April 12: Interfaces

CMPSCI 121, Spring 2012

Introduction to Problem Solving with Computers

Prof. Learned-Miller

Logistics

- No second midterm.
- Fiasco with Crypto problem.

Today...

- More on inheritance
- Interfaces

Overriding

- What happens when there are multiple choices for a method?
  - Why would this occur?
- toString() example.
  - class Dog extends Animal...
  - class Animal extends Object...

Overriding

- What happens when there are multiple choices for a method?
  - Why would this occur?
- toString() example.
  - class Dog extends Animal...
  - class Animal extends Object...
- Java always uses the MOST SPECIFIC method possible....

Polymorphism

- Can we do something like this:

```
Dog d=new Dog();
Animal a;
a=d;                    // Is this allowed? (yes)
Dog d2=a;               // How about this? (no)
```

Wrong Subtyping

- We can cast a superclass object to subclass, but need to be sure type is right
  - e.g., casting to-and-from Object is common for writing general code
- A Dog IS-A Animal, but not the other way around
  - class Dog extends Animal {...}
    class Cat extends Animal {...}

    Dog d = new Animal(); // Can't do this. An Animal is not nec. a Dog.
    Dog d = new Cat(); // Can't do this. A Cat is not a Dog.

  - Animal a = new Dog(); // This is fine. A Dog is an Animal.
    Dog d = (Dog) a; // Fine. This *particular* Animal is a Dog.
    Cat c = (Cat) a; // Not OK. This *particular* Animal is NOT a Cat.

Polymorphism

- ```
  class Animal {
        String name() {
              return “dunno”; }
        }

  class Dog extends Animal {
    String name() {
       return “Dog”;
    }
  }

  class Cat extends Animal {
    String name() { return “Cat”; }
  }

  class Farm {
    static report(Animal a) {
      System.out.println(“I am a ” + a.name());
    }
  }
  Farm.report(new Dog());  // Prints “I am a Dog”
  Farm.report(new Cat());  // Prints “I am a Cat”
  ```
  ```

Dynamic dispatch.

- More generally:
  - Animal a = new Dog();
    a.meth(); // method called is that of Dog, if it exists
  - Dog d = new Dog();
    ((Animal)d).meth(); // ***Still*** calls Dog method (if it exists).
  - Animal a = new Dog();
    ...
    Dog d = (Dog) a;
    a.meth(); // Calls Dog method.
    d.meth(); // Calls Dog method.
- Also called "dynamic dispatch"
  - we know what method to call only at run-time ("dynamically") as it depends on the actual type of the object (what's "in the box", not the name on the box, as I said in lecture).

Polymorphism

```
Dog d=new Dog();
((Animal) d).method();
```

Polymorphism

```
Animal a=new Dog();
a.method();
```

Inheritance: UsedCar Class

```
 1 public class UsedCar extends Car{
 2
 3   private int year; // year of manufacture
 4
 5   public UsedCar(String whatMake, double cap, double amt, int yr){
 6     super(whatMake,cap,amt);
 7     year = yr;
 8   }
 9
10   public int getYear(){
11     return year;
12   }
13 }
```

When is inheritance used?

- The first type of inheritance:
  - When you have one useful class...
    - Car
  - and you want to add some stuff to (extend) the class:
    - UsedCar

Other uses of inheritance

- Suppose we have a method to find the oldest Infant in an Array of Infants.

  int oldest(Infant[] array) {...

- and a method to find the oldest car

  int oldest(Car[] array) {...

- and a method to find the oldest boat

  int oldest(Boat[] array) {...

Redoing the same work

- All these methods will look the same.
- To avoid this, write one function for "Ageable" objects. Then they can all use the method:

```
int oldest(Ageable[] array) {...
```

Interfaces

- Used when you want to add certain generic capabilities or attributes to a class
- Can use one interface to add the SAME capabilities to multiple classes

Inheritance vs. Interfaces

- If A inherits from B,
  then “A is a B”.
  - Example:
    “Mammal” inherits properties from “Animal”,
    so “Mammal” is an “Animal”.
- If A implements B,
  then A has capabilities described by B.
  - Example:
    PlumberPerson implements CanPlumb
    so PlumberPerson has plumbing capabilities.
    or...
    HandyMan implements CanPlumb
    so HandyMan has plumbing capabilities.

Interface Example:

```
public interface Scoring{

  public double getScore();

  public void setScore(double newScore);
}
```

-Kind of like a class, but can't make one of these
-Doesn't specify implementation of methods, just what they should do.

```
public class CookieSeller implements Scoring
{
  private String name;
  private double boxesSold;

  public CookieSeller(String n, double sold)
  {
    name = n;
    boxesSold = sold;
  }

  public String getName()
  {
    return name;
  }
  public double getBoxesSold()
  {
    return boxesSold;
  }

  public void setName(String newName)
  {
    name = newName;
  }
  public void setBoxesSold(double sold)
  {
    boxesSold = sold;
  }
  public double getScore() // implements interface method
  {
    return boxesSold;
  }

  public void setScore(double sold)// implements interface method
  {
    boxesSold = sold;
  }
}
```

One method for multiple classes.

```
public static int scoreMax(Scoring[] theArray){
  // returns position of entry in array theArray with highest score
  // array theArray is an array of objects from class that implements
  // Scoring interface
  int highPos = 0;
  for(int j = 1; j < theArray.length; j++){
    if (theArray[j].getScore() > theArray[highPos].getScore())
      highPos = j;}
  return highPos;
}
```

One method for multiple classes

```
public class Scorefns  {
  // contains methods that exploit the Scoring interface

  public static int scoreMax(Scoring[] theArray){
    // returns position of entry in array theArray with highest score
    // array theArray is an array of objects from class that implements
    // Scoring interface
    int highPos = 0;
    for(int j = 1; j < theArray.length; j++){
      if (theArray[j].getScore() > theArray[highPos].getScore())
        highPos = j;}
    return highPos;
  }
}
```