Introduction to Java

Sergio Luján Mora

Contents

• What is Java?
• The programming language
• The platform
• Benefits of Java
• My first program: Hello World!
• My second program: Repeat
• Comments
• Variables
• Naming
• Primitive data types
• Arrays
• Operators
• Control flow statements
What is Java?

• Java technology is both a **programming language** and a **platform**

• JavaScript (Netscape) ≠ Java (Sun Microsystems)
  – JavaScript’s LiveConnect → Java

The programming language

• The Java programming language is a high-level language that can be characterized by all of the following features:
  – Simple
  – Architecture neutral
  – Object oriented
  – Portable
  – Distributed
  – High performance
  – Multithreaded
  – Robust
  – Dynamic
  – Secure
The programming language

- All source code is written in plain text files ending with the .java extension
- Source files are compiled into .class files by the javac compiler
- A .class file does not contain code that is native to your processor: contains bytecodes
  - Bytecodes: the machine language of the Java Virtual Machine (JVM)
- The java launcher tool then runs your application with an instance of the JVM
• JVM is available on many different operating systems
  – The same .class files are capable of running on Microsoft Windows, the Solaris TM Operating System (Solaris OS), Linux, or Mac OS
• Some virtual machines, such as the Java HotSpot virtual machine, perform additional steps at runtime to give your application a performance boost
  – Finding performance bottlenecks
  – Recompiling (to native code) frequently used sections of code.
The platform

• A platform is the hardware or software environment in which a program runs
  – Microsoft Windows, Linux, Solaris OS, and Mac OS
• Most platforms can be described as a combination of the operating system and underlying hardware
• The Java platform differs from most other platforms in that it’s a software-only platform that runs on top of other hardware-based platforms

The platform

• The Java platform has two components:
  – The Java Virtual Machine
  – The Java Application Programming Interface (API)
The platform

• The API is a large collection of ready-made software components that provide many useful capabilities
  – It is grouped into libraries of related classes and interfaces; these libraries are known as packages
• As a platform-independent environment, the Java platform can be a bit slower than native code
  – Advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability

Benefits of Java

• Get started quickly: Although the Java programming language is a powerful object-oriented language, it’s easy to learn, especially for programmers already familiar with C or C++.
• Write less code: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program written in C++.
• Write better code: The Java programming language encourages good coding practices, and automatic garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans™ component architecture, and its wide-ranging, easily extendible API let you reuse existing, tested code and introduce fewer bugs.
Benefits of Java

• **Develop programs more quickly:** The Java programming language is simpler than C++, and as such, your development time could be up to twice as fast when writing in it. Your programs will also require fewer lines of code.

• **Avoid platform dependencies:** You can keep your program portable by avoiding the use of libraries written in other languages.

• **Write once, run anywhere:** Because applications written in the Java programming language are compiled into machine-independent bytecodes, they run consistently on any Java platform.

• **Distribute software more easily:** With Java Web Start software, users will be able to launch your applications with a single click of the mouse. An automatic version check at startup ensures that users are always up to date with the latest version of your software. If an update is available, the Java Web Start software will automatically update their installation.

Hello World!

```java
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */

class HelloWorldApp {
    public static void main(String[] args) {
        // Display the string.
        System.out.println("Hello World!");
    }
}
```
Hello World!

• Compile:
  – javac HelloWorldApp.java

• Compiler generates file:
  – HelloWorldApp.class

Hello World!

• Run:
  – java HelloWorldApp

• The program prints:
import java.io.*;
class RepeatApp {
    public static void main(String[] args) throws IOException {
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        String userInput;
        System.out.println("Write Bye. to finish");
        System.out.print("Tell me: ");
        while ((userInput = stdIn.readLine()) != null) {
            System.out.println("You say: " + userInput);
            if (userInput.equals("Bye."))
                break;
            System.out.print("Tell me: ");
        }
    }
}
Repeat

- Run:
  - `java RepeatApp`

- The program prints:

\[\text{Screen capture showing the output of the program.}\]

Comments

- Comments are ignored by the compiler but are useful to other programmers (and for yourself!)
- The Java programming language supports three kinds of comments:
  - `/* text */`
    - The compiler ignores everything from `/*` to `*/`.
  - `/** documentation */`
    - This indicates a documentation comment (doc comment, for short)
    - The compiler ignores this kind of comment, just like it ignores comments that use `/*` and `*/`
    - The javadoc tool uses doc comments when preparing automatically generated documentation
  - `// text`
    - The compiler ignores everything from `//` to the end of the line
Questions

1. When you compile a program written in the Java programming language, the compiler converts the human-readable source file into platform-independent code that a Java Virtual Machine can understand. What is this platform-independent code called?

