Polymorphism and Interfaces

1 Feb 2011
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY
WESTERN
UNIVERSITY

Outline for today

- "Has a" vs. "is a" vs. "is a kind of"
- Polymorphism and late binding
- Keyword final
- Superclass constructors
- Multiple inheritance
- Abstract superclasses vs. interfaces

Relationships in OO design

- Recall classes are user-defined container types
- A subclass inherits attributes and methods from the superclass
- Subclasses are specializations of the superclass: “A is a kind of B”
- Instances are examples of a class: “A is a B”
- Attributes are components or parts of a class: “A has a B”

Example

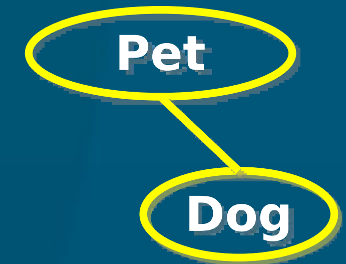
- class Mammal { Heart h; void eat(); }
- class Dog extends Mammal { void bark(); }
- class Cat extends Mammal { void meow(); }
- Dog fido = new Dog();
- Cat smokey = new Cat();

- "A Dog is a kind of Mammal."
- "fido is a Dog."
- "fido is a Mammal."
- "fido has a Heart."
- "smokey can meow()."
- "smokey can eat()."

'final' on methods/classes

- The 'final' keyword:
  - On attributes/variables: constant value
    - **public final float hourlyWage = 17.5;**
  - On methods: cannot be overriden by subclass
    - **public final void eat(Food f) {**
  - On classes: cannot be subclassed
    - **public final class Dog {**

Superclass constructor



- public class Dog extends Pet

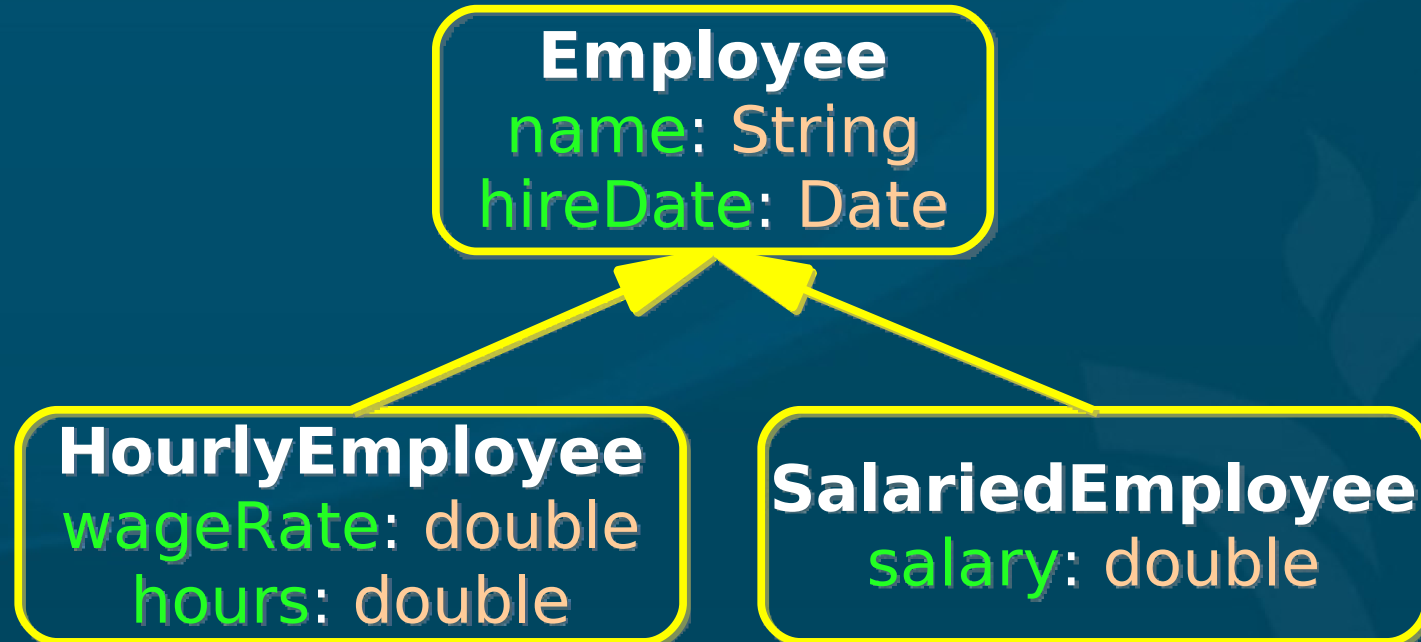
- A subclass' constructor does not inherit/override the superclass constructor
- But it implicitly calls the superclass constructor:
  - public Dog() { /* implicitly calls Pet() */ }
  - Can also explicitly call it with super():
  - public Dog() {

```
super();        // explicitly call Pet() first
...             // do Dog-specific stuff here
```

  - This is used to pass parameters to super()

Employee example (Savitch)

- Each class has set/get methods for its attribs
- .toString(): overrides superclass definitions
- .equals(): check for equality with another object
  - Takes an Object as the parameter
  - Object is the superclass of everything



Designing for polymorphism

- Spend time thinking carefully and designing the class hierarchy: “A is a kind of B” relationships
    - **Dog is a kind of Pet**
- Design your functions to act at the highest level of abstraction possible (highest superclass)
    - Methods of the superclass:
        - **Pet.eat() // inherited by both Dog and Cat**
    - Functions that take objects as params:
        - **public void feed(Pet p) { … }**
    - Functionality is inherited by subclasses
        - **feed(fido); feed(smokey);**

Multiple inheritance (arity)

- 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 version of eat()?
```

  - Specify superclass name:

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

- In C++, Python: arity is multiple.
- In Java: arity is single
  (each class has exactly one superclass)

Abstract vs. concrete classes

- Abstract classes:
  - Too generic to define a real object
    - e.g., TwoDimensionalShape
  - Not intended to be directly instantiated
    - Enforce this in Java with the 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

e.g: 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; }
```

- e.g., Java Swing programs that handle events implement 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), and do not contain attributes