CMSC131

Introduction to your Introduction to Java

Why Java?

It’s a popular language in both industry and many introductory programming courses.

It makes use of programming structures and techniques that can be transferred to using a wide variety of other languages.

Development on different OS platforms without platform-specific differences for what we will be doing is more straight-forward.

Phrases you might hear…

Garbage collection

- …when we talk about requesting memory to hold information for our program to use…

Cross-platform executables

- …the modules we build and even full applications can be used under different operating systems due to the Java virtual machine…
- …it is possible to build to a specific platform to take advantage of native code and the related advantages, but we won't…

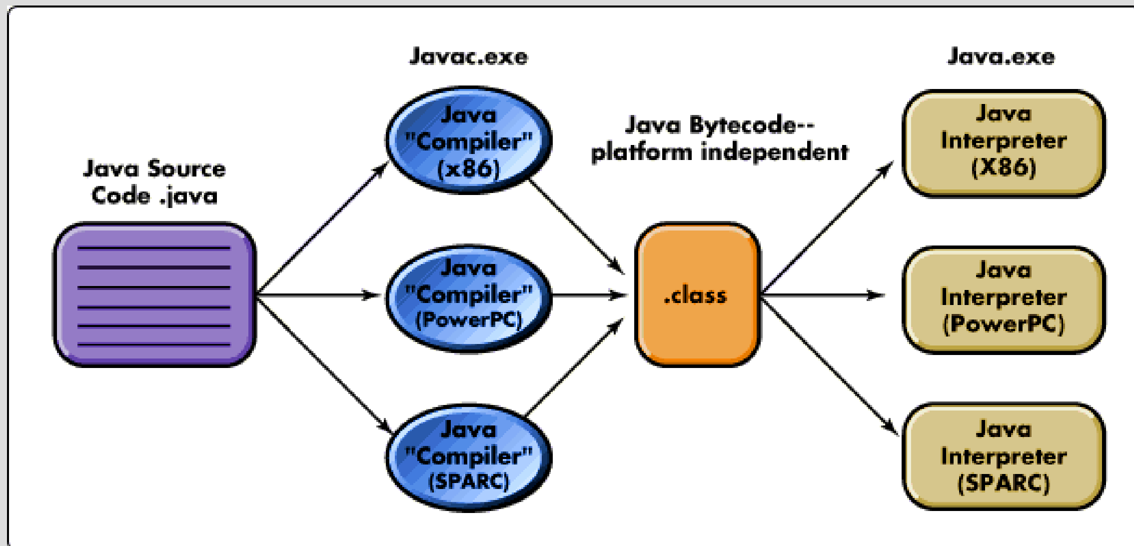
The JVM

We won't use the Java compiler to compile to the machine code of an actual physical machine's architecture, but rather to "bytecode" which is run on a Java Virtual Machine.

If you compile a JVM for a new operating system, then (in theory) all of your existing Java programs could run there too.

Again, there are times when (for efficiency) Java bytecode is actually compiled to a specific architecture's machine code (but is no longer portable).

Java to Bytecode to Execution



http://support.novell.com/techcenter/articles/img/ana1997070102.gif

Java is designed for large projects

One of the initial disadvantages to using Java is that a simple "one-line program" to print something like "Hello World!" is one line of code to accomplish that ***plus*** several lines of code to provide the framework that an object-oriented programming language like Java typically requires.

Everything in Java must live in a **Class.** The code that is meant to be the starting point of the program must live in a **Main Method**.

HelloWorld.java example

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

When compiled and executed, this program displays (to the terminal) the phrase in quotation marks.

Output

We will often use text-based output, especially for initial debugging, and see two basic options for sending text to the console:

System.out.println(*thing_to_print*);
System.out.print(*thing_to_print*);

Both are sent something to display. The difference between the two is that the first one adds an “end of line” character at the end of whatever it has sent to the console.

You can request the contents of variables to be displayed (which a variable may or may not support) or you can ask for a literal string to be displayed (contained in quotation marks in your output command).

Variables

Within a computer program you will need to store information in memory.

Variables of different types are used to do this. In essences, they all provide a way to give a human-readable name to a location in the computer's memory that can be used to read from and possibly write to that memory.

Some languages require you to declare the type of a variable before you use it, and Java is one of those languages.

Primitive Variables

In object-oriented languages we will see two categories of variables (primitives and class references) but for now we will focus on **primitive** variable types and one basic class type.

- Numeric Primitives: ***byte, short, int, long, float, double***
- Other Primitives: ***char, boolean***
- Objects: ***String***

For example, to declare a primitive variable to allocate some memory for us to use to store an integer value, and then store the value 0 there, we could use the following lines of code:

```
int counter;
counter = 0;
```

Simple Math Operation

Let's say that we have our integer variable and we assign an initial value into it, and then at some later point we decide we want to add 5 to whatever the current value there happens to be.

```
int counter;
counter = 0;

//some other code here...

counter += 5;
//the above instruction says to
//   read the current value from "counter"
//   add 5 to that read value
//   store the new value back into "counter"
```

Some Primitive Types and Sizes

- Integer values (and how many bytes they get)
  - byte (1), short (2), int (4), long (8)
- Real numbers (and how many bytes they get)
  - float (4), double (8)
- Individual **Unicode** characters
  - char (2)
- Boolean truth values
  - boolean (1)

Range of Values

The range of values depends on several things, one of which is how much memory is available for storing the value and another is how Java interprets the 0s and 1s stored in that memory.

A **byte** is one byte in size and can store a value between **-128** and **+127** but **boolean** is also one byte in size yet it can only store two values; **true** or **false**.

