Java Review

Lecture 2
CS 2110 Fall 2011

Announcements

- Assignment 1 has been posted
  - due Wednesday, September 7, 11:59pm
  - materials available in CMS
- Check that you are in CMS
  - report any problems to your Section TA (email is fine)
- It's really a good idea to start on A1 and check CMS this week (well before the assignment is due)
- Available help
  - consulting starts today
  - instructor & TA office hours are in effect
- Check daily for announcements
  http://courses.cs.cornell.edu/cs2110

More Announcements

- Sections start this week
  - section material will be useful for A1
- Activate your Piazza account
  aoc76  dash8  jx73  thr44
  aj232  edb56  mdr257  vs286
  ac23  efs277  mfe73  wwe95
  bpa26  hmx7  nbl68  xll67
  bps239  jhn47  pbl76  xmr32
  bzb86  jxh66  pbl78  zed2
  cac366  jx2224  pkn6  zy56
  clv38  jbl348  qj336

Today — A Smorgasbord

- A brief (biased) history of programming languages
- Review of some Java/OOP concepts
- Java tips, trick, and pitfalls
- Debugging and experimentation

Machine Language

- Used with the earliest electronic computers (1940s)
  - machines use vacuum tubes instead of transistors
- Programs are entered by setting switches or reading punch cards
- All instructions are numbers
- Example code
  0110 0001 0000 0110
  Add Reg1  6
- An idea for improvement
  - use words instead of numbers
  - result: assembly language

Assembly Language

- Idea: Use a program (an assembler) to convert assembly language into machine code
- Early assemblers were some of the most complicated code of the time (1950s)
- Example code
  ADD R1 6
  MOV R1  COST
  SET R1 0
  JMP TOP
- Idea for improvement
  - let's make it easier for humans by designing a high-level computer language
  - result: high-level languages
High-Level Language

- Idea: Use a program (a compiler or an interpreter) to convert high-level code into machine code
- The concept was initially controversial
  - FORTRAN (mathematical FORmula TRANslating system) was designed with efficiency vary much in mind
- Pro
  - Easier for humans to write, read, and maintain code
- Con
  - The resulting program will never be as efficient as good assembly-code
    - Waste of memory
    - Waste of time

FORTRAN

- Initial version developed in 1957 by IBM
- Example code
  ```c
  C   SUM OF SQUARES
  ISUM = 0
  DO 100 I=1,10
  ISUM = ISUM + I*I
  100 CONTINUE
  ```
- FORTRAN introduced many high-level language constructs still in use today
  - Variables & assignment
  - Loops
  - Conditionals
  - Subroutines
  - Comments

ALGOL

- Sample code
  ```
  comment Sum of squares
  begin
  integer i, sum;
  for i :=1 until 10 do
  sum := sum + i*i;
  end
  ```
- ALGOL = ALGOrithmic Language
- Developed by an international committee
- First version in 1958 (not widely used)
- Second version in 1960 (widely used)
- ALGOL 60 included recursion
  - Pro: easier to design clean, succinct algorithms
  - Con: difficult to implement; too inefficient

COBOL

- COBOL = COmmon Business Oriented Language
- Developed by the US government (about 1960)
- Design was greatly influenced by Grace Hopper
- Goal: Programs should look like English
- Idea was that anyone should be able to read and understand a COBOL program
- COBOL included the idea of records (a single data structure with multiple fields, each field holding a value)

Simula & Smalltalk

- These languages introduced and popularized Object Oriented Programming (OOP)
  - Simula was developed in Norway as a language for simulation in the 60s
  - Smalltalk was developed at Xerox PARC in the 70s
- These languages included
  - classes
  - objects
  - subclasses & inheritance

Java – 1995

- Java includes
  - assignment statements, loops, conditionals from FORTRAN (but syntax from C)
  - recursion from ALGOL
  - fields from COBOL
  - OOP from Simula & Smalltalk
- By many counts, Java is the most widely used language in the world today

Java

Java™ and Java® are trademarks of Oracle and/or its affiliates.
We will assume you already know something about ...

• Classes and objects
• Static vs instance fields and methods
• Primitive vs reference types
• Private vs public vs package
• Constructors
• Method signatures
• Local variables
• Arrays
• Subtypes and inheritance, shadowing

Constructors

• Called to create new instances of a class
• Default constructor initializes all fields to default values (0 or null)

```
class Thing {
    int val;
    Thing(int val) {
        this.val = val;
    }
    Thing() {
        this(3);
    }
}
```

```
Thing one = new Thing(1);
Thing two = new Thing(2);
Thing three = new Thing();
```

Static Initializers

• Run once when class is loaded
• Used to initialize static objects

```
class StaticInit {
    static Set<String> courses = new HashSet<String>();
    static {
        courses.add("CS 2110");
        courses.add("CS 2111");
    }
    public static void main(String[] args) {
        ...
    }
}
```

Static vs Instance Example

```
class Widget {
    static int nextSerialNumber = 10000;
    int serialNumber;
    Widget() {
        serialNumber = nextSerialNumber++;
    }
    public static void main(String[] args) {
        Widget a = new Widget();
        Widget b = new Widget();
        Widget c = new Widget();
        System.out.println(a.serialNumber);
        System.out.println(b.serialNumber);
        System.out.println(c.serialNumber);
    }
}
```

A Common Pitfall

local variable shadows field

```
class Thing {
    int val;
    boolean newVal(int v) {
        this.val = v;
    }
}
```

