Worldwide Developers Conference

Java for C++ Programmers

Randy Nelson

Senior Curriculum Designer

Java for C++ Programmers

- Audience:
  - Object programmers with experience in C and/or C++
- Objectives:
  - Read simple Java source files
  - Identify elements found in C and C++, how they are the same and how they differ
  - Identify elements unique to Java

Java Is…

- Simple, Fully Specified, Portable
- It's like:
  - C with objects
  - Smalltalk with types
  - C++ without guns and knives

Java Language Elements

- Object-oriented
- Dynamic
- Rooted
- Typed
- Language support for:
  - Security
  - Threads
  - Exceptions
  - Native code

Differences from C

- Objects
- No preprocessor
- No globals
- No pointers

Differences from C++

- Library-centric
- No operator overloading
- No multiple inheritance
- No deallocation

Java Virtual Machine

- Java code is written for an abstract virtual machine
- Java code is compiled into bytecode
  - The bytecode is the machine code for the virtual machine
  - This platform independent bytecode is interpreted at run-time by the Java implementation for each platform

Libraries

- Java is a rooted object-oriented language
  - It defines its own root object — Object — and object hierarchy
  - It relies on certain libraries to be present
- Java libraries are provided in packages
  - java.applet

Java Packages

- Base Java packages include:
  - java.lang — default base classes
  - java.applet — browser apps
  - java.awt — portable GUI
  - java.awt.image — graphics
  - java.io — input/output
  - java.net — networking
  - java.util — utility classes

Namespaces and Packages

- Java uses hierarchical package names that provide unique namespaces
  - By convention, a package name begins with the reverse of the enterprise's Internet domain name
  - Additional fields are determined by site and programmer
  - It ends with the name of the class
  - Fields are separated by periods

Package Statement

- A package statement indicates which package the code in a source file is part of
- A package statement uses the package keyword
  - package COM.apple.qt;
  - The package statement must be the first thing in the file
- If there is no package specified, the code is made part of the default unnamed package

Importing

- An import statement provides shorthand for using package names in source code
- Import statements use the import keyword
  - Importing a package name allows you to reference the names contained in the package without the including the entire package path
- It simply saves typing

Importing the Date Class

Without import

```
java.util.Date currentDate =
   new java.util.Date();
```

With import

```
import java.util.Date
...
Date currentDate = new Date();
```

Java Code

- Java code is contained within a class definition
  - Classes can contain class and instance variables and class and instance methods
  - These are collectively known as the class' members
- There are no global variables or functions

HelloWorld in Java

A simple class definition

```
class HelloWorld {
   public static void main(String[] args)
   {
      System.out.println("Hello World!");
   }
}
```

Java Data Types

- Primitive—non-object values
  - boolean, char, short, int, long, float, double
- Reference—class as type
  - Objects—refer to instances of a class
    - Date, Converter
  - Arrays—refer to ordered collections of the same class or primitive type
    - Date[], int[]

UNICODE

- Characters, strings and language identifiers are made up of 16-bit UNICODE characters
  - The first 256 UNICODE characters are the same as ASCII characters
- UNICODE characters can be represented by the following escape sequence: \unnnn where nnnn is a sequence of four hex digits

Declarations

- Declarations can only appear within a class definition
  - Local variables can be declared anywhere in a method
- Types are class names and primitives, identifiers are UNICODE strings
- Forward reference to undeclared members is allowed
- There is no prototyping

Creating an Instance

- Objects are instantiated using the operator new with a class name
  - new invokes the class' constructor
- Each variable must be explicitly initialized
  - A declaration only creates the variable to hold the reference to the object
  - References initially point at null

Strings

- Strings are represented by the String class: java.lang.String
  - They are not expressed as arrays of char
  - They are immutable: use StringBuffer for mutable strings
- Methods are provided that parallel those found in the C string library

