More Inheritance

7/2/2009

Important Dates

- Project 1 - Check-off
  - Thursday 7/02/2009 ready BEFORE lab
- Review Session
  - Sunday 7/05/2009 – 306 Soda 1-4pm
- Midterm 1
  - Tuesday 7/07/2009 – 10 Evans 5-6pm
  - Covers everything through Monday’s lab
- Project 2 released
  - Thursday 7/09/2009
- Project 1 due
  - Monday 7/13/2009 – 10pm

equals(Object o)

• Did you understand Wednesday’s lab?

public class Animal {
    public void sniff(Animal a) {
        System.out.println("Animal sniff Animal");
    }
    public void sniff(Dog d) {
        System.out.println("Animal sniff Dog");
    }
}

Inheritance

• Compilation:
  - CALLER: It makes sure that the static type of the object has the appropriate method
  - ARGS: It makes sure that the method takes in the static type of the arguments

• Run-time:
  - CALLER: When you call a method on an object it looks for the method starting at the object’s dynamic type
  - ARGS: When you pass an object as an argument, it looks for a method with that static type

Animal a1 = new Dog();
Animal a2 = new Dog();
Dog    d1 = new Dog();
a1.sniff(d1); ← Uses a1’s dynamic type
d1.sniff(a1); ← Uses a1’s static type
a1.sniff(a2); ← Uses a1’s dynamic type
Uses a2’s static type

Dog
age 0
bites false
What about `equals(Object o)`

- ArrayList was using an Object reference

```java
Object o1 = new Animal();
Object o2 = new Animal();
o1.equals(o2);
```

- If others will call your methods with more generic references, you want to provide a method that takes in an `Object`.

Interfaces

- I have a GREAT idea!
- Everyone will want to make Pets that can
  - `eatKibble()`
  - `sitOnLap()`
- How?!?!
  - I don’t care!!!!

The Pet interface

```java
public interface Pet {
    public void eatKibble();
    public void sitOnLap();
}
```

Dog can implement
the Pet interface

```java
public class Dog extends Animal implements Pet{
    public void eatKibble() {
        System.out.println("yum! kibble!"); 
    }
    public void sitOnLap() {
        System.out.println("zzzzz");
    }
}
```

Cat can implement
the Pet interface

```java
public class Cat extends Animal implements Pet{
    public void eatKibble() {
        System.out.println("got milk?");
    }
    public void sitOnLap() {
        System.out.println("purrppppp");
    }
}
```
We can have a Pet remote control (Pet reference)

```java
Pet p1 = new Dog();
Pet p2 = new Cat();
p1.eatKibble();
p2.sitOnLap();
```

Can we have a Pet object?

```java
Pet p1 = new Pet();
```

Interfaces can be cool!!!

```java
Pet[] myPets = new Pet[2];
myPets[0] = new Dog();
myPets[1] = new Cat();
for (Pet onePet : myPets) {
    onePet.eatKibble();
}
```

Example

- The `sort` method of the `Array` class promises to sort an array of objects, but under one condition: the objects in the array must implement the `Comparable` interface:

```java
public interface Comparable {
    int compareTo(Object other);
}
```

Summary of Interfaces

- Interfaces don’t implement ANY methods – just put a semicolon at the end of the method
- Classes can implement multiple interfaces
- To implement an interface you must write all of the methods that the interface defines

Cookies!

- Cookie
  - Girl Scout Cookie
  - Chocolate Chip
  - Thin mint
  - Shortbread
Abstract Classes
(less lazy than Interfaces)

- I have a GREAT idea!
- Everyone will want to make Cookies that can have these methods:
  - ingredients()
  - isDelicious()
- How?!?
  - isDelicious() is pretty simple, I’ll write that one
  - ingredients()?!? Sounds too hard! I’ll make that abstract

ChocolateChipCookie can extend the Abstract class Cookie

```java
public class ChocolateChipCookie extends Cookie{
    public ChocolateChipCookie(){
        this.delicious = true;
    }
    public String[] ingredients()
    {
        String[] ingredients = {"Sugar", "Chocolate"};
        return ingredients;
    }
}
```

Can we have a Cookie object?

```java
Cookie pl = new Cookie();
```

Abstract Class Summary

- Label an Abstract class as abstract
- Label any methods that you don’t want to implement as abstract
  - Your children MUST write all of the abstract methods
- Instance variables in the abstract class will be available in the child class
- You can only extend one Class (abstract or otherwise)