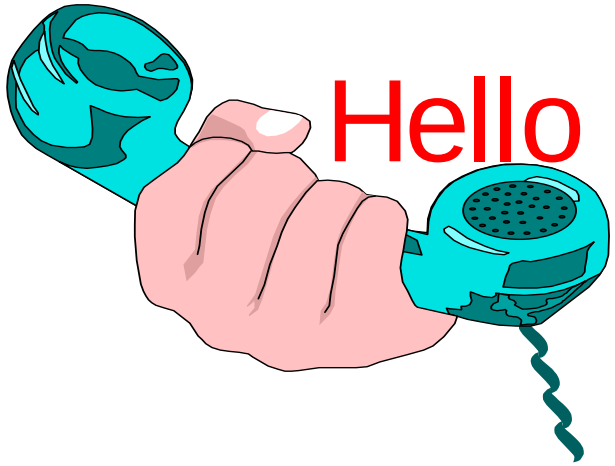
Basic Java Constructs and Data Types – Nuts and Bolts

Looking into Specific Differences and Enhancements in Java compared to C

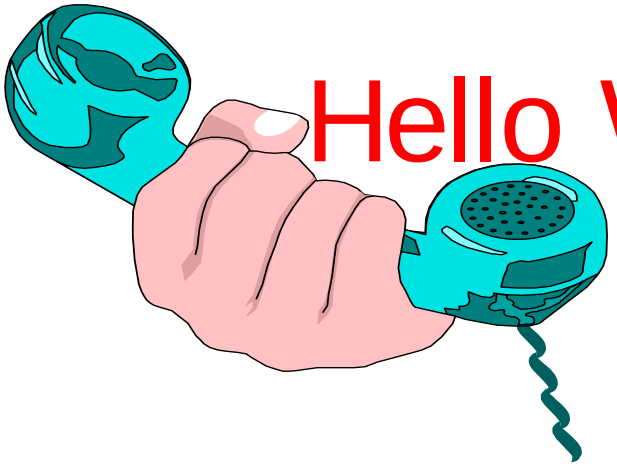
Contents

- Hello World Program Statements Explained
- Java Program Structure in General
- Java Classes and Static Methods
- Data Types, Variables and Constants
- Java Comments and Documents
- Control Flow
- Reading from Keyboard
- Command Line Arguments Processing
- Summary and References



Hello World

```
//  HelloWorld.java: Hello World program
import java.lang.*;
class HelloWorld
{
    public static void main(String args[])
    {
      System.out.println("Hello World");
    }
}
```



Hello World: Java and C

```
S1:  //  HelloWorld.java: Hello World program
S2:  import java.lang.*;
S3:  class HelloWorld
     {
S4:      public static void main(String args[])
         {
S6:          System.out.println("Hello World");
         }
     }
```

```
/*  helloworld.c: Hello World program */
#define <stdio.h>

void main(int argc, char *argv[])
{
   printf("Hello World\n");
}
```

Program Processing

- Compilation

```
# javac HelloWorld.java
```

results in HelloWorld.class

- Execution

```
# java HelloWorld
```

Hello World

Closer Look at - Hello World

- The class has one method – main()

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

- Command line input arguments are passed in the String array args[]

  e.g java HelloWorld John Jane

  args[0] – John args[1] – Jane

Closer Look at - Hello World

- import java.lang.*;
  - Java allows grouping of related classes into a package.
  - It allows different companies can develop different packages, may even have same class and method names, but they differ by package name:
    - ibm.mathlib.*
    - microsoft.mathlib.*
    - Helps in managing name clash problems.
  - Think of this package as library.
  - “import” statement somewhat serves similar purpose as C’s #include
  - If you don’t add import statement, then you need utilise fully qualified name.
    - ibm.mathlib.sin()
    - If you do “import ibm.*” then you can use mathlib.sin() instead.

Java imports java.lang.* by default

- So, You don't need to import java.lang.*
- That means, you can invoke services of java's "lang" package classes/entities, you don't need to use fully qualified names.
  - We used System.out.println() instead of
    - java.lang. System.out.println()

