Interfaces vs. Inheritance
Abstract Classes
Inner Classes
Lecture 33

Readings
This Week: No new readings. Consolidate!
(Reminder: Readings are absolutely vital for learning this stuff!)

Labs and Tutorials
This Week: Lab #10
111 TAs in Learning Centre might mark labs – more details coming… (121 TAs won’t do this marking, so check the posted schedule on the Learning Centre door.)

Final Exam
Wednesday, April 15, 7pm in SRC A
This wasn’t a good room last year, but we’re stuck with the date, time, and room UBC assigns.
Note: You are allowed to do scratch work on the exam paper! (But write your answer where indicated if the problem tells you to.)

Programming Assignment 3
- Assignment 3 is up on WebCT!
  - Click on the "Assignments" icon.
  - Assigned Sunday evening – sorry for delay!
  - Due at NOON, April 6 (Monday), via electronic hand in.
- Tips:
  - There is some Eclipse setup. Set-up ASAP!
  - Work in pairs. Some conceptual stuff.
  - Think carefully before coding. If concepts right, the coding is much much easier.

Learning Goals
By the end of class today you will be able to...
- List similarities and differences between interfaces and inheritance.
- Create abstract classes and extend abstract classes.
- Write code that uses inner classes.
Interfaces vs. Superclasses
We learned these as completely separate concepts:
- An **interface** is a contract, specifying some methods that must be implemented by any class that claims to implement the interface.
- A **superclass** is a class from which other classes can **inherit** methods and instance fields, so we can reuse the superclass’s implementation.

**Interfaces vs. Superclasses**
But they have similarities…
- Both allow declaring references that can point to different kinds of objects, e.g.,
  - `Comparable x;`
  - `x = new Double(3.14);`
  - `x = new UBCStudent(“Alan”);`
  - `Person y;`
  - `y = new UBCStudent(“Alan”,0.0);`
  - `y = new Celebrity(“Paris Hilton”);`

**Interfaces vs. Superclasses**
- Both allow creating different, new classes that share some of the same methods, e.g.
  - `Double` and `UBCStudent` both implement `Comparable`, so they both have `int compareTo()`.
  - `Swimmer` and `Crawler` both inherit from `Animal`, so they have e.g., `changeImage()`

**Interfaces vs. Superclasses**
- They even have similar UML:

**Interfaces vs. Superclasses**
- They have differences, too…
  - `LightInfantry` and `Kangaroo` both implement `Comparable`, so they share the `compareTo()` method.
  - `Animal` and `Crawler` both inherit from `LightInfantry`, so they can override the `changeImage()` method.
  - `Swimmer` and `Crawler` both inherit from `Animal`, so they have e.g., `changeImage()`.

OK!
Interfaces vs. Superclasses
They have differences, too…
- Java does NOT allow multiple inheritance…

```
Macropod
  ...

Light Infantry
  ...

Kangaroo
  ...
```

Interfaces vs. Superclasses
- (Java does allow one inheritance plus multiple interfaces.)

```
<<Interface>>
  Comparable
    ...

<<Interface>>
  Feedable
    ...

Light Infantry
  ...

UBC Student
  ...

Kangaroo
  ...
```

Interfaces vs. Superclasses
The fundamental difference:
- An interface provides no implementations.
- Everything in a (super)class is implemented.

Interfaces vs. Superclasses
The fundamental difference:
- To implement interface, a class must implement everything.
- Subclasses automatically inherit superclass implementation. Can optionally override.

Interfaces vs. Superclasses
Therefore:
- You cannot create objects of an interface type (only references).
- You can create objects of the (super)class type. (You can create references, too.)

Questions
Interfaces vs. Superclasses

The fundamental difference:
- An interface provides no implementations.
- Everything in a (super)class is implemented.

Wouldn’t it be cool to have something in-between?

Abstract Classes

- The classes we have written so far are called concrete classes.
- Abstract classes in Java provide a blend of the concepts of interfaces and inheritance:
  - Some (from none to all) methods are left unimplemented. These are called abstract methods.
  - Instances fields and implemented methods are also allowed.
  - Subclass must implement abstract methods (in order to become concrete).
  - Subclass inherits or overrides other stuff.

Abstract Class Syntax

- To create an abstract class, just add the abstract keyword:

  ```java
  public abstract class Foo {
    ...
  }
  ```

Abstract Method Syntax

- Inside an abstract class, you are allowed to declare abstract methods:
  - Just add the abstract keyword.
  - And put a semicolon instead of a body.

  ```java
  public abstract class Foo {
    ...
    public abstract void display();
    ...
  }
  ```

Abstract Classes vs. Interfaces/Inheritance

- An abstract class with no abstract methods is basically a normal, concrete class.
  - (But Java still thinks it’s an abstract class!)
- An abstract class with all abstract methods is basically like an interface.
  - (But Java still thinks it’s an abstract class!)
- An abstract class with some abstract methods is somewhere in-between.

The Interface-Class Spectrum

Presentation idea due to Steve Wolfman
The Interface-Class Spectrum

- Interface
- Abstract class w/ only abstract methods
- Abstract class w/ no abstract methods
- Concrete class

Can create objects only of concrete classes

Presentation idea due to Steve Wolfman

Example: Inner Class in a Method

```java
public static void main(String[] args) {
    ...
    class Foo {
        private int x;
        public int getX() { return x; }
    }
    ...
}
```

Foo is available only inside main.

Foo can access surrounding vars with restrictions.

Questions

Inner Classes

- So far, classes are always in a file by themselves (with the same name as the class).
- Sometimes, we just want a simple little class for something. It's a pain to have a separate file.
- We can declare classes inside another:
  - Method
  - Class

Inner Class Scope

- Remember the general principle that you can see out through curly braces, but not in.
- Therefore, inner classes should be able to see variables declared around them:
  - Instance variable in same class
  - Local variables if declared inside a method
- However, object of inner class can outlive the code where it was created!
  - Access is to instance variables of object that created inner class.
  - Access only to final local variables. (These can be references to mutable objects, though.)