Programming in Java

Objects, Classes, Program Constructs

Program Structure/Environment

- Java
  - Is interpreted (C/C++ are Compiled)
  - No Preprocessor
  - No #define, #ifdef, #include, ...
- Main method (for Java applications)
  - Embedded in a Class

```
public class Xyz {
  public static void main (String args[]) {
    …
  }
}
```

  - Each class can define its own main method
  - Program's starting point depends on how the interpreter is invoked.

```
$ java Xyz
```


Command Line Arguments

- Command Line Args are passed to main method

```
public class Echo { // From JEIN
  public static void main(String argv[]) {
    for (int i=0; i<argv.length; i++)
      System.out.print(argv[i] + " ");
    System.out.print("\n");
    System.exit(0);
  }
}
```

- **`main`** has a return type of **`void`** (not **`int`**)
  - The **`System.exit`** method is used to return value back to OS
- The **`length`** property is used to return array size

For Statement

- Java's **`for`** stmt is similar to C/C++, except:
  - Comma operator is *simulated* in Java

```
for (i=0, j=0; (i<10) && (j<20); i++, j++) {
  …
}
```

    - Allowed in initialization and test sections
    - Makes Java syntactically closer to C
  - Variable declaration
    - variables can be declared within for statement, but can't be overloaded

```
…
int i;
for (int i=0; i<n; i++) { … } // Not valid in Java
```

    - declaration is all or nothing

```
for (int i=0, j=0; … ) // Declares both i and j
```

  - Conditional must evaluate to a boolean
    - Also true for if, while

If, While, Do While, Switch

- These are (essentially) the same as C/C++

```
if (x != 2)
  y=3;

if (x == 3)
  y=7;
else
  y=8;

if (x >= 4) {
  y=2;
  k=3;
}

while (x<100) {
  System.out.println
     ("X=" + x);
  x *= 2;
}
```

```
do {
  System.out.println ("X=" + x);
  x *= 2;
}

char c;
...
switch (c) {
case 'Q':
   return;
case 'E':
   process_edit();
   break;
default:
   System.out.println ("Error");
}
```


Name Space

- No globals
  - variables, functions, methods, constants
- Scope
  - Every variable, function, method, constant belongs to a *Class*
  - Every class is part of a *Package*
- Fully qualified name of variable or method
  <package>.<class>.<member>
  - Packages translate to directories in the "*class path*"
  - A package name can contain multiple components
    java.lang.String.substring()
    COM.Ora.writers.david.widgets.Barchart.display()
    - ***This class would be in the directory "XXX/COM/Ora/writers/david/widgets", where XXX is a directory in the "class path"***

Package; Import

- Package Statement
  - Specifies the name of the package to which a class belongs

```
package Simple_IO; // Must be the first statement
public class Reader {
  …
}
```

  - Optional
- Import Statement
  - Without an import statement

```
java.util.Calendar c1;
```

  - After the import statement

```
import java.util.Calendar;
...
Calendar c1;
```

  - Saves typing

```
import java.util.*;  // Imports all classes
```


Access Rules

- Packages are accessible
  - If associated files and directories exist and have read permission
- Classes and interfaces of a package are accessible
  - From any other class in the same package
  - *Public* classes are visible from other packages
- Members of a class (C) are accessible
  - [Default] From any class in the same package
  - *Private* members are accessible only from C
  - *Protected* members are accessible from C and subclasses of C
  - *Public* members are accessible from any class that can access C
- Local variables declared within a method
  - Are not accessible outside the local scope

Data Types

- Primitive Types
  - Integral (**`byte`**, **`short`**, **`char`** , **`int`**, **`long`**)
    - **`char`** is unsigned and also used for characters
  - Floating Point (**`float`**, **`double`**)
  - **`boolean`**
- Classes
  - Predefined classes
    - String, BigInteger, Calendar, Date, Vector, ...
    - Wrapper classes (Byte, Short, Integer, Long, Character)
  - User defined classes
  - "Special" classes
    - Arrays

Expressions

- Arithmetic expressions in Java are similar to C/C++
  - Example

```
int i = 5 + 12 / 5 - 10 % 3
```