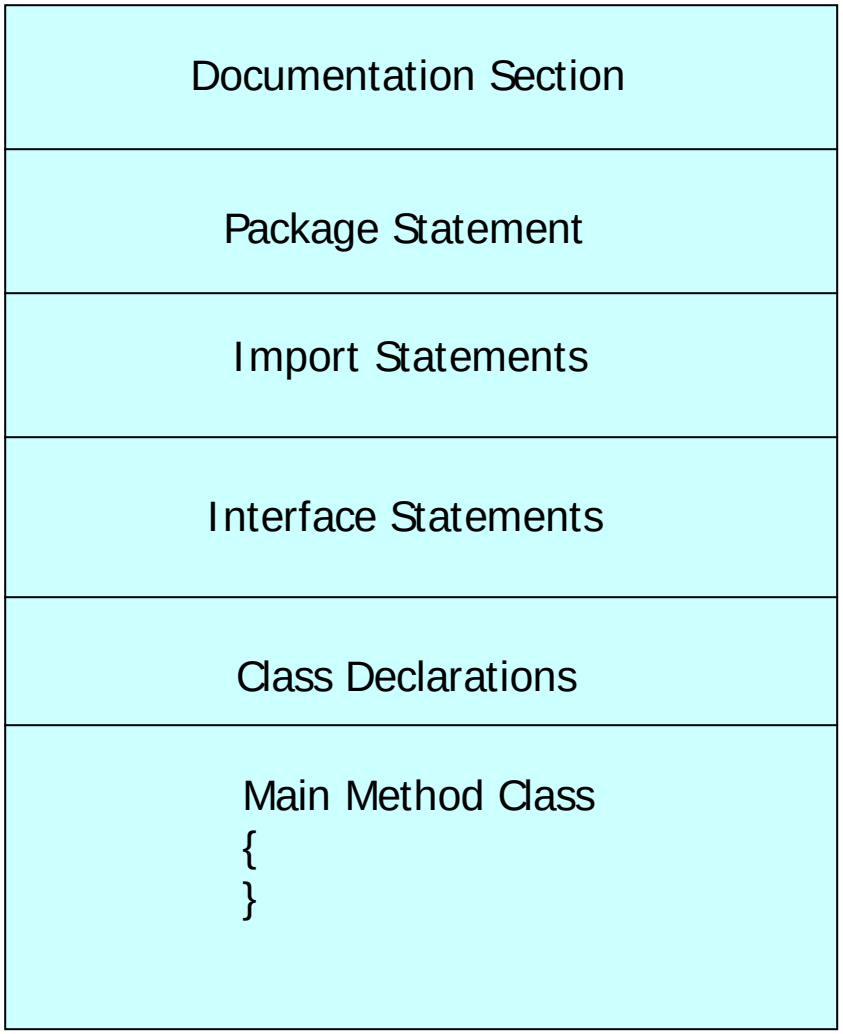
public static void main(String args[])

- public: The keyword “public” is an access specifier that declares the main method as unprotected.
- static: It says this method belongs to the entire class and NOT a part of any objects of class. The main must always be declared static since the interpreter users this before any objects are created.
- void: The type modifier that states that main does not return any value.

System.out.println("Hello World");

- java.lang.*
  - All classes/items in "lang" package of java package.
- System is really the java.lang.System class.
- This class has a public static field called out which is an instance of the java.io.PrintStream class. So when we write System.out.println(), we are really invoking the println() method of the "out" field of the java.lang.System class.

Java Program Structure



More Java: Classes and static methods

```
// SquareRoot.java: compute square root of number

import java.lang.Math;

class SquareRoot
{
   public static void main(String args [])
   {
        double x = 4;
        double y;
        y = Math.sqrt(x);
        System.out.println("y= "+y);
   }
}
```

Basic Data Types

- Types

boolean	**either true or false**
char	**16 bit Unicode 1.1**
byte	**8-bit integer (signed)**
short	**16-bit integer (signed)**
int	**32-bit integer (signed)**
long	**64-bit integer (singed)**
float	**32-bit floating point (IEEE 754-1985)**
double	**64-bit floating point (IEEE 754-1985)**

- String (class for manipulating strings)
- Java uses Unicode to represent characters internally

Variables

- Local Variables are declared within the block of code
- Variable has a type preceding the name
- Initial value is set by initialization expressions.

```
type variableName = initialValue;
```

e.g. `int x = 1;`

- Variables can be defined just before their usage (unlike C)
  - e.g., `for( int i = 0; i < 10; i++)`

Constants

- Constants are similar to variables except that they hold a fixed value. They are also called "READ" only variables.

- Constants are declared with the reserved word "final".

```
final  int MAX_LENGTH = 420;
final  double  PI = 3.1428;
```

- By convention upper case letters are used for defining constants.

