

Objectives

- Packages
- Wrapper Classes
- Inheritance
 - Final methods, fields
 - Abstract Classes
 - Interfaces

Sept 26, 2011

Sprenkle - CSCI209

1

Review

- How can we verify that an object variable is a certain type?
- How can we specify an object variable has a different type (a derived type)?
- How does Java decide what method to call on an object?
 - Example: `chicken[1] = foghorn;`
- What is the syntax for Javadoc comments?
- How has developing in Eclipse been going?
- Why can Eclipse provide a lot of functionality that Python can't?
 - Example: renaming a class's method everywhere it is used

Sept 26, 2011

Sprenkle - CSCI209

2

Code Review

- Compare and contrast the following code snippets:

```
for (int i = 1; i <= string.length(); i++){
    strB.append( string.charAt(string.length() - i) );
}
```

```
for( int i=string.length()-1; i >=0 ; i-- ) {
    strB.append(string.charAt(i));
}
```

Sept 26, 2011

Sprenkle - CSCI209

3

Eclipse Hints

- After you have written a method, type

```
/**
```

before the method, and then hit enter and the Javadocs template will be automatically generated for you

- Use command-spacebar for possible completions

Sept 26, 2011

Sprenkle - CSCI209

4

PACKAGES

Sept 26, 2011

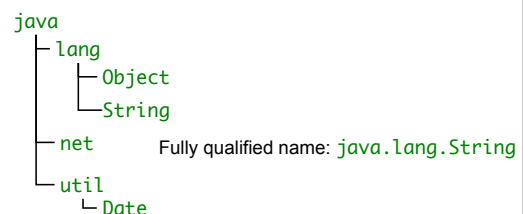
Sprenkle - CSCI209

5

Packages

- Hierarchical structure of Java classes

- Directories of directories



- Use `import` to access packages

Sept 26, 2011

Sprenkle - CSCI209

6

Standard Practice

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

Sept 26, 2011

Sprenkle - CSCI209

7

Importing Packages

- Can import one class at a time or all the classes within a package
- Examples:

```
import java.util.Date;
import java.io.*; ← Import entire package
```

- *** form may increase compile time**
 - BUT, no effect on run-time performance

Sept 26, 2011

Sprenkle - CSCI209

8

WRAPPER CLASSES

Sept 26, 2011

Sprenkle - CSCI209

9

Wrapper Classes

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

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

Sept 26, 2011

Sprenkle - CSCI209

10

Wrapper Classes

- Autoboxing** – automatically create a wrapper object
- Autounboxing** – automatically extract a primitive type

```
// implicitly 11 converted to
// new Integer(11);
Integer y = 11;
```

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

Convert right side for whatever is needed on the left

Sept 26, 2011

Sprenkle - CSCI209

11

Effective Java: Unnecessary Autoboxing

- Can you find the inefficiency from object creation?

```
Long sum = 0L;
for (long i=0; i < Integer.MAX_VALUE; i++) {
    sum += i;
}
System.out.println(sum);
```

- How to fix?

Sept 26, 2011

Sprenkle - CSCI209

Autobox.java

12

Effective Java: Unnecessary Autoboxing

- Can you find the inefficiency from object creation?

```
Long sum = 0L;
for (long i=0; i < Integer.MAX_VALUE; i++) {
    sum += i;
}
System.out.println(sum);
```

Constructs 2^{31} Long instances

- How to fix?

Sept 26, 2011

Sprenkle - CSCI209

Autobox.java

13

Effective Java: Unnecessary Autoboxing

- Can you find the inefficiency from object creation?

```
Long sum = 0L;
for (long i=0; i < Integer.MAX_VALUE; i++) {
    sum += i;
}
System.out.println(sum);
```

Constructs 2^{31} Long instances

- How to fix?

- Lessons:
 - Prefer primitives to boxed primitives
 - Watch for unintentional autoboxing

Sept 26, 2011

Sprenkle - CSCI209

Autobox.java

14

FINAL KEYWORD

Sept 26, 2011

Sprenkle - CSCI209

15

Preventing Inheritance

- Sometimes, you do not want a class to derive from one of your classes
- A class that cannot be extended is known as a **final** class
- To make a class final, simply add the keyword **final** in front of the class definition:

```
public final class Rooster extends Chicken {
    . . .
}
```

- Example of **final** class: **System**

Sept 26, 2011

Sprenkle - CSCI209

16

Final methods

- Can make a method **final**
 - Any class derived from this class cannot override the **final** methods

```
class Chicken {
    . . .
    public final String getName() { . . . }
    . . .
}
```

- By default, **all** methods in a **final** class are **final** methods.

