

Objectives

- Packages
- Wrapper Classes
- More on Inheritance
 - Abstract Classes
 - Interfaces

Sept 22, 2008

Sprenkle - CS209

1

Review

- What is the syntax for Javadoc comments?
- How do we say that a class or a method cannot be subclassed/overridden?
- How has developing in Eclipse been going?

Sept 22, 2008

Sprenkle - CS209

2

PACKAGES

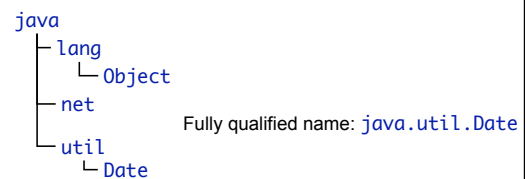
Sept 22, 2008

Sprenkle - CS209

3

Packages

- Hierarchical structure of Java classes
 - Directories of directories



- Use `import` to access packages

Sept 22, 2008

Sprenkle - CS209

4

Standard Practice

- To reduce the chance of a conflict between names of classes, put classes in packages
- Use the `package` keyword to say that a class belongs to a package:
 - `package java.util;`
- Typically, use a unique prefix, similar to domain names
 - `com.ibm`
 - `edu.wlu.cs.logic`

Sept 22, 2008

Sprenkle - CS209

5

WRAPPER CLASSES

Sept 22, 2008

Sprenkle - CS209

6

Wrapper Classes

- **Wrapper class** for each primitive type
- Sometimes need an instance of an Object
 - To use to store in **HashMaps** and other **Collections**
- Include the functionality of parsing their respective data types.

```
int x = 10;
Integer y = new Integer(10);
```

Sept 22, 2008

Sprenkle - CS209

7

Wrapper Classes

- **Autoboxing** – automatically create a wrapper object
 - // implicitly 11 converted to
 - // new Integer(11);
 - Integer y = 11;
- **Autounboxing** – automatically extract a primitive type

```
Integer x = new Integer(11);
int y = x.intValue();
int z = x; // implicitly, x is x.intValue();
```

Sept 22, 2008

Sprenkle - CS209

8

MORE ON INHERITANCE

Sept 22, 2008

Sprenkle - CS209

9

Abstract Classes

- Some methods defined, others not defined
- Classes in which not all methods are implemented are **abstract classes**
 - public abstract class ZooAnimal
- Blank methods are labeled as **abstract**
 - public abstract void exercise();

Sept 22, 2008

Sprenkle - CS209

10

Abstract Classes

- An abstract class cannot be instantiated
 - i.e., can't create an object of that class
- Child class of an abstract class can only be instantiated if it overrides and implements **each abstract method** of its parent class
 - If subclass does not override all abstract methods, it is **also abstract**

Sept 22, 2008

Sprenkle - CS209

11

Abstract Classes

- **static**, **private**, and **final** methods cannot be **abstract**
 - These cannot be overridden by a child class
 - **final** class cannot contain abstract methods
- A class can be abstract even if it has no abstract methods
 - Use when implementation is incomplete and is meant to serve as a parent class for subclass(es) that complete the implementation
- Can have array of objects of abstract class
 - Does dynamic dispatch for methods

Sept 22, 2008

Sprenkle - CS209

12

Abstract Classes

- Use **abstract** when have partial implementation

Sept 22, 2008

Sprenkle - CS209

13

Examples of abstract classes

- Define the abstract methods
 - **Example 1:**
 - `java.net.Socket`
 - `java.net.SSLSocket` (abstract)
 - **Example 2:**
 - `java.util.Calendar` (abstract)
 - `java.util.GregorianCalendar`

Sept 22, 2008

Sprenkle - CS209

14

Better Organization of Game Classes

- `GameObject` should be abstract
 - No default image associated with it
 - `move` method is abstract
- `Human` class should implement `move` method
 - From `GameObject` class

Sept 22, 2008

Sprenkle - CS209

15

Interfaces

- Like abstract classes with **all** abstract methods
 - A set of requirements for classes to conform to
- Pure specification, no implementation
- Classes can **implement** one or more interfaces

Sept 22, 2008

Sprenkle - CS209

16

Example of an Interface

- We have seen before how to make an array of `Chicken` object variables.
- We can call `Arrays.sort()` on an array
- `Arrays.sort()` sorts arrays of any object class that implements the `Comparable` interface
- Classes that implement `Comparable` provide a way to decide if one object is less than, greater than, or equal to another object

Sept 22, 2008

Sprenkle - CS209

17

`java.lang.Comparable`

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

- Any object that is `Comparable` must have a method named `compareTo()`
- Returns:
 - `< 0` for less than
 - `0` for equals
 - `> 0` for greater than
- Similar to Python's `__cmp__` method

Sept 22, 2008

Sprenkle - CS209

18

Implementing an Interface

- In the class definition, specify that the class will **implement** the specific interface

```
public class Chicken implements Comparable
```

- Provide a definition for all methods specified in interface

Sept 22, 2008

Sprenkle - CS209

19

How to determine Chicken order?