2. Which of the following is not a valid comment:
   a. /** comment */
   b. /* comment */
   c. /* comment
   d. // comment

3. What's the first thing you should check if you see the following error at runtime:
   Exception in thread "main"

4. What is the correct signature of the main method?

5. When declaring the main method, which modifier must come first, public or static?

6. What parameters does the main method define?
Questions

• The following code has some errors:

class HelloWorldApp2 {
    public void main(String[] args) {
        System.out.println("Hello World!")
    }
}

Variables

• The Java programming language defines the following kinds of variables:
  – **Instance Variables (Non-Static Fields)** Technically speaking, objects store their individual states in "non-static fields", that is, fields declared without the static keyword. Non-static fields are also known as *instance variables* because their values are unique to each *instance* of a class (to each object, in other words); the currentSpeed of one bicycle is independent from the currentSpeed of another.
  – **Class Variables (Static Fields)** A *class variable* is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated. A field defining the number of gears for a particular kind of bicycle could be marked as static since conceptually the same number of gears will apply to all instances. The code `static int numGears = 6;` would create such a static field. Additionally, the keyword `final` could be added to indicate that the number of gears will never change.
Variables

- The Java programming language defines the following kinds of variables:
  - **Local Variables** Similar to how an object stores its state in fields, a method will often store its temporary state in local variables. The syntax for declaring a local variable is similar to declaring a field (for example, int count = 0;). There is no special keyword designating a variable as local; that determination comes entirely from the location in which the variable is declared — which is between the opening and closing braces of a method. As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.
  - **Parameters** You've already seen examples of parameters, both in the Bicycle class and in the main method of the "Hello World!" application. Recall that the signature for the main method is public static void main(String[] args). Here, the args variable is the parameter to this method. The important thing to remember is that parameters are always classified as "variables" not "fields". This applies to other parameter-accepting constructs as well (such as constructors and exception handlers) that you'll learn about later in the tutorial.

Naming

- Variable names are case-sensitive
  - A variable's name can be any legal identifier — an unlimited-length sequence of Unicode letters and digits, beginning with a letter, the dollar sign "$", or the underscore character "_".
  - The convention, however, is to always begin your variable names with a letter, not "$" or "_".
  - Additionally, the dollar sign character, by convention, is never used at all.
  - You may find some situations where auto-generated names will contain the dollar sign, but your variable names should always avoid using it.
  - A similar convention exists for the underscore character; while it's technically legal to begin your variable's name with "_", this practice is discouraged.
  - White space is not permitted.
Naming

- Subsequent characters may be letters, digits, dollar signs, or underscore characters
  - Conventions (and common sense) apply to this rule as well
  - When choosing a name for your variables, use full words instead of cryptic abbreviations
  - Doing so will make your code easier to read and understand. In many cases it will also make your code self-documenting; fields named cadence, speed, and gear, for example, are much more intuitive than abbreviated versions, such as s, c, and g
  - Also keep in mind that the name you choose must not be a keyword or reserved word