Why would we want to use **final**?
What are possible benefits to us, the compiler?

Sept 26, 2011

Why final methods and classes?

- Efficiency**
 - Compiler can replace a **final** method call with an inline method
 - Does not have to worry about another form of this method that belongs to a derived class
 - JVM does not need to determine which method to call dynamically
- Safety**
 - No alternate form of the method; straightforward which version of the method you called

Sept 26, 2011

Sprenkle - CSCI209

18

ABSTRACT CLASSES

Sept 26, 2011

Sprenkle - CSCI209

19

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 26, 2011

Sprenkle - CSCI209

20

Abstract Classes

- An abstract class **cannot** be instantiated
 - i.e., can't create an object of that class
 - But can have a constructor!
- Child class of an abstract class can only be instantiated if it overrides and implements **every abstract method** of parent class
 - If child class does not override all abstract methods, it is **also abstract**

Sept 26, 2011

Sprenkle - CSCI209

21

Abstract Classes

- *static*, *private*, and *final* methods cannot be *abstract*
 - B/c cannot be overridden by a child class
- *final* class cannot contain abstract methods Why?
- 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 class(es) that complete the implementation
- Can have array of objects of abstract class
 - Does dynamic dispatch for methods

Sept 26, 2011

Sprenkle - CSCI209

22

Examples of abstract classes

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

Sept 26, 2011

Sprenkle - CSCI209

23

Summary: Defining Abstract Classes

- Define a class as *abstract* when have *partial implementation*

Sept 26, 2011

Sprenkle - CSCI209

24

Better Organization of Game Classes

- GamePiece should be abstract
 - No default image associated with it
 - move method is abstract

Sept 26, 2011

Sprenkle - CSCI209

25

INTERFACES

Sept 26, 2011

Sprenkle - CSCI209

26

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 26, 2011

Sprenkle - CSCI209

27

Example of an Interface

- 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` must provide a way to decide if one object is less than, greater than, or equal to another object

Sound similar to anything in Python?

Sept 26, 2011

Sprenkle - CSCI209

28

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 26, 2011

Sprenkle - CSCI209

29

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 26, 2011

Sprenkle - CSCI209

30

## How to determine Chicken order?

- What if made the Chicken class Comparable?

Sept 26, 2011

Sprenkle - CSCI209

31

## 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?

Update  
Chicken.java 32

Sept 26, 2011

Sprenkle - CSCI209

## Comparable Interface API/Javadoc

- Specifies 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 26, 2011

Sprenkle - CSCI209

33

## Interface Summary

- Contain only object (*not class*) methods
- All methods are **public**
  - Implied if not explicit
- Fields are constants that are **static** and **final**
- A class can implement multiple interfaces
  - Separated by commas in definition

Sept 26, 2011

Sprenkle - CSCI209

34

## 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 26, 2011

Sprenkle - CSCI209

35

## 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 26, 2011

Sprenkle - CSCI209

36

## Interface Definitions

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

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

Sept 26, 2011

Sprenkle - CSCI209

37

## Interface Definitions and Inheritance

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

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

Sept 26, 2011

Sprenkle - CSCI209

38

## Interface Definitions and Inheritance

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

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

Sept 26, 2011

Sprenkle - CSCI209

39

## 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 26, 2011

Sprenkle - CSCI209

40

## 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 26, 2011

Sprenkle - CSCI209

41

## Multiple Interfaces

- A class can implement multiple interfaces
  - Must fulfill the requirements of each interface

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

- But NOT possible with inheritance
  - A class can only extend (or inherit from) **one** class

Sept 26, 2011

Sprenkle - CSCI209

42

## Common Uses of Interfaces

- Define constants for multiple classes/  
package
  - Something like global constants
  - However, not great design practice
- Marker Interface
  - Interface that is empty
  - Use to identify an object that has a certain property
    - E.g., Cloneable

Sept 26, 2011

Sprenkle - CSCI209

43

## 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 26, 2011

Sprenkle - CSCI209

44

## One Option: Use Both!

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

Sept 26, 2011

Sprenkle - CSCI209

45

## Abstract Classes and Interfaces

- Important structures in Java
  - Make code easier to change
- Will return to/apply these ideas throughout the course

Sept 26, 2011

Sprenkle - CSCI209

46

## TODO

- Assignment 5: due Wednesday
- Assignment 6: due Friday
  - 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 26, 2011

Sprenkle - CSCI209

47