Literals

- String literals are instances created directly from entered data
  - String objects are normally instantiated from a string literal: a value between double quotes
    - "Hello World!"
- Numbers and other primitive types are also entered directly
  - 3.14159

Declarations and Instantiations

Declarations

```
Converter theConverter;
HelloWorld myGreeter;
String aGreeting;
```

Instantiations

```
theConverter = new Converter();
myGreeter = new HelloWorld();
aGreeting = "Hello World!";

Date now = new Date();
```

Classes and Files

- Each class definition is compiled into a class file
  - A file called HelloWorld.java results in HelloWorld.class
- The file name must match the name of the primary class defined in it
  - The file containing the Converter class must be named Converter.java

Class Definition

- A class definition starts with the keyword class followed by the class name
  - The class is assumed to inherit from Object
- Variables and methods are defined in any order
  - Class variables and methods are preceded by the keyword static

Converter Class Definition

```
class Converter {
    static int numberOfConversions;
    String name = "Identity";

    static void incrementConversions() {
        numberOfConversions++;
    }
    double convert(double inputValue) {
        incrementConversions();
        return inputValue;
    }
}
```

Extending a Class

- To subclass, follow the class name with the extends keyword and the superclass name
  - class Subclass extends Superclass
- In an override, to invoke the overridden method, use the keyword super as the object reference
  - super.methodName();

Extending Converter

A converter that doubles its input

```
class Doubler extends Converter {
   double convert(double inputValue) {
      double result;
      result = super.convert(inputValue);
      return result * 2;
   }
}
```

Accessing Variables

- Variables are accessed using an object reference, followed by a period, followed by the variable name
- reference.variableName
  - Instance variable are accessed using instance references
  - Class variables can be accessed using the class name or an instance reference

Invoking Methods

- Methods can be invoked by using an object reference, followed by a period, followed by the method name
- reference.methodName()
  - Instance methods are invoked using instance references
  - Class methods can be invoked using the class name or an instance reference

Using Converter

Accessing a class variable

```
Converter.numberOfConversions;
```

Accessing an instance variable

```
theConverter.name;
```

Invoking a class method

```
Converter.incrementConversions();
```

Invoking an instance method

```
theConverter.convert(100);
```

Referring to the Receiver

- The instance that was invoked can be referred to within the method as this
  - An instances' variables and methods are implicitly referenced via this if they do not have an explicit reference
- The following are equivalent:
  - memberName
  - this.memberName

Method Overloading

- Multiple methods with the same name and different number or type of parameters can be defined
  - The system will choose the appropriate implementation at run-time based on the arguments
  - Return type must remain the same
- Language operators cannot be redefined

Overloading convert

Expects a double

```
double convert(double doubleIn)...
```

Expects an integer

```
double convert(int intIn)...
```

Expects an integer array

```
double convert(int[] intArrayIn)...
```

Expects and integer and a double

```
double convert(int i, double d)...
```

Constructors

- A constructor is special type of method that initializes a new instance
  - A constructor has the same name as the class
    - Classname()...
  - Constructors do not indicate a return type
- A default constructor is created automatically for every class

Multiple Constructors

- It is typical to have several different constructors, each of which takes different arguments
- From within a constructor, use this() to invoke another constructor from the same class, use super() to invoke a superclass' constructor

Using Multiple Constructors

```
class Multi extends Single {
    int size;
    String name;
    Multi(int size) {
        super(size);
        this.size = size;
    }
    Multi(int size, String name) {
        this(size);
        this.name = name;
    }
}
```

Constants

- Constants can be defined by preceding a variable declaration with the keyword final
  - A final variable cannot be changed
  - Constants are typically also declared static
    - final static int A = 1;
    - final static int Z = 26;

Argument Passing

- Arguments to methods are passed by value
- Passing a reference type by value is somewhat like passing a pointer type by value
  - The members of the referenced object can be modified in the receiving method
  - The object itself cannot be replaced