- What if made the Chicken class Comparable?

Sept 22, 2008

Sprenkle - CS209

20

Comparable Chickens

One way: order by height:

```
public class Chicken implements Comparable {
    public int compareTo(Object otherObject) {
        Chicken other = (Chicken)otherObject;
        if (height < other.getHeight() )
            return -1;
        if (height > other.getHeight())
            return 1;
        return 0;
    }
}
```

What if otherObject is not a Chicken?

Sept 22, 2008

Sprenkle - CS209

Update Chicken.java 21

Comparable Interface API

- Says what the compareTo() method should do:
 - Return a -1 if the first object is less than the second object (passed as a parameter)
 - Return a 1 if the second object (passed as a parameter) is less than the first object
 - Return a 0 if the two objects are equal
- Says what Java library classes implement **Comparable**

Sept 22, 2008

Sprenkle - CS209

22

Interfaces

- Contain only object (*not class*) methods
- All methods are **public**
 - Implied if not explicit
 - Error to have protected or private (Why?)
- Fields are constants that are **static** and **final**
- Can implement multiple interfaces
 - Separated by commas in definition

Sept 22, 2008

Sprenkle - CS209

23

Testing for Interfaces

- Use the **instanceof** operator to see if an object implements an interface
 - e.g., to determine if an object can be compared to another object using the **Comparable** interface

```
if (obj instanceof Comparable) {
    // runs if whatever class obj is an instance of
    // implements the Comparable interface
}
else {
    // runs if it does not implement the interface
}
```

Sept 22, 2008

Sprenkle - CS209

24

Interface Object Variables

- Can use an object variable to refer to an object of any class that implements an interface
- Using this object variable, can *only* access the interface's methods
- For example...

```
Object obj;
...
if (obj instanceof Comparable) {
    Comparable comp = (Comparable) obj;
    boolean res = comp.compareTo(obj2);
}
```

Sept 22, 2008

Sprenkle - CS209

25

Interface Definitions

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

- Do not *need* to specify methods as **public**
 - Interface methods are **public** by default

Sept 22, 2008

Sprenkle - CS209

26

Interface Definitions and Inheritance

- Can extend interfaces
 - Allows a chain of interfaces that go from general to more specific with each step
- For example, define an interface for an object that is capable of moving:

```
public interface Movable {
    void move(double x, double y);
}
```

Sept 22, 2008

Sprenkle - CS209

27

Interface Definitions and Inheritance

- A powered vehicle is also **Movable**
 - Must also have a **MPG()** method, which will return its gas mileage

```
public interface Powered extends Movable {
    double milesPerGallon();
}
```

Sept 22, 2008

Sprenkle - CS209

28

Constants in an Interface

- If a variable is specified in an interface, it is automatically a constant
 - **public static final variable**

```
public interface Powered extends Movable {
    double milesPerGallon();
    double SPEED_LIMIT = 95;
}
```

- An object that implements **Powered** interface has a constant **SPEED_LIMIT** defined

Sept 22, 2008

Sprenkle - CS209

29

Interface Definitions and Inheritance

- **Powered** interface extends **Movable** interface
- An object that implements **Powered** interface must satisfy all requirements of that interface as well as the parent interface.
 - A **Powered** object must have a **milesPerGallon()** and **move()** method

Sept 22, 2008

Sprenkle - CS209

30

Multiple Interfaces

- A class can implement multiple interfaces
 - Must fulfill the requirements of each interface
- But NOT possible with inheritance
 - A class can only extend (or inherit from) one class

```
public final class String implements
    Serializable, Comparable, CharSequence { ...
```

Sept 22, 2008

Sprenkle - CS209

31

Common Uses of Interfaces

- Define constants for multiple classes /package
 - Something like global constants
- Marker Interface
 - Interface that is empty
 - Use to identify an object that has a certain property

Sept 22, 2008

Sprenkle - CS209

32

Using An Interface or Abstract class

Interfaces

- ✓ Any class can use
 - ✓ Can implement multiple interfaces
- No implementation
- Implementing methods multiple times
- Adding a method to interface will break classes that implement

Abstract Classes

- Contain partial implementation
- Can't extend/subclass multiple classes
- ✓ Add non-abstract methods without breaking subclasses

Sept 22, 2008

Sprenkle - CS209

33

One Option: Use Both!

- Define interface, e.g., **MyInterface**
- Define abstract class, e.g., **AbstractMyInterface**
 - Implements interface
 - Provides implementation for some methods

Sept 22, 2008

Sprenkle - CS209

34

Abstract Classes and Interfaces

- Important structures in Java
- Will return to/apply these ideas throughout the course

Sept 22, 2008

Sprenkle - CS209

35

Due Friday: Assignment 6

- Abstract classes practice
 - Make **GameObject** an **abstract** class
 - Define move as an abstract method
- Packages
 - Organize **MediaItem** classes into a package
- Interfaces practice
 - **MediaItem** and subclasses implement **Comparable** interface

Sept 22, 2008

Sprenkle - CS209

36