Language Basics

Introduction to Java

Alan Hohn
Alan.M.Hohn@lmco.com

29 August 2013

Contents

Course Contents

- Getting started writing Java programs (complete)
- Java programming language basics (today is #2 of 4 sessions)
- Packaging Java programs (1 session)
- Core library features (6 sessions)
- Java user interfaces (2 sessions)

Our Basic Java Example

```
package org.anvard.introtojava;

// Classes in other packages that we need
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class HelloName {

  public static void main(String[] args)
      throws Exception {
    System.out.print("What is your name? ");
    // 1 statement, 3 objects
    String name = new BufferedReader(
        new InputStreamReader(System.in)).readLine();
    if (name.length() > 10) {
      System.out.println("You have a long name.");
    }
    // Smart concatenation, but no operator overloading
    System.out.println("Hello, " + name);
  }
}
```

This Time

- Method Calls and Field References
- Control Flow
- Operators
- Object Equality

The `this` keyword

- Java methods and fields exist inside a class
  - Unless static, methods and fields belong to a particular instance of the class (to an object)
  - Java uses `this` to refer back to the current object instance
- In Java, all references are "scoped"

Scoped references

- Dot notation is used in Java to "enter" a level of scope
- The compiler searches outward for an unqualified reference
- The scope ends with the class (there are no global fields or methods)
- In other words, it is OK to use the keyword `this`, but not necessary unless avoiding ambiguity

```
public void setValue(int value) {
  this.value = value;
}
```

- Field references and method calls can be "chained"

```
// out is a (static) field in the System class
// println is a method in the PrintStream class
System.out.println("Hello, world!");
```

Method Definitions

- Java methods have four parts
  - Return type (may be void)
  - Identifier (name)
  - Parameter list
  - Exceptions
- No two methods in the same class may have the same name and parameter list
- Methods may also have qualifiers such as `public` or `static`

```
public int add(Integer left, Integer right)
  throws OverflowException;
```

Method Calls

- The instance on which the method operates is like an implicit first parameter
  - It is available to the method using `this`
  - It is used implicitly when the method refers to instance variables
- Objects returned from methods may be "chained"
- Method calls must use parentheses even if there are no parameters, to avoid ambiguity with field references
- Instantiating an object with `new` is like calling a constructor method; it returns an object of the instantiated type

```
System.out.println("Hello, world!");
String name = new BufferedReader(
  new InputStreamReader(System.in)).readLine();
```

Quick Note on Naming

- The compiler searches for an unqualified reference in the current scope
- The `import` statement brings a class into the current scope so it can be used unqualified (without its full package name)
- By convention, Java classes are UpperCamelCase, while packages, fields, methods and variables are lowerCamelCase
- This is done to avoid hiding names unnecessarily

```
// import System; is implied in any Java program
public String system() { return "Laptop"; }
public String system = "Desktop";
System.out.println("Hello, " + system()); // Hello, Laptop
System.out.println("Hello, " + system); // Hello, Desktop
System.out.println("Hello, " + System); // Compiler error
```

Java Control Flow

- Java has the expected set of flow control keywords
  - `if - else if - else`
  - `switch - case - default`
  - `while` and `do - while`
  - `for` (including an enhanced version for iterators)
- `if` conditionals and `for` expressions must be in parentheses
- Curly braces ({ }) are used to group multiple statements; they are optional for a single-statement block but are recommended
- `break` works with `switch`, `for`, `while`, or `do-while`
- `continue` works with `for`, `while`, or `do-while`
- Labels are supported for `break` and `continue`
- `return` can appear anywhere in a method
- There is no `goto`

Control Flow Example

```
class BreakWithLabelDemo {
  public boolean search2dArray(int[][] arrayOfInts,
    int searchfor) {
    boolean foundIt = false;
    search:
    for (int i = 0; i < arrayOfInts.length; i++) {
      for (int j = 0; j < arrayOfInts[i].length; j++) {
        if (arrayOfInts[i][j] == searchfor) {
          foundIt = true;
          break search; // Or just 'return true;'
        }
      }
    }
    return foundIt;
  }
}
```

Enhanced `for`

- Some Java classes implement an interface called `Iterable`
- This includes arrays and most of the built-in collection classes
- This means they will provide an "iterator" for looping through multiple objects
- Java provides a cleaner version of `for` for `Iterable` classes
- This version can also be more performant for some collections

```
class EnhancedForDemo {
  public static void main(String[] args){
    int[] numbers =
      {1,2,3,4,5,6,7,8,9,10};
    for (int item : numbers) {
      System.out.println("Item is: " + item);
    }
  }
}
```

Java Operators

- Java has the expected set of operators (`+ - * / %`)
- Similar to C, there is a difference between bitwise operators (`& | ^`) and comparison operators (`&& ||`)
- Unlike C, Java has no "unsigned" values, but in addition to the shift operators (`<< >>`) there is an "unsigned" shift right `>>>`.
- Java has clever compound operators like C (e.g. `++ -- +=`) and the ternary operator `(? :)`
- The comparison operators (`< <= > >=`) only work for primitive types and their wrapper classes
- There is no exponentiation operator; use `Math.pow()`

Widening and Narrowing Conversions

- Java is a strongly typed language
- However, operators can mix types under certain circumstances
- Java will automatically perform "widening" conversions (`byte` → `short` → `int` → `long` → `float` → `double`)
- Narrowing conversion can lose information, so Java requires an explicit cast

```
double d = 123.45; float f = (float) d;
```

- When mixing operators, the resulting value will be the 'widest' required
- For example, for the division operator /, the result can be an integer (for integer / integer) or a floating-point value (for floating-point or mixed types)

Equality Operators

- Java has two kinds of equality: reference equality (`o1 == o2`) and object equality (`o1.equals(o2)`)
- Reference equality means the two references refer to exactly the same object in memory
- Object equality means the objects are semantically the same (mean the same thing)
- For classes you create, you can (and often should) define your own rules for object equality by making your own `equals()` method
- Later we will talk about `hashCode()`, an important method that's often seen together with `equals()`

Equality example

```
public class Person {
  public String firstName;
  public String lastName;
  public boolean equals(Object obj) {
    if (null == obj) return false;
    if (obj.getClass() != this.getClass()) return false;
    Person other = (Person)obj;
    return other.firstName == this.firstName &&
        other.lastName == this.lastName;
  }
}

Person p1 = new Person();
p1.firstName = "Joe";
p1.lastName = "Smith";
Person p2 = new Person();
p2.firstName = "Joe";
p2.lastName = "Smith";

p1 == p2; // false
p1.equals(p2) // true
```

Next Time

- Exception Handling
- Checked and Unchecked Exceptions
- Try-With-Resources

Credit in LMPeople

LMPeople Course Code: 071409ILT04