

Java Primer

Thomas Pelaia II, Ph.D.

XAL Lecture Series

Lesson 1

July 1, 2008



XAL Lecture Series

1. **Java Primer**
2. **Tour of XAL**
3. **Building Java Applications in XAL**
4. **Ruby Primer**
5. **Scripting Applications in XAL**
6. **Channel Access in XAL**
7. **XAL Accelerator Hierarchy**

Java Primer

- **Procedural Programming**
- **Object-Oriented Programming**
- **Java Language Overview**
- **Programming Guides**
- **Tour of Java Packages**
- **Java Resources**

Procedural Programming

- **Foundation for most languages**
- **Natural at the machine level**
 - **Flow of code analogous to low level execution**
 - **Sequential write statements to be executed sequentially**
 - **Logical Flow Control operations**
- **Functions encapsulate reusable code**
- **Global variables allow sharing of named data across functions and libraries**
- **Local variables provide named data storage within a block of code**

Sample Procedural Pseudocode Snippet

```
def distance( x1, y1, x2, y2 )  
    return sqrt( (x1 - x2)^2 + (y1 - y2)^2 )  
end
```

```
electron_x = 1.0  
electron_y = 2.0  
electron_charge = -1.6e-19  
proton_x = 4.0  
proton_y = 5.0  
proton_charge = 1.6e-19  
k = 9e9
```

```
elec_prot_dist = distance( electron_x, electron_y, proton_x, proton_y)  
elec_prot_force = k * electron_charge * proton_charge / elec_prot_dist^2
```

Object-Oriented Programming

- **Structured code**
- **More natural for human processes**
 - Different objects have different roles
 - Objects can be **classified** according to characteristics and behavior
- **Motivated by good programming practices **but not a substitute for them****
- **A Class is a template for objects**
- **Objects encapsulate data and operations to be performed on data**

Sample Object-Oriented Pseudocode Snippet

```
class Point
  float x, y
  def distanceTo( Point point )
    return sqrt( (x - point.x)^2 + (y - point.y)^2 )
  end
end
```

```
class Particle
  static float k = 9e9
  Point location
  float charge
  def forceFrom( Particle particle )
    return k * charge * particle.charge / ( location.distanceTo( particle.point ) )^2
  end
end
```

```
Particle electron = Particle.new( Point.new(1.0, 2.0), -1.6e-19 )
```

```
Particle proton = Particle.new( Point.new(4.0, 5.0), 1.6e-19 )
```

```
elec_prot_force = electron.forceFrom( proton )
```

Object-Oriented versus Procedural Programming

	Procedural	Object-Oriented
Low Initial Overhead	✓	X
Scales well with Growth	X	✓
Provides Classification	X	✓
Data Encapsulation	X	✓

Java Language Overview

- **Java is in its sixth major release**
 - We use Java version 5 due to Linux bugs for version 6
- **Platform independent**
- **Strongly typed (and case sensitive)**
- **Object-Oriented**
- **Dynamic (Somewhat)**
- **Garbage Collected**
- **Big and complex language**
 - Too big to completely cover here

Class

- **Classification of objects**
- **Template for instantiating objects**
 - Defines typed data associated with the class (**Class Variables**)
 - Defines an instantiated object's typed data structure (**Instance Variables**)
 - Defines how to initialize class variables (**Static Initializer**)
 - Defines how to initialize an object (**Constructors**)
 - Defines operations on a class (**Static Methods**)
 - Defines operations on an object (**Methods**)

Instance Members

Note that instance methods and instance variables are collectively referred to as instance members.

Variable Types

- **Local**
 - data local to blocks and methods
 - persists only during block/method execution
- **Instance**
 - data available to an object
 - persists along with an object
- **Class**
 - data available to a class independent of instance
 - persists for duration of program execution

Variables Example

```
/** Defines a Cartesian Point */  
public class Point {  
    static final public double METERS_TO_MM = 1000.0;  
    public double x;  
    public double y;  
}
```

Constructors

- **Construct an object (instance of a class)**
 - Performs initialization of an object's instance variables
 - Sometimes performs other operations during object initialization
- **Multiple constructors for the same class identified by argument list**
- **Good practice to define a primary constructor which all other constructors call**

