Lecture 2
Quick Review of Java Concepts

Paul Chew
CS 211 – Fall 2005

Announcements
- Assignment 1 is online and is due Friday at 5pm
- Next topic: Induction

Overview
- Goals
  - Review some of the key constructs of Java (especially those useful for Assignment 1)
  - Discuss some of the new features of J2SE 5.0
- Contents
  - Classes & Objects (brief review)
  - Reference Types vs. Primitive Types
  - Arrays (brief review)
  - Array vs. ArrayList vs. HashMap
  - Command Line Interface
  - New features of J2SE 5.0 (generics, enhanced for, enum)
  - Debugging and experimentation

Classes & Objects (brief review)
- A program usually consists of several classes: class ClassName {
  field declarations;
  method(parameters) {
    local variable declarations;
    body;
  }
}
- Each class has some fields and some methods
- A typical class declaration:

The Main Method
- To run as an application, a class must have a main method, which is always declared as:
  public static void main(String[] args) {
    ... body ...
  }
- There are other specific methods needed for a class to run as, for instance, an applet or a thread

Instance Fields vs. Static Fields
- Instance fields:
  - Belong to individual instances of a class
  - Each instance of the class has a different version of the field
- Static fields:
  - Belong to the entire class
  - A single static field is shared by all instances of the class
  - Are declared using keyword "static"
Static vs. Instance Example

```java
class Widget {
    static int nextSerialNumber = 10000;
    int serialNumber;

    Widget() { serialNumber = nextSerialNumber++;
    }
    Widget(int sn) { serialNumber = sn; }

    public static void main(String[] args) {
        Widget a = new Widget();
        Widget b = new Widget();
        Widget c = new Widget();
        Widget d = new Widget(42);
        System.out.println(a.serialNumber);
        System.out.println(b.serialNumber);
        System.out.println(c.serialNumber);
        System.out.println(d.serialNumber);
    }
}
```

Instance Methods vs. Static Methods

- Instance methods are called
  - Via `instance.method(args)`
  - Or, when called from within an instance method via `method(args)`

- Static methods
  - Cannot refer to instance fields or instance methods (except via qualified name)
  - Can be called even if no objects of the class have been created

- Static methods are called
  - Via `Classname.method(args)`
  - Or, when called from within any method in same class via `method(args)`

Parameters and Local Variables

- Parameters and local variables exist only while the method is running
  - Use parameters to pass input data to a method
  - Use local variables for temporary data used in a method

- Use fields for persistent data or data shared by several methods

Names

- Reference fields, methods in own class by unqualified name
  - serialNumber
  - nextSerialNumber

- Reference static fields in another class by qualified name
  - Widget.nextSerialNumber

- Reference instance fields with qualified name
  - a.serialNumber

- Example
  - `System.out.println(a.serialNumber)`
  - The value of `out` is an instance of a class that has a method `println(int)`
  - If an object has to refer to itself, use `this`

Overloading of Methods

- A single class can have several methods of the same name as long as they have different signatures

- Look at `String.valueOf(…)` in Java API
  - There are 9 of them, one for each of 9 different argument types
  - Think of argument types as part of the name of the method

The Class Hierarchy

- The class hierarchy is a tree
  - Object is at the root (top)
  - E.g., String and StringBuilder are subclasses of Object

- The hierarchy is a tree because
  - Each class has at most one superclass
  - Each class can have zero or more subclasses

- Within a class, methods and fields of its superclass are available
  - But must use `super` for access to overridden methods
**Primitive Types vs. Reference Types**

- **Primitive types**
  - int, long, float, byte, char, boolean, ...
  - Take a single word or 2 words of storage
  - Are not objects in the Java sense
  - A variable of that type contains the actual data

- **Reference types**
  - Arrays and objects (e.g., String, StringBuffer, HashSet)
  - Usually take more storage
  - A variable of that type contains a pointer to the actual data
  - null is a reference type

**== vs. equals( )**

- Use == for primitive types
- For reference types (e.g., Strings), use == only if you mean actual identity of the two objects
  - This is ALMOST NEVER what you want!
- E.g., can have two different strings, both with value "hello"
  - Use x.equals("hello")
  - Not x == "hello".

