity with interfaces

```
public interface A  
{ int val=1; }
```

```
public interface B extends A  
{ int val=2; }
```

```
public class C implements B {...};
```

- Which value to we access?

```
C c = new C();  
c.val == 2;  
((A)c).val == 1;
```

Why interfaces?

- Separate appearance and behavior
- Interface == how the object looks to programmer
- Implementation == how the object behaves
- How is this different than inheritance?
- Interfaces allow a clean “view” of a class
- Do you write the test for...
 - DeathBox interface or..
 - DeathBox<yournamehere> class
 - How does JUnit affect this?

Next time....

- Generics
 - Abstract type
 - Further separate appearance and behavior
 - Avoid dangerous runtime type casting
 - Allow you to avoid redundant coding! (YAY)