CS601: Principles of Software Development

Abstract Classes.

Olga A. Karpenko

Parts of this presentations are based on the Java Solftware Solutions book by Lewis&Loftus.

Abstract methods

- You can declare a method but not define it
  - The body of the method is missing

- Called an “abstract method”

```
public abstract void draw(int size);
```

Abstract Class

- Represents a generic concept, a placeholder in a class hierarchy
- Is “incomplete”
- Cannot be instantiated

```
public abstract class Shape
{
    // class contents
}
```

Abstract Class

- Often contains abstract methods with no definitions
- Typically contains non-abstract methods with full definitions
- Does not have to contain abstract methods
  - simply declaring it as abstract makes it so

Example

```
public abstract class Animal {
	private String name;
	public Animal(String name) {
	    this.name = name;
	}

	abstract int eat();

	abstract void breathe();

	// non-abstract methods...
	public String toString() {
		return name + "eats " + eat();
	}
}
```

- This class cannot be instantiated
- Any non-abstract subclass of Animal must provide the eat() and breathe() methods

The slide is based on the presentation of Prof. David Matuszek

Abstract Classes

- The child of an abstract class must either
  - override the abstract methods of the parent,
  - be declared as abstract
- An abstract method cannot be defined as `final` or `static`

Why have Abstract Classes?

- Good for defining a general category containing specific, "concrete" classes
- Example: Superclass class Shape
  - Subclasses: Oval, Rectangle, Triangle, etc.
- You don't want to allow creation of a "Shape"
  - Only *particular* shapes make sense, not *generic* ones

The slide is based on the presentation of Prof. David Matuszek

Why have Abstract Classes?

- If Shape is not abstract
  - Shape should *not* have a draw() method
  - Each subclass of Shape *should* have a draw() method

- Suppose we define an object reference variable figure of type "Shape" that refers to a Circle:

```
Shape figure = new Circle(1.0);
figure.draw(); // compile error
```

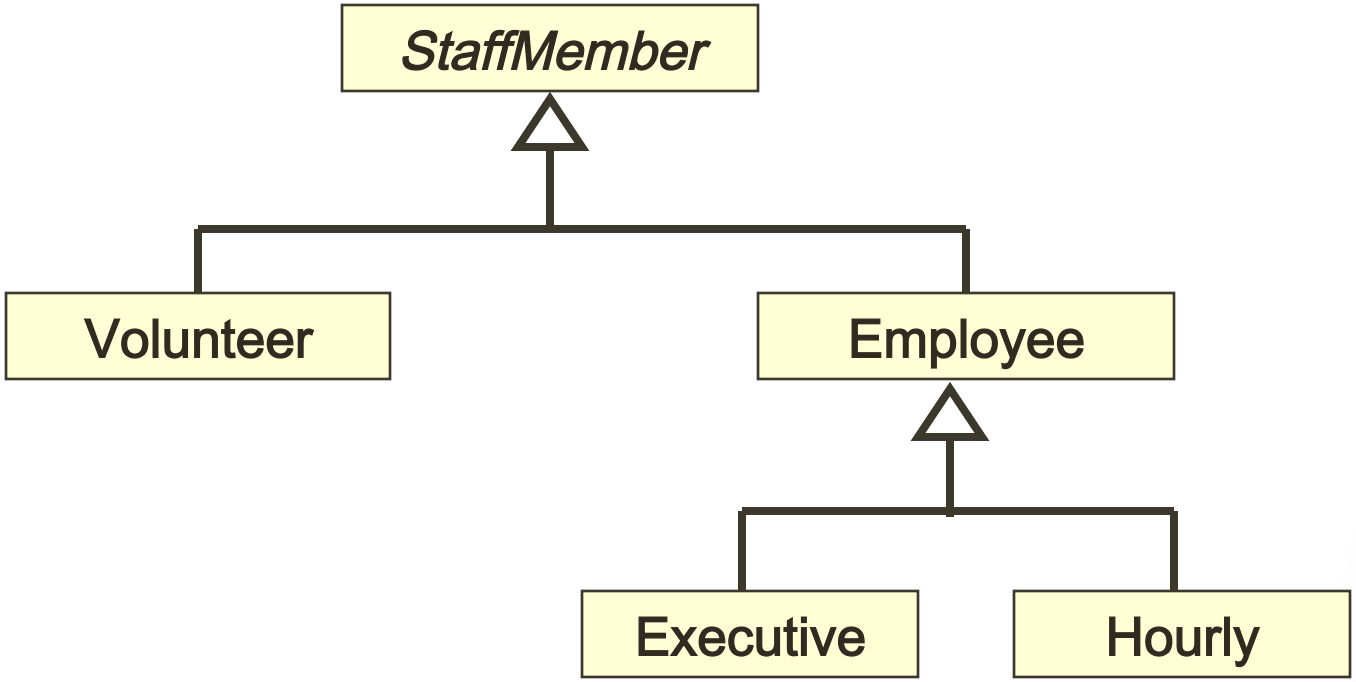
The slide is based on the presentation of Prof. David Matuszek

Solution

- Give Shape an *abstract* method draw()
- Now the class Shape is abstract
  - The figure variable cannot refer to a Shape object
  - Any subclass object (ex: Star object) *will* have the draw() method
  - The Java compiler is happy
    - can depend on figure.draw() being a legal call

Example

- Consider the following class hierarchy:



Polymorphism via Inheritance

- Let's look at an example that pays a set of diverse employees using a polymorphic method

- See `Firm.java`
- See `Staff.java`
- See `StaffMember.java`
- See `Volunteer.java`
- See `Employee.java`
- See `Executive.java`
- See `Hourly.java`

Interface vs. abstract class

	Interface	Abstract class
Variables	Only constants, Everything public	Constants and variable data, Any access modifier
Methods	Abstract or, in Java 8, default	Abstract or concrete

This material is courtesy of Professor Evan Korth

Interface vs. abstract class (cont)

	Interface	Abstract class
Inheritance	A subclass can implement many interfaces	A subclass can inherit only one class
	Can extend numerous interfaces	Can implement numerous interfaces
	Cannot extend a class	Extends one class

This material is courtesy of Professor Evan Korth

Interfaces and abstract classes

- An interface is a contract:
  - Programmer of the interface: "*I accept things looking that way"*
  - User of the interface: "*OK, the class I write looks that way*".
- Programmer of abstract class:

"these classes should look like that, and

they got that in common, so fill in the blanks!"

From http://stackoverflow.com/questions/1913098/what-is-the-difference-between-an-interface-and-abstract-class