Inheritance

29 Jan 2010
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Superclasses and subclasses

- Attribute: "has a" relationship:
  - A Car has a steeringWheel
- Subclass: "is a kind of" relationship:
  - A Convertible is a kind of Car
  - Inheritance relationships form tree-like class hierarchies
  - "extends": more specific, less inclusive, more complex
- Polymorphism: write once
  - changeOil() method works on all Cars, not just Convertibles

Why use inheritance?

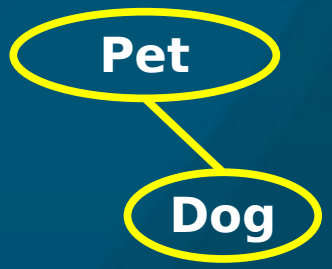
- Reusability
  - Create new classes from existing ones
    - Absorb attributes and behaviours
    - Add new capabilities
- Polymorphism
    - Enable developers to write programs with a general design
    - A single program can handle a variety of existing and future classes
    - Aids in extending program, adding new capabilities

Subclassing in Java

- When declaring a class, indicate its superclass (parent):
  - **public class Dog extends Pet { ....**
    - A Dog is a kind of Pet
    - Inherits everything Pet has
    - Can add Dog-specific attribs/methods
    - Can override general Pet methods with Dog-specific versions

Using subclass instances

- An instance of a subclass can be treated as an instance of the superclass:
  - **Pet fluffy = new Dog();**
  - Cannot do vice-versa:
    - **Dog myDog = new Pet(); // doesn't work!**

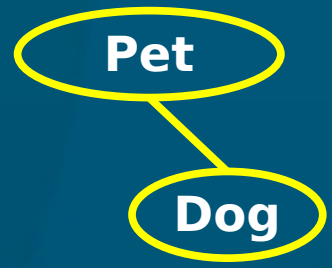


- instanceof checks the class of an object:
  - **if (fluffy instanceof Dog) { …**
- A superclass reference may be downcast back to the subclass if appropriate:
  - **// this is ok: fluffy is really a Dog**
  - **Dog myDog = (Dog) fluffy;**

Overriding methods

- A subclass can override a method defined by the superclass
  - Every Pet knows how to speak()
  - But Dogs speak() differently from Cats
  - Subclasses override the speak() method
- Late binding: which version of speak() to use?
  - Decided at run-time
- Polymorphism: same code works on several different types, all subclasses of the same parent
- Contrast with overloading (type signature)

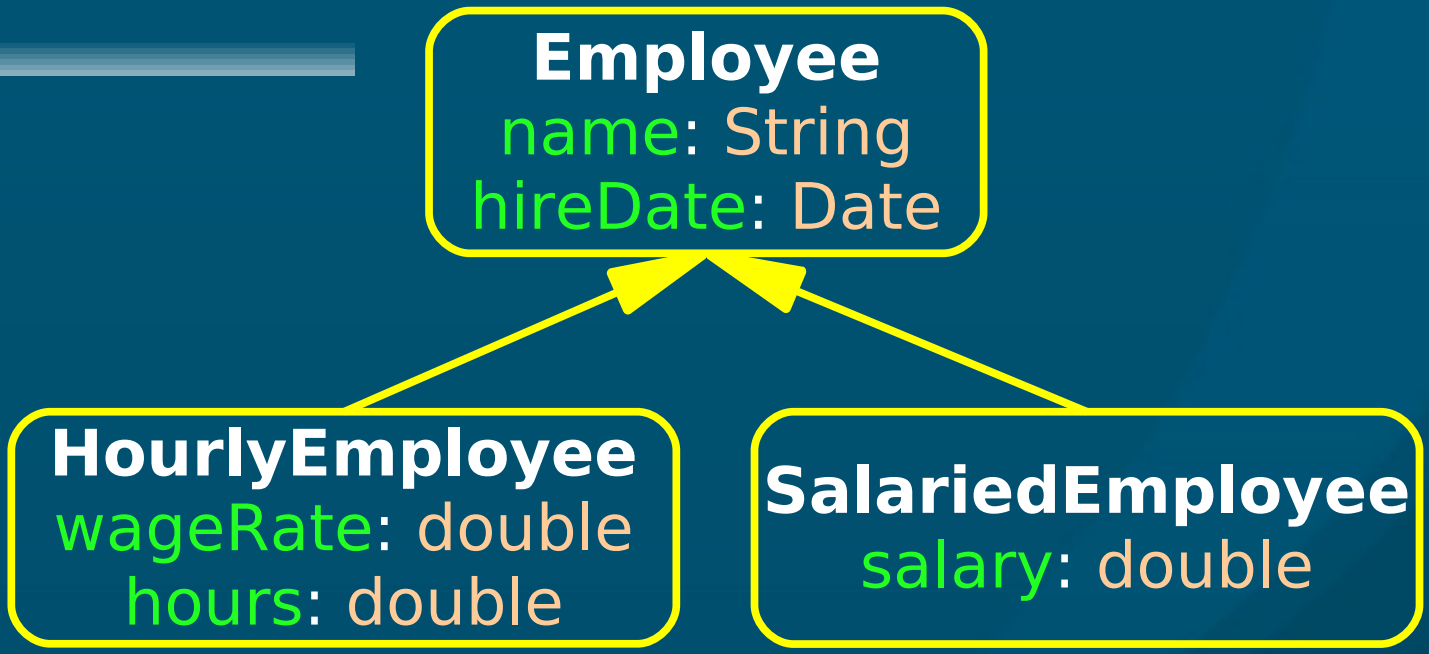
Constructors



- **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 with super()
    - **public Dog() {**
      - **super(); // explicitly call Pet() first**
      - **... // do Dog-specific stuff here**
    - Arguments can also be passed 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.speak() // Dog, Cat inherit**
  - Functions that take objects as params:
    - **public void feed(Pet p) { … }**
  - Functionality trickles down to subclasses

'final' on methods/classes

- The 'final' keyword:
  - On attributes/variables: constant value
  - On methods: cannot be overriden by subclass
  - On classes: cannot be subclassed