**Arrays**

- Arrays are reference types
- Array elements can be reference types or primitive types
  - E.g., int[] or String[]
- If a is an array, a.length is its length
- Its elements are a[0], a[1], ..., a[a.length - 1]

- Multidimensional arrays are really arrays of arrays
  - E.g., int[][] is an array of integer arrays
  - Multidimensional arrays can be ragged (i.e., all the arrays in the 2nd dimension need not be the same length)

**HashMap Example**

- Create a HashMap of numbers, using the names of the numbers as keys:
  
  ```java
  HashMap numbers = new HashMap();
  numbers.put("one", new Integer(1));
  numbers.put("two", new Integer(2));
  numbers.put("three", new Integer(3));
  
  To retrieve a number:
  ```
  ```java
  Integer n = (Integer)numbers.get("two");
  if (n != null) System.out.println("two = " + n);
  ```
  ```java
  ```
- Caveat: returns null if does not contain the key
- Can use numbers.containsKey(key) to check this

**Ragged Multidimensional Array**

```java
class MultiArray {
    static int[][][] a = new int[2][3][];
    public static void main(String[] args) {
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                a[i][j] = new int[i+j];
            }
        }
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                System.out.println(a[i][j].length);
            }
        }
    }
}
```
Command Line Interface

- Command line arguments are contained in the String array parameter of the main method

```java
class CmdLineArgs {
    public static void main(String[] args) {
        System.out.println(args.length);
        for (int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```

- > java Foo
  0
- > java Foo asdf zxcv ss
  3
  asdf
  zxcv
  ss
- > java Foo hello world
  2
  hello
  world

Try your programs in a command window, not just in DrJava
- That's how we'll be testing them
- Behavior may be a little different

New Feature of J2SE 5.0: Generics

- Old
  ```java
  HashMap h = new HashMap();
  h.put("one", new Integer(1));
  Integer s = (Integer)h.get("one");
  ```

- New
  ```java
  HashMap<String, Integer> h = new HashMap<String, Integer>();
  h.put("one", new Integer(1));
  Integer s = h.get("one");
  ```

- No longer necessary to do a class cast each time you take an element out of the HashMap

New Feature of J2SE 5.0: Enhanced for

- Old
  ```java
  String[] a = {"hello", "world"};
  for (int i = 0; i < a.length; i++) {
      System.out.println(a[i]);
  }
  ```

- New
  ```java
  String[] a = {"hello", "world"};
  for (String s : a) {
      System.out.println(s);
  }
  ```

- Also works generics (using HashMap h from previous example):
  ```java
  for (String key : h.keySet()) {
      System.out.println(key);
  }
  ```

New Feature of J2SE 5.0: Enum

- Old
  ```java
  class Enum1 {
      public static final int WINTER = 0;
      public static final int SPRING = 1;
      public static final int SUMMER = 2;
      public static final int FALL = 3;
      public static void main(String[] args) {
          System.out.println(WINTER);
      }
  }
  ```

- New
  ```java
  enum Season { WINTER, SPRING, SUMMER, FALL }
  class Enum2 {
      public static void main(String[] args) {
          System.out.println(Season.WINTER);
      }
  }
  ```

- The first program prints 0; the second prints WINTER

Experimentation and Debugging

- Don't be afraid to experiment if you don't know how things work
  - An IDE (e.g., DrJava) makes this easy by providing an Interactions Panel

- Debugging
  - Isolate the bug using, for example, print statements combined with binary search
  - An IDE makes this easy by providing a Debugging Mode
    - Can step through the program while watching chosen variables