= 5 + (12 / 5) - (10 % 3)
= 5 + 2 - 1
= 6

    - Operators cannot be overloaded in Java
    - Integer division vs. floating point division
    - Operator precedence

Objects

- Objects
  - Instances of classes are called objects
  - Object variables store the *address* of an object
    - Different from primitive variables (which store the actual value)
    - Primitive Data Type example

```
int i=3;
int j=i;
i=2;  // i==2; j==3
```

    - Object Example1

```
java.awt.Button b1 = new java.awt.Button("OK");
java.awt.Button b2 = b1;
b2.setLabel("Cancel"); // Change is visible via b1 also
b1 = new java.awt.Button("Cancel")
```

  - No explicit dereferencing (i.e., no **&**, ***** or **->** operators)
    - No pointers
    - **`null`** = "Absence of reference" = a variable not pointing to an object

Objects are handled by Reference

- Objects in Java are handled "by reference"
  - Comparison is by reference
    - Following is true if b1, b2 point to the same object

```
if (b1 == b2) { … }
if (b1.equals(b2)) { … } // member by member comparison
```

  - Assignment copies the reference

```
b1 = b2;
b1.clone(b2); // Convention for copying an object
```

- Parameters passing is *always* by value
  - The value is always copied into the method
  - For objects, the *reference* is copied (passed by value)
    - The object itself is not copied
    - It is possible to change the original object via the reference

Parameter Passing Example

```
class ParameterPassingExample {
  static public void main (String[] args) {
    int ai = 99;
    StringBuffer as1 = new StringBuffer("Hello");
    StringBuffer as2 = new StringBuffer("World");

    System.out.println ("Before Call: " + show(ai, as1, as2));
    set(ai,as1,as2);
    System.out.println ("After Call: " + show(ai, as1, as2));
  }
  static void set (int fi, StringBuffer fs1, StringBuffer fs2) {
    System.out.println ("Before Change: " + show(fi, fs1, fs2));
    fi=1;
    fs1.append(", World");
    fs2 = new StringBuffer("Hello, World");
    System.out.println ("After Change: " + show(fi, fs1, fs2));
  }
  static String show (int i, StringBuffer s1, StringBuffer s2) {
    return "i=" + i + "s1='" + s1 + "'; s2='" + s2 + "'";
  }
}
```


Constants

- Constants
  - Value of variable is not allowed to change after initialization
    - Example

```
final double PI = 3.14159;
```

    - Initialization can be done after declaration

```
final boolean debug_mode;
…
if (x<20) debug_mode = true;  // Legal
else debug_mode = false; // Legal
…
debug_mode = false; // Error is caught at compile time
```

    - Value of variable cannot change; value of object can change

```
final Button p = new Button("OK");
p = new Button ("OK"); // Illegal. P cannot point to
                       // a different object
p.setLabel ("Cancel"); // Legal.
```


Input/Output

- java.io.OutputStream - A byte output stream
  - System.out (C:stdout; C++:cout)
  - System.err (C:stderr; C++:cerr)
  - Convenience methods: **`print`**, **`println`**
    - send characters to output streams
- java.io.InputStream - A byte input stream
  - System.in (C:stdin; C++:cin)
  - InputStreamReader
    - Reads bytes and converts them to Unicode characters
  - BufferedReader
    - Buffers input, improves efficiency
    - Convenience method: **`readLine()`**

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader stdin = new BufferedReader (isr);
String s1 = stdin.readLine();
```


Echo.java

- A version of Echo that reads in data from System.in

```
import java.io.*;
class Echo {
  public static void main (String[] args) throws IOException
  {
    BufferedReader stdin = new BufferedReader
        (new InputStreamReader(System.in));
    String message;

    System.out.println ("Enter a line of text:");
    message = stdin.readLine();
    System.out.println ("Entered: \"" + message + "\"");
  }  // method main
}  // class Echo
```

- java.lang.Integer.parseInt converts a string to an integer

```
int message_as_int = Integer.parseInt(message);
```

- java.io.StreamTokenizer handles more advanced parsing