Constructor Example

```
/** Defines a Cartesian Point */
public class Point {
    static final public double METERS_TO_MM = 1000.0;
    final public double X;
    final public double Y;

    /** Primary Constructor */
    public Point( final double x, final double y ) {
        this.X = x;
        this.Y = y;
    }

    /** Empty Constructor */
    public Point() {
        this( 0.0, 0.0 );
    }
}
```

Method Types

- **Instance Method**
 - Performs operations on an object's data
 - Can call static methods
 - Can access static variables
- **Static Method**
 - Performs operations on a class's static data
 - Sometimes called "Class Method"
 - Instance members aren't available to it
 - Oddly, no access to the class itself

Method Example

```
/** Defines a Cartesian Point */
public class Point {
    static final private double METERS_TO_MM = 1000.0;
    final protected double X;
    final protected double Y;

    /** Primary Constructor */
    public Point( final double x, final double y ) {
        this.X = x;
        this.Y = y;
    }

    /** Factory method to instantiate a point using polar coordinates */
    static public Point getPolarPoint( final double r, final double theta ) {
        return new Point( r * Math.cos( theta ), r * Math.sin( theta ) );
    }

    /** Accessor to get the horizontal coordinate */
    public double getX() {
        return X;
    }

    /** Accessor to get the vertical coordinate */
    public double getY() {
        return Y;
    }

    /** Calculate the distance from the origin */
    public double distanceFromOrigin() {
        return Math.sqrt( X * X + Y * Y );
    }

    /** Compute the distance to another point */
    public double distanceTo( final Point point ) {
        return Math.sqrt( (X - point.X) * (X - point.X) + (Y - point.Y) * (Y - point.Y) );
    }
}
```

Member Access Control

	Keyword	Member Access From
Public	public	Any object
Private	private	Objects of the same class
Protected	protected	Objects of the same class, its subclasses and objects of classes within the same package
Package		Objects of classes within the same package

Class Inheritance

- **Java supports single class inheritance**
- **Instance members may be added**
- **Instance methods may be overridden**
- **Only the empty constructor is inherited**
- **All of a class's instance members are inherited but public, protected and sometimes package members are accessible to a subclass**
- **No static members are inherited**
- **The “**super**” keyword refers to the superclass's version of a member**

Inheritance Example

```
/** Defines a 3D Cartesian Point */
public class Point3D extends Point {
    final protected double Z;

    /** Primary Constructor */
    public Point3D( final double x, final double y, final double z ) {
        super( x, y );
        Z = z;
    }

    /** Accessor to get the longitudinal coordinate */
    public double getZ() {
        return Z;
    }

    /** Calculate the distance from the origin */
    public double distanceFromOrigin() {
        return Math.sqrt( X * X + Y * Y + Z * Z );
    }

    /** Compute the distance to another 3D point */
    public double distanceTo( final Point3D point ) {
        return Math.sqrt( (X - point.X) * (X - point.X) + (Y - point.Y) * (Y - point.Y) + (Z - point.Z) * (Z - point.Z) );
    }
}
```

Interfaces

- **Declares a set of methods that implementing classes define**
- **Allows for abstraction across otherwise unrelated classes**
- **Classes may implement multiple interfaces**

Interface Example

```
/** Defines methods common to an object capable of greeting someone */  
public interface Greeter {  
    /** Return a greeting to the specified named individual */  
    public String greet( final String name );  
}
```

```
/** Defines a Person */  
public class Person implements Greeter {  
    final String NAME;
```

```
    /** Primary Constructor */  
    public Person( final String name ) {  
        NAME = name;  
    }
```

```
    /** Return an greeting to the specified named individual */  
    public String greet( final String name ) {  
        return "Hello, " + name + "! My name is " + NAME + ".";  
    }  
}
```

```
/** Defines a Computer */  
public class Computer implements Greeter {  
    /** Return an greeting to the specified named individual */  
    public String greet( final String name ) {  
        return "Hello, " + name + "! I am your computer and ready to perform tasks for you.";  
    }  
}
```

Commonly Used Control Statements

- **if / else conditional execution**
- **continue / break flow control**
- **switch / case option execution**
- **for loop execution**
- **while loop execution**
- **try / catch / finally exception handling**

Programming Guides

- **Design for change**
- **Keep it simple**
- **Code should be self documenting**
 - **Use consistent expressive names for variables, methods, classes, packages, etc.**
- **Code and test incrementally**
- **Minimize dependencies**
 - **Use strictest access control**
 - instance variables typically protected or private
 - instance methods typically public
- **Optimize code through good practices**

