Announcements/Follow-ups

• Lab09 tomorrow
  – Practice with generics and inheritance
• P6 due + P7 posted on Wednesday
• Quiz5 Thursday (last one!)
  – Study questions posted today
• Follow-ups
  – MutableDouble example
  – HashTable example
  – Generics and new
Inheritance

• What is it?
  – The ability to define a new class that automatically possesses the properties of an existing class
    • Fields
    • Methods
    • Nested classes

• What is it good for?
  – Conceptually organizing code
  – Reusing code
    • RedundantInterfacing example
Basic terminology

• Any of the following refer to the existing class:
  – “Base” class
  – “Super” class
  – “Parent” class

• Any of the following refer to the new class:
  – “Derived” class
  – “Sub” class
  – “Child” class

• The derived class “inherits” the members of the base class.

• mySubObject “Is-a” mySuperObject
Inheritance

An instance of the base class

Base

superData 1

An instance of a different sub class

Sub2

superData 3

subData2 true

An instance of a one sub class

Sub1

superData 4

subData1 null
Basic syntax and usage

• The extends keyword:
  public class SubClass extends SuperClass {...}
  – Mother/Child example
  – ReusableExtending example
  – MyBigInt example

• Object is the default super-class when extends is omitted

• Interfaces can extend other interfaces as well
Basic syntax and usage

• The final keyword (for classes):
  – Prevents extension by any other class
  – Truly immutable classes (String, Integer, etc.) are final
    • Otherwise MyMutableString could extend String, and violate the “is-a” relationship
  – Final children are leaves in the tree of inheritance
• Casting rules: based on the “is-a” relationship again
  – Mother/Child example
There is one more access modifier: protected.

A sub-class can only access its own copy of a protected field.

- Mommy/Daddy/Baby example

http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html
Overriding instance methods

• Instance methods with identical declarations in a sub-class and super-class
  – sub-class version “overrides” super-class version
  – String toString()
  – boolean equals(Object other)
  – Mommy/Baby void invest()

• The analogy to static methods is called “hiding”.

• The final keyword (for methods): If the class is not final, sub-classes may exist, but they cannot override final methods.
The super keyword

• super is like this, but it refers to inherited members.
• super and this can also be used in one constructor to call another constructor
  – this(...) calls another constructor in the sub-class (with a different signature)
  – super(...) calls a constructor in the super-class (potentially with the same signature).
    • Must be the first line of constructor body
    • If omitted, super() without arguments is called by default. In this case, a zero-parameter constructor must be defined in the base-class, or there is a compile-time error
• Super example
The abstract keyword

• A middle-ground between inheritance and interfaces. Useful in the following situation:
  – Some members will be identical for all sub-classes, so they should be implemented in the base class.
  – Other members must exist, but their implementations will depend entirely on the sub-classes. So they cannot be implemented in the base class.
• The abstract keyword identifies the second category.
• A class with abstract members must be declared abstract itself.
• Abstract example
The `instanceof` operator

• Similar concept to `getClass()`, but with important differences
  – Usage: `ref1 instanceof MyClass`
  – True when the run-time type of `ref1` is `MyClass` or any `sub-class of MyClass`
  – Mommy/Baby inheritance tests example