Java Language Tutorial

Java the Language

Comments

Three styles:

- // comment on one line

- /* comment on one or
  more lines */

- /** documenting comment */

Identifiers

- Used to name objects, variables, etc.
- Start with
  - Letter (upper or lower case, case-sensitive)
  - "_"
  - "$"
- Examples:

```
int fuel_level;
```

```
char _filelist;

float $money;
```

Keywords

abstract	continue	for	null	synchronize d
boolean	default	if	package	this
break	do	implements	private	threadsafe
byte	double	import	protected	throw
	else	instanceof	public	throws
case	extends	int	return	transient
catch	false	interface	short	true
char	final	long	static	try
class	finally	native	super	void
volatile	float	new	switch	while

Data types

- boolean (1 bit)

- byte (8 bits - signed)
- char (16 bits - unsigned, UNICODE)
- short (16 bits signed)
- int (32 bits signed)
- long (64 bits signed)
- float (32 bits IEEE 754)

- double (64 bits IEEE 754)

Data Types

- ALWAYS have a pre-defined value
- byte, short, int, long, float, double

  initialized to zero
- char

  initialized to “\u0000” (NULL)
- all object reference types

initialized to null

Arrays

- Are objects
- Arrays have one instance variable length
- Can declare arrays of any type

```
int[] array1;

MyObject s[];
```

- Can build array of arrays

```
int a[][] = new int[10][3];

a.length --> 10

a[0].length --> 3
```

Creating arrays

- An empty array:

```
int list[] = new int [50];
```

- Pre-initialized:

```
String names[] = { “Marc”, “Tom”, “Pete” };
```

- Cannot create static compile time arrays

```
int nogood[20];   // compile time error
```

Operators

.   [ ]   ( )
++   --
!   ~   instanceof
*   /   %
+   -
<<   >>   >>>
<   >   <=   >=   ==   !=
& ^ |
&& ||
?:
=   op=   (*=   /=   %=   +=   -=   etc.)   ,

- instanceof tests whether specified object is class type:

```
if (event.target instanceof Button) { ...
```

Branching Statements

Looping Statements

Control flow

- Can use break label to “jump” out of loop

```
loop:    while (true)
         switch (c = in.read())) {
            case -1:
            case ‘\n‘:
               break loop;   // jumps past while
         ...
         }
```

```
}  // end while
```

Classes

- Every class contains data variables and methods
- Every class descends from another (at least Object)
- New classes create new types that are “derived” from the original class
- The derived class is called a “subclass”

- The original class is called the parent or “superclass”

- Derivation is transitive. Given A->B->C, then C is a subclass of A

Classes

- Syntax:

```
modifiers class classname
[extends Superclassname] [implements Interface{,
Interface}] {

     body

     }
```

- Modifiers:
  - abstract - an incomplete implementation;

cannot be instantiated

- final - cannot be subclassed
- public - accessible everywhere

Class Declaration

```
public class MyClass {
    int i;
    // MyClass constructor
    public MyClass() {
       i = 10;
    }
    public void add_to_i (int j) {
       i += j;
    }
}
```

- Class instance example

```
MyClass mc = new MyClass();
mc.add_to_i (20);
```

Subclassing

- Single inheritance!

```
// MyNewClass overrides add_to_i of MyClass
public class MyNewClass extends MyClass {
    public void add_to_i (int j) {
        i = i + (j / 2);
    }
}
```

- Example:

```
MyNewClass mnc = new MyNewClass();

mnc.add_to_i (20);
```

Abstract classes

- Allow a partial construction of a class definition
- abstract class rules:
  - can have abstract methods
  - cannot be instantiated
  - if one or more methods is abstract, the class must be

Interfaces

- An interface defines methods without bodies that are inherited and defined by the implementing class
- Multiple interfaces may be implemented
- Provide encapsulation of method protocols without restricting implementation to single inheritance tree
- Using interfaces allows several classes to share

a programming interface without being aware of each implementation

Interface Example

```
public interface Storing {

    void FreezeDry (URLConnection stream);

    void reconstitute(URLConnection stream);

}

public class Image implments Storing, Painting {
    void FreezeDry (URLConnection stream) {
       // Compress image before storing
       ...
    }
    void reconstitute (URLConnection stream) {
       // Decompress image before reading
       ...
}
```

Methods

- Methods are the operations performed on an object or class
- Can be declared in both classes and interfaces
- Can only be implemented in classes
- Method declaration syntax:

```
modifiers returnType methodName ( params ) {
    [methodBody]

}
```

- All methods must have a return type except constructors

Modifiers

- public - accessible everywhere
- protected - accessible by subclasses of the class and by members of the class package
- private - accessible only within the class body
- default - “friendly” - accessible throughout the package