An **int** is four bytes in size and can store a value between **-2,147,483,648** and **+2,147,483,647** but a **float** is also four bytes in size yet it can store numbers between (roughly) $\mathbf{-3.4 \times 10^{38}}$ and $\mathbf{3.4 \times 10^{38}}$ (though only with 7 digits of precision).

What happens when you get too big?

- 0% A. The program crashes.
- 0% B. The program allocates more space.
- 0% C. The program throws an exception.
- 0% D. The value becomes 32,767.
- ✓ 0% E. The value becomes -32,768.
- 0% F. The value becomes 0.

Response Counter

If a ***short*** integer holds values from -32,768 to 32,767, what do you think happens if it is holding the value 32,767 and you **add 1** to it?

Fastest Responders

Seconds	Participant	Seconds	Participant

Widening/Narrowing

In Java, by default, real number values typed in as literals are assumed to be "double" values.

float val = 17.5; won't work because a ***float*** variable is a 4-byte data type for real numbers and a ***double*** variable is an 8-byte data type for real numbers.

Java allows automatic "widening" (you can assign something smaller to something bigger) but not automatic narrowing.

… so it needs to be **float val = 17.5F;**

Hierarchy

There are two classifications of automatic conversion attempts; widening and narrowing.

- Automatic widening is valid in Java.
- Automatic narrowing is invalid in Java.

The hierarchy for primitive numerics in Java is:

byte→short→int→long→float→double

Some Math Operations

Addition: **+** (also used for String concatenation)

Subtraction: **-** (also used as unary "negative")

Multiplication: *

Division: /

- Integer division discards remainder

  17 / 5 yields **3**

Modulus: **%**

- Returns what the remainder would be if the two operands were "divided"

  17 % 5 yields **2**

QuickMath.java example

```
public class QuickMath {
  public static void main(String[] args) {
  int x, y, z;      ← Variable Declarations

     x = 17;
     y = 23;                  ← Commands/Assignments/Instructions
     z = x + y;
     x = 42;

     System.out.println(x);          ← Output Instructions
     System.out.println(y);
     System.out.println(z);
     System.out.println(x + y);
  }
}
```

System.out.println(x) displays what?

- 0% A. 17
- 0% B. 42
- 0% C. Some other value.

```
x = 17;
y = 23;
z = x + y;
x = 42;

System.out.println(x);
System.out.println(y);
System.out.println(z);
System.out.println(x + y);
```

Response Counter

System.out.println(z) displays what?

0% A. 17

0% B. 42

0% C. Some other value.

```
x = 17;
y = 23;
z = x + y;
x = 42;

System.out.println(x);
System.out.println(y);
System.out.println(z);
System.out.println(x + y);
```

Response Counter

Fastest Responders

Seconds	Participant	Seconds	Participant

Addition

There are multiple ways to add values in Java. For example:

```
x += 1;
x = x + 1;
x++;
```

Each line of code **x=x+1;** would essentially "instruct the computer" to add 1 to the value stored in x, but they do so in slightly different ways behind the scenes.

The ++ Operator

Also, more generally, the **++** operator allows you to ***increment*** the values held by certain data types by a single "unit" of value.

For example, it can be used with any of the ***numeric primitives*** or ***char***, but cannot be used on things like ***boolean*** or ***String***.

While a syntactic detail, it can be interesting or even useful to know that there are actually two different ++ operators.

```
x++; //post increment
++x; //pre increment
```

Let’s go into Eclipse and see what the output of this block of code will be…

```
x=1;
  System.out.println(x++);
  System.out.println(x);

x=1;
  System.out.println(++x);
  System.out.println(x);

x=1;
  System.out.println(x++ + ++x);
  System.out.println(x);
```

What is `'a'+'b'` in Java?

0% A. ab

0% B. c

0% C. Ã

0% D. 195

Response Counter

How to store characters in Base 2

The typical way to represent a character (like “A”) is to have a standard conversion table that assigns a number to each characters.

ASCII (American Standard Code for Information Interchange) is one:

- (“A” = 65 = 01000001)

Unicode is another:

- (“蓝” = 34013 = 1000010011011101)

Which requires more BITs to store the letter “M” on a computer?

0% A. ASCII

0% B. Unicode

0% C. They use the same amount

Response Counter

Comparison Operators (for numeric primitives and char)

Less than:	**<**
Greater than:	**>**
Less than or equal to:	**<=**
Greater than or equal to:	**>=**
Is equal to:	**==**
Is not equal to:	**!=**

NOTE 1: = is for assignment, == is for comparison

NOTE 2: == when used with classes will be a topic that we discuss in more detail later

Comps.java example

```
public class Comps {
  public static void main(String[] args) {
  boolean flag;
     flag = (7<14);
     System.out.println(flag);
     flag = (17<14);
     System.out.println(flag);
     flag = (17=14); //THIS WON'T COMPILE
  }
}
```

Constants

There are times when you might want to have something like a named variable but don’t want it to actually *be* variable.

- These are typically called constants; named pieces of memory whose values cannot be altered via instructions using that name.
- In Java, what we have are final variables; once they are given a value, that value can not be changed by any of the operators we have.

The syntax is to put the keyword **final** in front of the type in the declaration line.

Comments

There are times that you will want to leave notes for yourself and other programmers within the code without it being executed.

This is where comments come in!

There are several types of comments in Java:

```
/* comment goes until the "close"  marker */
/* comment goes until the "close“ marker
      which might not be on the same line
      as where the "open" appears  */

// comment goes from "start" to the end of that line
```

Copyright © 2010-2019 : Evan Golub