• you would like to set the instance field val = v
• but you have declared a new local variable val
• assignment has no effect on the field val

The main Method

Can be called from anywhere

A class method don’t need an object to call it

No return value

Method must be named main

```
public static void main(String[] args) {
    ...
}
```

Parameters passed to program on command line
Names

- **this** - used by an object to refer to itself
- Refer to static and instance fields & methods of **this** by (unqualified) name:
  - `serialNumber`
  - `nextSerialNumber`
- Refer to static fields & methods in another class using name of the class
  - `Widget.nextSerialNumber`
- Refer to instance fields & methods of another object using name of the object
  - `a.serialNumber`

Example

```
System.out.println(a.serialNumber)
```

**Overview of Methods**

- A class can have several methods of the same name
  - But all methods must have different signatures
  - The signature of a method is its name plus types of its parameters

**Example:** `String.valueOf(…)` in Java API

- There are 9 of them:
  - `valueOf(boolean);`
  - `valueOf(int);`
  - `valueOf(long);`
  - …
- Parameter types are part of the method's signature

**Primitive vs Reference Types**

- **Primitive types**
  - `int`, `short`, `long`, `float`, `byte`, `char`, `boolean`, `double`
  - Efficient
  - 1 or 2 words
  - Not an `Object`—unboxed
- **Reference types**
  - Objects and arrays
  - `String`, `int[]`, `HashSet`
  - Usually require more memory
  - Can have special value `null`
  - Can compare `null` using `==`, `!=`
- Generates `NullPointerException` if you try to dereference `null`

**== vs equals()**

- `==` tests whether variables hold identical values (shallow equality)
- `=` tests whether variables hold identical values (deep equality)
- `==` works fine for primitive types
- `=` works fine for primitive types
- For reference types (e.g., `String`), you usually want to use `equals()` (deep equality)

```
x = "hello";
y = "hello";
x == y?  // true
```

```
x = new String("hello");
y = new String("hello");
x == y?  // false
```

To compare object contents, override `Object.equals()`

```
boolean equals(Object x): false
```

But if you do this, must also override `Object.hashCode()` (more on this later)
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].`
- The length is fixed.

```
String[] a = new String[4];
a.length = 4
```

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].`
- The length is fixed.

```
String[] a = new String[4];
a[2] = "hello";
a.length = 4
```

Accessing Array Elements Sequentially

```
public class CommandLineArgs {
  public static void main(String[] args) {
    System.out.println(args.length);
    // old-style
    for (int i = 0; i < args.length; i++) {
      System.out.println(args[i]);
    }
    // new style
    for (String s : args) {
      System.out.println(s);
    }
  }
}
```

Class Hierarchy

- Every class (except `Object`) has a unique immediate superclass, called its parent.

Overriding

- A method in a subclass overrides a method in superclass if:
  * both methods have the same name
  * both methods have the same signature (number and type of parameters and return type)
  * both are static methods or both are instance methods.
- Methods are dispatched according to the runtime type of the object.

Accessing Overridden Methods

- Suppose a class `S` overrides a method `m` in its parent.
- Methods in `S` can invoke the overridden method in the parent as `super.m()`.
- In particular, can invoke the overridden method in the overriding method.
- Caveat: cannot compose super more than once as in `super.super.m()`.
### Array vs ArrayList vs HashMap

- **Array (in java.util)**
  - storage is allocated when array created; cannot change
- **ArrayList (in java.util)**
  - an "extensible" array
  - can append or insert elements, access i-th element, reset to 0 length
  - use with List interface
- **HashMap (in java.util)**
  - save data indexed by keys
  - can lookup data by key
  - can iterate over keys or values
  - use with Map interface

### Unexpected Consequence

An overriding method cannot have more restricted access than the method it overrides

```java
class A {
    public int m() {...}
}

class B extends A {
    private int m() {...} //illegal!
}

A supR = new B(); //upcasting
supR.m(); //would invoke private method in class B at runtime!
```

### Shadowing

- Like overriding, but for fields instead of methods
  - Superclass: variable `v` of some type
  - Subclass: variable `v` perhaps of some other type
  - Method in subclass can access shadowed variable using `super.v`

- Variable references are resolved using **static binding** (i.e., at compile-time), not **dynamic binding** (i.e., not at runtime)
  - Variable reference `r.v` uses the static type (declared type) of the variable `r`, not the runtime type of the object referred to by `r`

- Shadowing variables is bad medicine and should be avoided

### HashMap Example

- Create a `HashMap` of numbers, using the names of the numbers as keys:

  ```java
  Map<String, Integer> 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 = numbers.get("two");
  ```

  - returns null if the `HashMap` does not contain the key
  - Can use `numbers.containsKey(key)` to check this

### Generics and Autoboxing

- **Pre-Java 5**

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

- **Java 5 (with generics)**

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

- **Java 5 (with generics + autoboxing)**

  ```java
  Map<String, Integer> numbers = new HashMap<String, Integer>();
  numbers.put("one", 1);
  int s = numbers.get("one");
  ```

### Experimentation and Debugging

- Don’t be afraid to experiment if you are not sure how things work
  - Documentation isn’t always clear
  - Interactive Development Environments (IDEs), e.g. Eclipse, make this easier

- **Debugging**
  - Do not just make random changes, hoping something will work
  - Think about what could cause the observed behavior
  - Isolate the bug – use print statements
  - An IDE makes this easier by providing a Debugging Mode
    - Can set breakpoints, step through the program while watching chosen variables