Design Patterns (Few of many)

- **Delegate Pattern**
 - An object is delegated to perform certain operations on behalf of another object
- **Adaptor Pattern**
 - An adaptor binds otherwise separate pieces of code
 - Allows for good code separation
- **Model-View-Controller (MVC) Pattern**
 - A model and its view are independent
 - A controller binds a model to its view
 - Allows for good code separation

Delegate Example

- An interface may define **Logging** methods
- A **ConsoleLogger** class may implement the **Logging** methods to log to a console
- A **CentralLogger** class may implement the **Logging** methods to log to a persistent store
- An object uses a **Logging** delegate to log its output
 - By default the object may log to the console using a **ConsoleLogger**
 - Its **Logging** delegate may be set to the **CentralLogger** to redirect its output to the persistent store

Adaptor Example

- **Problem**

- A **WireScanner** class and a **BPM** class are different, but both can measure beam position
 - The classes were implemented by different vendors who did not know about the other's devices
 - Each class implements different methods for measuring the beam position

- **Solution**

- A **PositionMeasurer** interface is defined to measure position but neither **WireScanner** nor **BPM** implement it
- **WireScanPositionAdaptor** and **BPMPositionAdaptor** classes are defined to implement **PositionMeasurer** and operate internally on instances of the **WireScanner** and **BPM** classes respectively

Model-View-Controller Example

- **You have two different models for calculating the beam energy from measurements**
- **You have written a view to display the current measured beam energy**
- **You wrote another view for displaying the history of the measured beam energy while updating with the live one**
- **Separate controllers allow you to bind the existing views and models as your wish**

Packages

- **Classes are organized into packages**
- **Packages provide namespace resolution**
- **Import statements resolve which package namespaces to use**
 - can import an entire package
 - can import individual classes within a package
- **java.lang** package is imported by default

Package Example

```
// Specify the package in which this class belongs
// All public classes within this package will be in this name space
package gov.sns.apps.greatapp;

import java.io.*;
import java.util.List;
import java.util.ArrayList;

/** Defines a Helper for my great application */
public class Helper {
    // Let's assume we have some code here

    /** Write something to the given file */
    public void write( final File aFile ) { // note that File is in the imported java.io package
        final List aList = new ArrayList(); // note that List and ArrayList were imported explicitly
        // Some code here to write to the given file
    }
}
```

Tour of Java Packages

- **Java includes thousands of classes in dozens of packages**
- **A few commonly used packages**

	Package	Description
Core	java.lang	System, thread control, math, numeric types
Utilities	java.util	Lists, hash tables
Input/Output	java.io	File operations, input/output streams
User Interface	javax.swing	User interface components and layout

Core Java Package Sample

java.lang

Class	Description
System	Manage the runtime environment; Standard input/output/error
Math	Common math functions (sine, cosine, logarithm, square root, ...)
Thread	Spawn and manage threads
Number	Concrete subclasses for numeric values (Double, Integer, Boolean, ...)
Class	Class introspection; Load classes; Instantiate an instance;

Java Utilities Package Sample

java.util

Class or Interface	Description
List	Interface for lists (get elements, add elements, count elements, iterate)
ArrayList	Concrete List implementation
Map	Interface for container of key-value pairs (get and put keyed values)
HashMap	Concrete Map implementation
Collections	Functions on collections (sort, search, rotate, reverse, ...)
Timer	Schedule operations

Java Input/Output Package Sample

java.io

Class	Description
File	File representation; File path operations; Get and Set file properties
InputStream	Read a stream of bytes
OutputStream	Write a stream of bytes
Reader	Read a character stream
Writer	Write a character stream

Java User Interface Package Sample

`javax.swing`

Class	Description
JFrame	Window
JDialog	Dialog box
JTable	Table view of data presented in row/ column format
JList	Vertical view of a list data
JScrollPane	Container which scrolls its content
JTextField	Control which takes text input
JButton	Control which handles mouse presses

Java Resources

- **Java Site:** <http://java.sun.com>
- **Java Guide:** <http://java.sun.com/j2se/1.5.0/docs/>
- **Java API:** <http://java.sun.com/j2se/1.5.0/docs/api/>
- **Java Download:** http://java.sun.com/javase/downloads/index_jdk5.jsp
- **Ant:** <http://ant.apache.org/>