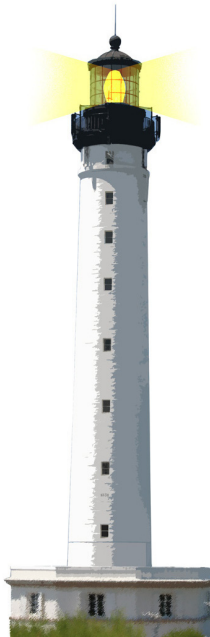
Learning Object-Oriented Programming and Design with TDD

Interfaces Support Evolution

S. Ducasse

http://stephane.ducasse.free.fr



Interfaces

- A good element of Java :)
- Group of method signatures
  - and since Java 80 default methods...
- Used by the type checker
- Support the manipulation of instances of classes not in subtype relation (i.e. not in the same hierachy)

```
interface {
  //methods
}
```

Example

All methods are implicitly public and all fields are public static final

```
interface Polygon {
  public static final String color = "blue";
  public  double getArea();
}
```

```
interface Polygon {
  String color = "blue";
  double getArea();
}
```

class A Implements I

```
class Rectangle implements Polygon {
  ...
  public double getArea() {
    return length * width;
  }
}
```

Classes - Interfaces

Any class implementing an interface MUST defined the methods specified in the interface.

- A class can implement many interfaces
- A class inherits from a single superclass
- An interface can implement from multiple interfaces

```
interface Line {
  //members of Line interface
}

interface Polygon extends Line {
  //members of Polygon interface and Line interface
}
```

Interfaces: step back

A nice mechanism for statically checked languages

- defines what is expected
- lets the system evolve

When you use a class as a type:

- You freeze the possible instances
- You will only be able to have instances of type or subtypes

When you use an interface as a type:

- You will be able to use any instance of classes implementing the interface

Remember

```
Box b = new Box();
```

b can only contain instances of Box and its subclasses

```
class MyBox extend Object {...}
Box b = new MyBox()
>>>> BREAK!!!
```

Because there is no type relationship between MyBox and Box

Now with interfaces

```
interface IBox {
  double getArea()
  double volume()
  }
```

```
class MyBox extend Object implement IBox {...}
IBox b = new MyBox();
>>> Valid
```

```
class Box extend Rectangle implement Ibox {...}
IBox b = new Box();
>>> Valid
```

```
class MyBox2 extend ZKZ implement IBox {...}
IBox b = new MyBox2();
>>> Valid
```

Interfaces support evolution

- You can reuse a program expecting a given interface by passing a new class implementing the give interface
- This is key!
- This cannot be done if you use a class.
- With a class you can only pass a subclass.

An interface can be composed of interface

```
interface IRectangle { double getArea() }
interface IBox extend IRectangle { double getVolume() }
class BBox extend Object implement IBox {}
```

```
IRectangle r = new BBox()
>>> Valid
```

```
IBox r = new BBox()
>>> Valid
```

Interfaces and nominal types

- Pay attention two interfaces with different names but the same contents are NOT compatible
- You will not be able to substitute instances of a class using one interface by instances of another class using another interface with the same contents

A course by Stéphane Ducasse

http://stephane.ducasse.free.fr

Reusing some parts of the Pharo Mooc by

Damien Cassou, Stéphane Ducasse, Luc Fabresse

http://mooc.pharo.org