- abstract - no implementation, class must also be declared abstract

Modifiers

- final - cannot be overridden by subclass
- static - implicitly final; can only refer to static variables
- native - a method implemented in some other language, usually C
- synchronized - per-instance (object) access lock

- acquired on entry
- released on exit
- primary mechanism for dealing with synchronization between multiple threads

Result type

- declares the type of value returned
- void can be used to not return a value
- Must have a return type
- Can return arrays’s using the [] syntax

Argument list

- Only pass-by-value

- Use type/name syntax:

```
String print_sum (int x, int y) {
   return (“Result is: ” + (x+y));
}
```

- Only arguments are used to distinguish methods

Overriding

- Methods with same name, return type and number and type of arguments in the class overrides the parent class method:

```
class Parent {
    int add_to_i (int i,int j) {
        return (i+=j);
    }
}
class Child extends Parent {
    int add_to_i (int i, int j) {
        return (i+=j/2);
```

```
          }

}
```

Overloading

- Methods with the same name, but different parameters, either count or type are overloaded:

```
class Parent {
    int add_to_i (int i,int j) {
       return (i+=j);
    }

    int add_to_i (float i, float j) {
       return ((int)i+=j);
    }
}
```

Constructors

- One or more optional constructors
- Same name as class and don’t specify a return type
- Called automatically upon object creation
- May be overloaded
- Default constructor is parent class constructor

- First statement may be

```
super(...); // calls superclass initializer

this(...);   // calls this class initializer
```

Finalizers

- Called once before garbage collection
- When (and if) called is non-deterministic
- Any uncaught exceptions are ignored
- Example:

```
protected void finalize () {
    try { myStream.close(); }
```

```
        catch (IOException e) {
            System.out.println ("Stream close failed");
        }
}
```

Object Scope and Garbage Collection

- Objects have a lifetime
- Objects that are no longer accessible should release resources
- Java implements a background GC thread
- Object scope example:

```
String s;   // No memory allocation

s = new String (“abc”);   // memory allocated

s = new String (“def”);   // new object
```

Static methods and variables

- Apply to the class itself, rather than an instance
- Static methods are implicitly final
- Static methods may access only static variables
- Static variables exist only once per class, regardless of how many instances are created
- Static variables and methods are accessed

using the class name, but may also be accessed using an instance name

- May also be applied to a block of code which is initialized once at runtime

Exceptions

- ArithmeticException
- NullPointerException
- IncompatibleClassChangeException
- ClassCastException
- NegativeArraySizeException
- OutOfMemoryException
- NoClassDefFoundException

- ArrayIndexOutofBoundsException
- UnsatisfiedLinkException
- InternalException

Creating New Exceptions

```
class MyVeryOwnException extends Exception { }


class MyClass{
    void oops(){
        if (some error occurred){
            throw new MyVeryOwnException();
        }
    }
}
```

Exception Handling

- Exceptions must either be caught or thrown
- You can catch exceptions and recover from them:

```
try { a[i] = 10;
} catch (ArrayIndexOutOfBounds e){
   System.out.println("a[] out of bounds");
} catch (MyVeryOwnException e){
   System.out.println("Caught my error");
} catch (Exception e){
   System.out.println (e.toString());
} finally {
   /* stuff to do regardless of whether an */
   /* exception was thrown     */
}
```

Native methods

- Call C and C++ functions from Java code
- 3 steps required
  - javah used to create header file and stubs file
  - write/modify C code to include java header
  - compile stubs file and C file into dynamic library

- Java code defines C call and loads function:

```
private native int time ();

static { System.loadLibrary (“time”); }
```

Packages

- The tool used to prevent namespace collision
- Packages group classes and interfaces
- Convention starts packages with company names (currently)
- Each class/interface is specified as part of a package name with:

```
package packageName;
```

- Compilation units without package are placed in a default namespace

Importing classes

- The import command specifies which packages to load
- Used by the compiler to mark class names
- Three ways to declare class names:
  - import all classes within a package:

```
import java.io.*;
```

- import specific class names:

```
import java.net.Socket;
```

- use fully qualified class names:

```
paint (java.awt.Graphics g) {
```

Java packages

- java.applet -- Image display and Audio classes
- java.awt -- Window, Frame, Button, and other GUI widget Components
- java.io -- File I/O, InputStream, OutputStream, PrintStream
- java.lang - Object, Thread, Exception, Ssytem, Strings, all the essential stuff

- java.net -- ServerSocket, Socket, URL, client/server connections, TCP/IP and UDP
- java.util -- Date, Vector, Hashtable, Stack, Random