Java Principles

Mark Allen Weiss
Copyright 2000

Java Philosophy

- **Syntax is similar to C++, but**
  - **Hard-to-use fluff is removed**
  - **Silly stuff is illegal**
  - **More effort added to catch silly mistakes at compile-time**
  - **error handling built in from day 1**
  - **inheritance supported very neatly**
  - **language written to be portable among platforms**
  - **large, portable APIs to handle drudery**

What's Completely Gone

- **Pointers / pointer math / *, -> operator**
- **preprocessor**
- **enum**
- **typedef**
- **parameter passing difficulties**
- **goto**
- **:: operator**
- **global functions and variables**
- **operator overloading (ouch!)**
- **templates (ouch!)**

New Stuff Not in C++

- **Compiler must do a conservative flow analysis to ensure variables are definitely assigned and return values are returned.**
- **Exception handling is first rate.**
- **Automatic garbage collection.**
- **Inheritance is easy to use, with automatic dynamic binding.**
- **Objects are accessed by reference variables (aka respectable pointers).**
- **Classes loaded on demand.**

 Copyright 1996, 1999, M. A. Weiss

The Environment

- **Command line interface**
  - **use notepad to enter program**
  - **`javac program.java` (applied for each source file)**
  - **`java program`**
- **JBuilder**
  - **Integrated environment with project file, debugger, and highlighted editor**
- **Source files must end in .java**
- **javac generates .class files (one per class)**
- **java loads class; goes to main method**

 Copyright 1996, 1999, M. A. Weiss

First Program

```
class FirstProgram
{
  public static void main( String args[ ] )
  {
    System.out.println( "Get me out of here!" );
  }
}
```

- **Place in `FirstProgram.java`**
- **Java is case-sensitive!**

 Copyright 1996, 1999, M. A. Weiss

Comments

```
/* this is a comment */
// comment extends to end of line
/** this is a javadoc comment */
```

- **Comments do not nest**

 Copyright 1996, 1999, M. A. Weiss

`import` directive

- **No `#include` statements in java**
- **`import` directive is similar to `using`**

 Copyright 1996, 1999, M. A. Weiss

main

- **In an application program execution begins at `main`. (This is not true for applets).**
- **`main` must have the following prototype:**

```
public static void main( String [] args )
```

- **`args[]` is similar to `argv[]`, except that `args[0]` is equivalent to `argv[1]`, and so on.**
- **`main` does not return a value.**

 Copyright 1996, 1999, M. A. Weiss

I/O

- **I/O, especially with files in Java is complex (because it is assumed the the majority of applications will use a GUI for terminal I/O and binary files otherwise).**
- **Use `System.out.println` for simple terminal output.**
- **We will discuss I/O at later date.**

 Copyright 1996, 1999, M. A. Weiss

Built-in Types and Object

- **The following are the Java built-in types:**
  - **byte, short, int, long**
  - **float, double**
  - **char**
  - **boolean**
- **Everything else is an object.**
- **A built-in type IS NOT AN OBJECT!!!**

 Copyright 1996, 1999, M. A. Weiss

Constant Things

- **Constant things (both built-in types and objects) can be defined class-wide and are both static and final.**

```
class Dumb
{
  public static final double PI = 3.14159265358979323;

  public static void main( String args[ ] )
  {
    System.out.println( "Pi is " + PI );
  }
}
```

 Copyright 1996, 1999, M. A. Weiss

Declarations of Built-in Types

- **Like C++**

```
int intVal;
double doubleVal = 0.0;
```

- **Can declare variables local to a function or global to a class.**
- **Can declare variables anywhere prior to use inside a function.**
- **Can declare a variable in the first expression of a for loop.**

 Copyright 1996, 1999, M. A. Weiss

Operators

- **All are like C++, unless indicated:**
  - **!, ++, --, unary +, unary -, . operator**
  - **+, -, *, /, %**
  - **==, !=, <, >, <=, >= (all return boolean)**
  - **&&, ||**
  - **&, |, ^, ~**
  - **<<, >>, >>> (slightly better defined)**
  - **?:**
  - **+=, -=, *=, /=, %=, etc.**
- **Some operators have better semantics across platforms, and left-to-right evaluation**

 Copyright 1996, 1999, M. A. Weiss

Type Conversions

- **Type mixing rules are stronger than C++.**
- **May need to use type conversion, which follows old C-style:**

```
int x = 4;
int y = 6;
double z = (double) x / y;
```

 Copyright 1996, 1999, M. A. Weiss

Conditional Statements

- **Same as C++ unless noted:**
  - **`if`, `while`, `do`, `for`, `switch`**
  - **`goto` is a reserved word, but you cannot use it.**
  - **`break` and `continue` (break and continue may be labelled).**

```
done:
   while( blah1 ) {
       while( blah2 )
           if( blah3 )
               break done;
   }
```

