CompSci 100e
Program Design and Analysis II

February 10, 2011

Prof. Rodger
Announcements

• What is due/coming up?
  – Apt due Feb 15
  – Markov Assignment due Feb 17
    • Will look at parts of it in lab
  – Test on Feb 22

• Finish slides from last time
Simple Inheritance

• Allows you to reuse code
• Start with a Class (superclass)
• Create another class that extends the class (subclass)
• The subclass can use the methods from the superclass or override them (use the same name, but the code is different)
• If the subclass redefines a superclass method, can still call the superclass method with the word “super” added.
Access to Instance Variables (state)

• public
  – Any class can access

• private
  – subclasses cannot access

• protected
  – subclasses can access
  – other classes cannot access
Example

- Student (superclass)
- DukeStudent (extends Student)
- CosmicStudent (extends DukeStudent)

- Look at code, what is the output?
More on Inheritance - Abstract Class

• Abstract class – class that is declared abstract
• Cannot be instantiated – cannot create an object for this class
• Another class must extend this class
• May have some methods declared abstract
  – Abstract methods have no bodies
  – Those methods have to be implemented in the class that extends the abstract class
• Example:
  – AbstractModel.java is an abstract class
  – MarkovModel extends AbstractModel
More on Inheritance - Interface

- Class that is declared as an interface
- A group of related methods with empty bodies
- To implement the interface, your class would implement the methods for those named in the interface.

- Example

```java
public interface IModel {
    public void initialize(Scanner s);
    public void process(Object o);
}

  // AbstractModel Implements IModel
```

CompSci 100 Fall 2010
What can an Object do (to itself)?

  - Look at java.lang.Object
  - What is this class? What is its purpose?

- `toString()`
  - Used to print (System.out.println) an object
  - **overriding `toString()`** useful in new classes
  - String concatenation: `String s = "value " + x;`
  - Default is basically a pointer-value
What else can you do to an Object?

- `equals(Object o)`
  - Determines if guts of two objects are the same, must override, e.g., for using `a.indexOf(o)` in `ArrayList a`
  - Default is `==`, pointer equality

- `hashCode()`
  - Hashes object (guts) to value for efficient lookup

- If you're implementing a new class, to play nice with others you **must**
  - Override `equals` and `hashCode`
  - Ensure that equal objects return same `hashCode` value
Objects and values

• Primitive variables are boxes
  – think memory location with value

• Object variables are labels that are put on boxes

```java
String s = new String("genome");
String t = new String("genome");
if (s == t) {they label the same box}
if (s.equals(t)) {contents of boxes the same}
```

What's in the boxes? "genome" is in the boxes
Objects, values, classes

- For primitive types: int, char, double, boolean
  - Variables have names and are themselves boxes (metaphorically)
  - Two int variables assigned 17 are equal with ==

- For object types: String, ArrayList, others
  - Variables have names and are labels for boxes
  - If no box assigned, created, then label applied to null
  - Can assign label to existing box (via another label)
  - Can create new box using built-in new

- Object types are references/pointers/labels to storage