<table>
<thead>
<tr>
<th>abstract</th>
<th>continue</th>
<th>for</th>
<th>new</th>
<th>switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert</td>
<td>default</td>
<td>goto</td>
<td>package</td>
<td>synchronized</td>
</tr>
<tr>
<td>boolean</td>
<td>do</td>
<td>if</td>
<td>private</td>
<td>this</td>
</tr>
<tr>
<td>break</td>
<td>double</td>
<td>implements</td>
<td>protected</td>
<td>throw</td>
</tr>
<tr>
<td>byte</td>
<td>else</td>
<td>import</td>
<td>public</td>
<td>throws</td>
</tr>
<tr>
<td>case</td>
<td>enum</td>
<td>instanceof</td>
<td>return</td>
<td>transient</td>
</tr>
<tr>
<td>catch</td>
<td>extends</td>
<td>int</td>
<td>short</td>
<td>try</td>
</tr>
<tr>
<td>char</td>
<td>final</td>
<td>interface</td>
<td>static</td>
<td>void</td>
</tr>
<tr>
<td>class</td>
<td>finally</td>
<td>long</td>
<td>strictfp</td>
<td>volatile</td>
</tr>
<tr>
<td>const</td>
<td>float</td>
<td>native</td>
<td>super</td>
<td>while</td>
</tr>
</tbody>
</table>
Naming

- If the name you choose consists of only one word, spell that word in all lowercase letters
- If it consists of more than one word, capitalize the first letter of each subsequent word
- The names gearRatio and currentGear are prime examples of this convention
- If your variable stores a constant value, such as static final int NUM_GEAR = 6, the convention changes slightly, capitalizing every letter and separating subsequent words with the underscore character
- By convention, the underscore character is never used elsewhere

Primitive data types

- **byte**:  
  - An 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive)
- **short**:  
  - A 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive)
- **int**:  
  - A 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive)
- **long**:  
  - A 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive)
Primitive data types

• float:
  – A single-precision 32-bit IEEE 754 floating point
  – This data type should never be used for precise values, such as currency. For that, you will need to use the `java.math.BigDecimal` class instead

• double:
  – A double-precision 64-bit IEEE 754 floating point

• boolean:
  – It has only two possible values: true and false
  – This data type represents one bit of information, but its "size" isn't something that's precisely defined.

• char:
  – A single 16-bit Unicode character
  – It has a minimum value of `\u0000` (or 0) and a maximum value of `\uffff` (or 65,535 inclusive).

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Default Value (for fields)</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>0</td>
</tr>
<tr>
<td>short</td>
<td>0</td>
</tr>
<tr>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>long</td>
<td>0L</td>
</tr>
<tr>
<td>float</td>
<td>0.0f</td>
</tr>
<tr>
<td>double</td>
<td>0.0d</td>
</tr>
<tr>
<td>char</td>
<td><code>\u0000</code></td>
</tr>
<tr>
<td>String (or any object)</td>
<td>null</td>
</tr>
<tr>
<td>boolean</td>
<td>false</td>
</tr>
</tbody>
</table>
**Primitive data types**

- Java also provides special support for character strings via the `java.lang.String` class.
- Enclosing your character string within double quotes will automatically create a new String object:
  ```java
  String s = "this is a string";
  ```
- String objects are **immutable**, which means that once created, their values cannot be changed.
- The `String` class is not technically a primitive data type, but considering the special support given to it by the language, you'll probably tend to think of it as such.

**Arrays**

- An **array** is a container object that holds a fixed number of values of a single type.
- The length of an array is established when the array is created.
- After creation, its length is fixed.

![Array Diagram](image-url)
Arrays

class ArrayDemo {
    public static void main(String[] args) {
        int[] anArray; // declares an array of integers
        anArray = new int[10]; // allocates memory for 10 integers
        anArray[0] = 100;
        ...
        anArray[9] = 1000;
        System.out.println("Element at index 0: " + anArray[0]);
        System.out.println("Element at index 1: " + anArray[1]);
        System.out.println("Element at index 2: " + anArray[2]);
        System.out.println("Element at index 3: " + anArray[3]);
        System.out.println("Element at index 4: " + anArray[4]);
        System.out.println("Element at index 5: " + anArray[5]);
        System.out.println("Element at index 6: " + anArray[6]);
        System.out.println("Element at index 7: " + anArray[7]);
        System.out.println("Element at index 8: " + anArray[8]);
        System.out.println("Element at index 9: " + anArray[9]);
    }
}

Arrays

• Alternatively, you can use the shortcut syntax to create and initialize an array:

    int[] anArray = {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000};

• Here the length of the array is determined by the number of values provided between { and }

