

Abstract Classes

CSE 114 INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

Announcements

Today: Abstract classes

Recommended reading for this slide set: Chapter 13 of Liang

Abstract classes

If a class has at least one method defined as abstract, the class is called an abstract class.

An abstract method is a method that has only its signature defined without its body - just like the methods in an interface.

See [abstract/Benchmark.java](#)

- [abstract/MethodBenchmark.java](#)

Interfaces

An interface is a **pure abstract class**. That is, an interface does not include any implementation of any method in it.

It only contains the design aspects, i.e., only abstract methods (without the keyword `abstract` although it can be included optionally).

Plus, an interface is defined as an interface, not as a class.

- Of course, you can have an abstract class that has all of its methods defined as abstract.
- In that case, it would be more useful if it is turned into an interface. Why is that?

Class (regular class)

A class contains a full implementation of **all** of the methods that are declared in it or in its super-classes and super-interfaces.

That is, a class cannot have any method (whether it is declared in it or in one of the inherited interfaces or in one of the inherited abstract classes) that is **not** fully implemented.

So, we call the usual class that we have been using a '**concrete**' class (as opposed to an 'abstract' class).

Abstract class

If a class is neither an interface nor a concrete class, then it is an **abstract** class.

That is, an abstract class has some methods implemented in it while there is at least one method not-implemented (as in an interface).

Obviously, we cannot make any instances (objects) of an abstract class just like we cannot make any instances (objects) of an interface.

So:

- An **interface** is a design that is **not implemented at all**
- An **abstract class** is a design that is **partially implemented**
- A **(concrete) class** is a design that is **fully implemented**

Abstract class (cont.)

If a method has the 'abstract' keyword in front of it, it is an abstract method.

If there is at least one abstract method in a class, then the class is called abstract class and we also add the keyword 'abstract' in front of the class declaration to indicate that.

final class or method

Sometimes, a class designer can decide to make a method to be 'final' to tell its subclasses that they are **not** allowed to override the definition given in the superclass.

The 'final' modifier can also be applied to an entire class.

A final class cannot be extended at all. For example, given this class definition:

```
public final class Standards {  
    // class internals  
}
```

The class **Standards** cannot be used in the '**extends**' clause of another class.

Ex: Abstract class w/ all abstract methods

An example of an abstract class which has all of its methods defined as abstract

See [abstract/Shape.java](#) (Shape as a class, not an interface)

[abstract/Circle.java](#) (these are given for completeness)

[abstract/Box.java](#)

[abstract/Point.java](#)

You would **not** use an abstract class like this in your programming practice though. I created this example just to illustrate the point. Since a class can inherit only one class in Java, you will lose the opportunity of inheriting another class if you defined Shape as a **class** and inherited it. You would want to define it as an **interface** so that it can be inherited in addition to another class.

Ex: Abstract class w/ only some abstract methods

This is the usual kind of an abstract class

In this example, [Shape](#) is an abstract class, not an interface

See [abstract2/Shape.java](#)

[abstract2/Square.java](#)

[abstract2/Circle.java](#)

[abstract2/Rectangle.java](#)

[abstract2/UseShape.java](#)

Here, [Shape](#) is a [class](#) that implements [Comparable](#)

Ex: Interface

[Shape](#) as an interface (same as what we saw in one of our earlier lectures)

See [abstract3/Shape.java](#)

[abstract3/Box.java](#)

[abstract3/Circle.java](#)

[abstract3/Rectangle.java](#)

[abstract3/Point.java](#)

[abstract3/UseShape.java](#)

What is an OO Programming Language?

Here is a paper entitled “[What is an Object- Oriented Programming Language?](#)” by Kathleen Fisher and John C. Mitchell.

It is a very good paper and the first 2 sections of the paper will be readable for you based on what we have studied so far.