ch8: More on Polymorphism

1 Feb 2008
CMPT166
Dr. Sean Ho
Trinity Western University

Protected access

- In Java, the choices for access modifier are:
  - private: only code in this class defn can access
  - (none): code in other classes in same package
  - protected: subclasses can access
  - public: everyone
- Protected:
  - An instance of a subclass can access its own members from the superclass, but not that of other instances
    - e.g., superclass in a different package

Package vs. friend

- Default in Java is package access:
  - All classes with same package declaration at top
  - No declaration: default package is current directory
- C++ doesn't have packages like Java does, but you can grant friendship to other classes:
  - class myClass {

```
friend yourClass;          // allow yourClass access to myClass
friend void QueryData();   // allow only to this method in yourClass
```

  - Friends have access to private and protected elements
  - Most people frown on friending (breaks security)

Multiple inheritance

- Some languages (C++) allow a subclass to inherit from more than one superclass:

```
class Horse { public void eat(); }
class Donkey { public void eat(); }
class Mule : public Horse, Donkey {}        // it's both!
```

- How do disambiguate name collisions?

```
myMule.eat();      // which one?
```

  - Specify superclass name:

```
myMule.Horse::eat();
```

- C++, Python: arity is multiple. Java: arity is single.

Abstract vs. concrete classes

- Abstract classes:
  - Too generic to define a real object
    - e.g., TwoDimensionalShape
  - Not intended to be directly instantiated
    - Java can enforce this: use abstract keyword
    - abstract classes can have abstract methods:
      - No body defined; each subclass must implement
- Concrete classes:
  - Subclass of an abstract class, meant to be instantiated
    - e.g., Square, Circle, Triangle

Example: TwoDimensionalShape

- Abstract superclass: TwoDimensionalShape
  - Abstract method: draw()

```
abstract public class TwoDimensionalShape {
    abstract public void draw();      // no body
```

- Concrete subclasses: Circle, Square, Triangle
  - Each provide own implementation of draw()

```
public class Circle extends TwoDimensionalShape {
    public void draw() { drawOval( x, y, r, r ); }
}
public class Square extends TwoDimensionalShape {
    public void draw() { drawRect( x, y, w, h ); }
}
```

Interfaces

- Define a set of abstract methods

```
public interface drawableShape {
    public abstract void draw();
    public abstract double area();
}
```

- Classes implement these methods

```
public class Circle implements drawableShape {
    public void draw() { drawOval( x, y, r, r ); }
    public double area() { return 2 * Math.PI * r * r; }
```

- We've already been using the actionListener interface

Abstract classes vs. interfaces

- Abstract superclasses declare identity:
  - "Circle is a kind of TwoDimensionalShape"
  - Each class can have only one superclass
    - No multiple inheritance in Java
  - Inherit methods, attributes; get protected access
- Interfaces declare capability:
  - "Circles know how to be drawableShapes"
  - May implement multiple interfaces
  - Interfaces are not ADTs (abstract data types)