Declaring Constants - example

```
class CircleArea
{
   public static void main(String args[])
   {
        final double PI = 3.1428;
        double radius = 5.5; // in cms
        double area;

        area = PI * radius * radius;

        System.out.println("Circle Radius = "+radius+" Area="+area);
   }
}
```

Comments

- English text scattered through the code are comments
- JAVA supports 3 types of comments

/* */ - Usually used from multi-line comments

// - Used for single line comments

/** */ - Documentation comments

Javadoc

- Effort to make Java self-documenting
- True OOP style, encapsulate documentation within code :)
- Comments beginning with /** and ending with */ can be extracted and turned into html documentation

Control Flow

- Control Flow Statements in JAVA
  - while loop
  - for loop
  - do-while loop
  - if-else statement
  - switch statement
- JAVA does not support a *goto* statement

Control Flow - Examples

- while loop

```
while (squared <=  MAX) {
        squared = lo * lo; // Calculate square
        System.out.println(squared);
        lo = lo + 1;    /* Compute the new  lo value */
}
```

Control Flow - Examples

- for loop

```
for (int i = 1; i < MAX;  i++) {
  System.out.println(i);     // prints 1 2 3 4 5 …
}
```

Control Flow - Examples

- do-while loop

```
do {
        squared = lo * lo; // Calculate square
        System.out.println(squared);
        lo = lo + 1;    /* Compute the new  lo value */
} while (squared <=  MAX);
```

Control Flow - Examples

- if-else loop

```
if ( i < 10) {
    System.out.println(“i is less than 10” );
}
else {
   System.out.println(“i is greater than or equal to 10”);
}
```

Control Flow - Examples

- switch statement

```
switch (c) {
   case ‘a’:
            System.out.println (“ The character is ‘a’” );
            break;
   case ‘b’;
           System.out.println (“ The character is ‘b’” );
            break;
   default;
           System.out.println (“ The character is not ‘a’ or ‘b’” );
            break;
}
```

Reading from Keyboard

- As Java does not support a simple APIs for Keyboard Input, we created a class called "Keyboard", which you can use in your program. The Keyboard class facilitates keyboard input by abstracting details about input parsing, conversions, and exception handling. This class reads from standard input (keyboard) and converts the characters into an appropriate type based on the method you call. Some methods you can use are:
  - Keyboard.readString()
  - Keyboard.readWord()
  - Keyboard.readChar()
  - Keyboard.readBoolean()
  - Keyboard.readInt()
  - Keyboard.readLong()
  - Keyboard.readFloat()
  - Keyboard.readDouble()

Keyboard class Usage Example

- Simply copy the Keyboard.java file from: http://www.cs.mu.oz.au/254/Keyboard/keyboard.html into your program directory and access its methods as if they are standard methods. The Java complier will link them automatically.
- An Example Program:

```
// A class to execute one or more Keyboard methods
class Test
{
    public static void main(String[] args)
    {
        System.out.print("Please enter a string: ");
        String str = Keyboard.readString();
        System.out.println("String = " + str);

        System.out.print("Please enter an int number: ");
        int numInt = Keyboard.readInt();
        System.out.println("int = " + numInt);
    }
}
```

Command Line Arguments

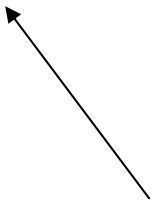
- Command line arguments provide one of the ways for supplying input data at the time of execution instead of including them in the program. They are supplied as parameters to the main() method:

  public static void main(String args[])

- "args" is declared of an array of strings (aka string objects).
  - args[0] is the first parameter, args[1] is the 2nd argument and so on
- The number of arguments passed identified by:
  - args.length
  - E.g. count = args.length;
- Example Invocation and values:
  - java MyProgram hello melbourne
  - args.length will be 2
  - args[0] will be "hello" and args[1] will be "melborune"

Printing command line arguments

```
// ComLineTest.java: testing command line arguments
class ComLineTest
{
   public static void main(String args[])
   {
        int count, i = 0;
        String myString;
        count = args.length;
        System.out.println("Number of Arguments = "+count);
        while( i < count )
        {
                myString = args[i];
                i = i + 1;
                System.out.println(i + " : " + "Java is "+myString+ " !");
        }
   }
}
```



+ concatenates strings or numbers

Execution Example

- `java ComLineTest Simple Object_Oriented Distributed Robust Secure Portable Multithread Dynamic`

- The output of program will be:

```
Number of Arguments = 8
1 : Java is Simple !
2 : Java is Object_Oriented !
3 : Java is Distributed !
4 : Java is Robust !
5 : Java is Secure !
6 : Java is Portable !
7 : Java is Multithread !
8 : Java is Dynamic !
```

Summary

- We discussed meaning of statements in “hello world” program
- We discussed various basic constructs and syntax.
- Apart from OO specific items, most keywords or constructs in Java have similar meaning and usage style as C.

References

- Chapter 3: Overview of Java Language
- Chapters To Browse (if you have forgotten C syntax/constructs):
  - Chapter 4: Constants, Variables, and Data Types
  - Chapter 5: Operators and Expressions
  - Chapter 6: Decision Making and Branching
  - Chapter 7: Decision Making and Looping