Subclasses: Private is Private!

- Private = only in class
- Excludes subclasses too!
- How access fields?
  - getters and setters
  - Use super() to initialize

public class Animal {
  private String name;
  private int age;
}

public class Cat extends Animal {
  age = a;
  name = n;
}

Mixing Subclasses in Vector

QUESTION:
Which method is called by v.get(0).toString()?
- A: One in (hidden) Object part of @105dc
- B: One in Animal part of @105dc
- C: One in Cat part of @105dc
- D: One in Dog part of @3cf92
- E: None of these

Apparent Type of an Expression

The call
v.get(k).getWeight()

is illegal (will not compile).

The apparent type of v[k] is Animal
- Does not declare getWeight()
- Does not inherit getWeight()

Casting Up and Down the Class Hierarchy

- Review of casting
  - (int) (5.0 / 7.5)
  - (double) 6
  - double d = 5;  // automatic cast
- Can also cast class types:
  - Animal h = new Cat("N", 5);
  - Cat c = (Cat) h;

The Class Hierarchy
(→ means "extends" or "is a kind of")

Implicit Casting in the Class Hierarchy

public class Animal {
  /** "this is older than b" */
  public boolean isOlder(Animal b) {
    return this.age > b.age;
  }
}

Cat c = new Cat("C", 5);
Dog d = new Dog("D", 6);
c.isOlder(d) ?????

Casts up the hierarchy are automatic.
Dog
Cat
c.isOlder(d)
Cat
Dog
return
Animal ob = 
if (!(
public
values in its fields as this Animal */
/** Yields: "this is older than h" */
public boolean isOlder(Animal h) 
{ return this.age > h.age; }
}
/* A952
  ^
Cat c = new Cat("C", 5);
Dog d = new Dog("D", 6);
if (c.isOlder(d)) ?????

Real type of h:
  • Semantic Property
  • Type of the folder
Apparent type of h:
  • Syntactic Property
  • Type that is declared

How Do We Resolve h.toString()?

public class Animal {
  /** = "this is older than h" */
  public String toString() { return h.toString(); }
}
/* A105dc
  ^
Cat c = new Cat("C", 5);
Dog d = new Dog("D", 6);
if (c.toString() ==d.toString()) ?????

Determined by the real type of h

How to Override equals(Object)

public class Animal {
  /** = "this is older than h" */
  public boolean equals(Object o) { return (o instanceof Animal) &&
    ((Animal) o).age == this.age; }
  }
/* A105k
  ^
Cat c = new Cat("C", 5);
Dog d = new Dog("D", 6);
c.equals(d) ?????

May want to define equals() in Cat and Dog:
A cat is not equal to a dog, even if they have the same name and age!

What Can Variable h reference?

public class Animal {
  /** = "this is older than h" */
  public boolean isOlder(Animal h) 
{ return this.age > h.age; }
}
/* 100de
  ^
Cat c = new Cat("C", 5);
Dog d = new Dog("D", 6);
d.isOlder(c) ?????

• Apparent type determines what methods calls are legal
  • Cannot call h.getWeight();
  • This gives a syntax error
  • Even though real type is Cat

Casting Down the Class Hierarchy

public class Animal {
  /* If Animal is a cat, return weight; else return 0 */
  public static double checkWeight(Animal h) {
    return 0;
    }
    // h is a Cat
    Cat c = (Cat) h; // Downward cast
    return c.getWeight();
  }
/* 100de
  ^
Cat c = (Cat) h; // Downward cast

(Dog) h would lead to a runtime error.
You can’t cast an object to something that it is not!

Overriding Versus Overloading

public class Animal {
  public boolean equals(Object o) { return (o instanceof Animal) &&
    ((Animal) o).age == this.age; }
  }
/* 105dc
  ^
Cat c = new Cat("C", 5);
Dog d = new Dog("D", 5);
c.equals(d) ?????

• Method calls match on
  • Name of the method
  • Types of the parameters
• If no match:
  • Upscasts the arguments
  • Searches again for match