Protection and Access

- Access to objects and members of a class is controlled
  - An object's members can be:
    - public—visible to all other objects
    - Visible within the current package
    - protected—visible to subclasses
    - private—only visible within the object itself

Garbage Collection

- The system takes care of deallocation
  - There is no deallocate or free method
- Objects are automatically garbage collected when there are no more references pointing to them
  - Java does not define the garbage collection technique

Operators

- Operator precedence is similar to that of C or C++
- Java adds some new operators
  - + —string concatenation
  - instanceof—checks if the value is an instance of particular class or interface
- C's *, &, and sizeof operators do not exist

Flow Control

- if statements can only test booleans
- switch statements must use ints
- Labelled break and continue statements allow branching to specific locations in the code
- for statements allow loop variables to be declared within the initialization block

Exception Handling

- Java provides exception handling for errors using try—catch—finally
  - Code to be executed is found in the try block
  - Code to provide remedies for errors in the try block follows in the catch block
  - Code to be executed in any case is in the finally block

Simple Output

- System.out.println provides the ability to write to the standard output
- Use string concatenation to create your output
  - String concatenation will automatically convert many types
    - String pi;
    - pi = "pi = " + 3.14159;
    - System.out.println(pi);

Dynamic Loading and Binding

- Java classes are loaded as they are needed
- Method names are bound to their implementation when they are first called
- Superclasses can change their implementation without forcing their subclasses to be recompiled

Abstract and Final Classes

- Abstract classes are classes that act as the basis for subclassing
  - They cannot be instantiated
  - Precede the class keyword with the abstract keyword
- Final classes are essentially sealed
  - They cannot be subclassed
  - Precede the class keyword with the final keyword

Interfaces

- Interfaces are a way of sharing method declarations across multiple classes
  - An interface only describes the method name, parameters and return type
  - It has no implementation
- An interface definition uses the interface keyword
  - It contains method prototypes

Interfaces

- Interfaces can be extended using the extends keyword
- A class indicates the interfaces it implements with the implements keyword
  - It must implement every method
  - It can implement multiple interfaces
- An interface can be used as a type

Interfaces

```
interface Steerable {
   void turnLeft();
   void turnRight();
   void goStraight();
}
class Vehicle implements Steerable {
   void turnLeft() { ... }
   void turnRight() { ... }
   void goStraight() { ... }
}
Steerable car = new Vehicle();
```

Persistence

- There is no persistence for Java instances
  - Instances are created at run-time from class files
  - To store state you need to use external files or databases

Applications

- Java applications are classes that contain a main method
  - main is defined as follows:
  - public static void
        main(String[] args)
- The array args passed to main contains the command line arguments that the application was invoked with
  - args[0] is the first argument

Applets

- Applets are subclasses of the class java.applet.Applet
- Applets take over an area of a web browser's page
    - Applets respond to events and draw
- Applets also respond to a set of well defined lifecycle messages
    - init, start, paint, stop, destroy

Threads

- Java has language level support for threads
  - Java can manage locks for each instance and method
  - The locks prevents the use of the instance or method in one thread if another already has the lock
- The synchronized keyword indicates the method or object to lock

Security

- Java supports a runtime security manager
  - It can check the bytecodes it is provided to make sure they are valid
  - It provides policies to prevent or allow certain operations based on the source of the code—local or network—to be run

Native Methods

- Java supports calls to externally compiled code
  - A method with the native keyword acts as a stub that gets bound to the actual code
  - This allows existing code, like Macintosh toolbox code, to be invoked using Java
    - See MRJ SDK for examples

References

- Here are some references to use to continue learning Java
  - The Java Tutorial
    - http://java.sun.com/tutorial/
  - The Java Programming Language
    - Arnold, Gosling—Addison-Wesley
  - Java in a Nutshell
    - Flangan—O'Reilly & Associates

Worldwide
Developers
Conference