 Copyright 1996, 1999, M. A. Weiss

Functions

- **`public static` member is a C++ global function**
- ***All parameters* are passed using call-by-value.**
- **Recursion is supported.**
- **Function overloading is supported.**
- **Default parameters are not allowed.**
- **Inline functions are not allowed.**

 Copyright 1996, 1999, M. A. Weiss

Objects and References

- **All non-primitive entities are objects.**
- **Objects are**
  - **always created on the heap.**
  - **always accessed by reference variables**
- **Reference variables are logically pointers; also known as object handles.**
- **Legal operations on references:**
  - **assignment via =**
  - **comparison via == and != (compares handles)**
  - **accessing object members via . operator**
  - **instanceof operator**

 Copyright 1996, 1999, M. A. Weiss

Using Objects

- **new creates a new object; called with appropriate parameters specified in set of constructors**
- **Reference always references either**
  - **an object of appropriate type**
  - **null**
- **Invoking method from a null references generates a runtime exception**
- **= copies reference values, not object states.**

 Copyright 1996, 1999, M. A. Weiss

Strings

- **Java strings are halfway between built-in types and objects (they are closer to objects that built-ins). This is not a wonderful feature of the language!!**
- **The type is `String`. Case matters.**
- **Examples:**

```
String empty = "";
String message = "Hello";
String repeat = message;
```

 Copyright 1996, 1999, M. A. Weiss

String Operations

- **Strings are immutable!**
- **Concatenation uses +**
- **Concatenation with non-strings works too; note that you provide the blank space.**

```
System.out.println( "Pi is " + PI );
```

- **Substrings work too, but note that positions start at zero and you specify the starting point and the first non-included position.**

```
String greeting = "hello";
String s = greeting.substring( 0, 4 ); // s is "hell"
```

 Copyright 1996, 1999, M. A. Weiss

Strings: The Different

- **Use `.length( )` to get the string length:**

```
int n = greeting.length( ); // is 5
```

- **Use `.charAt( )` to get an individual char**
- **You cannot change an individual character in a string; you must alter the string as a single object.**

 Copyright 1996, 1999, M. A. Weiss

Strings: The Downright Ugly

- **Use `.equals( )` to compare for equality. Note: `==` determines if two strings are stored in the same location, so NEVER use it. Usage is**

```
string1.equals( string2 )
```

- **Use `.compareTo( )` to mimic the C-style `strcmp`. It returns <0, 0, or >0 depending on `string1` and `string2` in**

```
string1.compareTo( string2 )
```

- **Can use `.equalsIgnoreCase` or `compareToIgnoreCase`**

 Copyright 1996, 1999, M. A. Weiss

StringBuffer

- **Helper class useful for optimizing repeated string concatenations that are in different statements.**

```
// Efficient even if n=1000000
String manyAs( int n )
{
  StringBuffer sb = new StringBuffer( );
  for( int i = 0; i < n; i++ )
    sb.append( 'A' );

  return new String( sb );
}
```

 Copyright 1996, 1999, M. A. Weiss

Arrays: The Good

- **Arrays are indexed starting at 0.**
- **The number of items in the array is always accessible via `.length`. Note, that unlike strings, `length` is not a function, so don't use parentheses.**
- **Bounds checking is performed.**

 Copyright 1996, 1999, M. A. Weiss

Arrays: The Bad

- **Arrays are objects, not built-in types. Thus they have a different declaration syntax, and the meaning of = is different.**

 Copyright 1996, 1999, M. A. Weiss

Declaration and Definition

- **An object is declared to be an array of `int` as follows (note the [] can go in other places; stick to this style for now)**

```
int [ ] arrayOfInt;
```

- **No memory is allocated yet. To allocate, use `new`:**

```
arrayOfInt = new int[ 100 ];
```

- **Declaration and definition can be combined:**

```
int [ ] arrayOfInt = new int [ 100 ];
```

 Copyright 1996, 1999, M. A. Weiss

What = Means for Arrays

```
int [ ] array1 = new int [ 100 ];
int [ ] array2 = new int [ 100 ];
int [ ] array3;
array3 = new int [ 100 ];
array3 = array1;
```

- **`array3` now refers to the same memory as `array1`. Any change in either changes both!!! (Old memory is garbage collected)**
- **Reason: objects are references, references are pointers. So watch out!!**

 Copyright 1996, 1999, M. A. Weiss

Arrays and Functions

- **Can be returned (unlike C++).**
- **When passed as a parameter, contents can be changed. There is no `const` equivalent.**

 Copyright 1996, 1999, M. A. Weiss

Next Time

- **Objects and Classes**
- **Packages**
- **Inheritance**
- **Exceptions**

 Copyright 1996, 1999, M. A. Weiss