• The built-in length property determines the size of any array
Arrays

- The System class has an arraycopy method that you can use to efficiently copy data from one array into another:

```java
public static void arraycopy(Object src,
   int srcPos, Object dest, int destPos,
   int length)
```

class ArrayCopyDemo {
   public static void main(String[] args) {
      char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };
      char[] copyTo = new char[7];
      System.arraycopy(copyFrom, 2, copyTo, 0, 7);
      System.out.println(new String(copyTo));
   }
}
```
Questions

1. Create a small program that defines some fields. Try creating some illegal field names and see what kind of error the compiler produces. Use the naming rules and conventions as a guide.

2. In the program you created in Exercise 1, try leaving the fields uninitialized and print out their values. Try the same with a local variable and see what kind of compiler errors you can produce. Becoming familiar with common compiler errors will make it easier to recognize bugs in your code.
Operators

- Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.
- Operators with higher precedence are evaluated before operators with relatively lower precedence.

Operator Precedence

<table>
<thead>
<tr>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>postfix</td>
<td>expr++ expr--</td>
</tr>
<tr>
<td>unary</td>
<td>++expr --expr +expr -expr !</td>
</tr>
<tr>
<td>multiplicative</td>
<td>* / %</td>
</tr>
<tr>
<td>additive</td>
<td>+ -</td>
</tr>
<tr>
<td>shift</td>
<td>&lt;&lt;= &gt;&gt;= &gt;&gt;&gt;</td>
</tr>
<tr>
<td>relational</td>
<td>&lt; &gt; &lt;= &gt;= instanceof</td>
</tr>
<tr>
<td>equality</td>
<td>== !=</td>
</tr>
<tr>
<td>bitwise AND</td>
<td>&amp;</td>
</tr>
<tr>
<td>bitwise exclusive OR</td>
<td>^</td>
</tr>
<tr>
<td>bitwise inclusive OR</td>
<td></td>
</tr>
<tr>
<td>logical AND</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>logical OR</td>
<td></td>
</tr>
<tr>
<td>ternary</td>
<td>? :</td>
</tr>
<tr>
<td>assignment</td>
<td>= += -= *= /= %= &amp;= ^=</td>
</tr>
</tbody>
</table>
Questions

1. Consider the following code:
   arrayOfInts[j] > arrayOfInts[j+1]
Which operators does the code contain?
2. Consider the following code:
   int i = 10;
   int n = i++%5;
What are the values of i and n after the code is executed?
What are the final values of i and n if instead of using the postfix increment operator (i++), you use the prefix version (++i))?
3. To invert the value of a boolean, which operator would you use?
4. Which operator is used to compare two values, = or == ?
5. Explain the following code sample:
   result = someCondition ? value1 : value2;
Questions

Change the following program to use compound assignments:

class ArithmeticDemo {
    public static void main (String[] args){
        int result = 1 + 2; // result is now 3
        System.out.println(result);
        result = result - 1; // result is now 2
        System.out.println(result);
        result = result * 2; // result is now 4
        System.out.println(result);
        result = result / 2; // result is now 2
        System.out.println(result);
        result = result + 8; // result is now 10
        result = result % 7; // result is now 3
        System.out.println(result);
    }
}

Questions

In the following program, explain why the value "6" is printed twice in a row:

class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        System.out.println(i); // "4"
        ++i;
        System.out.println(i); // "5"
        System.out.println(++i); // "6"
        System.out.println(i++); // "6"
        System.out.println(i); // "7"
    }
}
Control flow statements

• Statements inside your source files are executed from top to bottom, in the order that they appear
• Control flow statements break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code

```plaintext
if(exp) {
    // exp is true
}
else {
    // exp is false
}
```
Control flow statements

switch(exp) {
    case a:
        ...
        break;
    case b:
        ...
        break;
        ...
    default:
        ...
        break;
}

Control flow statements

while (expression) {
    statement(s)
}

do {
    statement(s)
} while (expression);
Control flow statements

for (initialization; termination; increment) {
    statement(s)
}

• The initialization expression initializes the loop; it’s executed once, as the loop begins.
• When the termination expression evaluates to false, the loop terminates.
• The increment expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

Control flow statements

• break:
  – Terminates the innermost switch, for, while, or do-while statement

• continue:
  – Skips the current iteration of a for, while, or do-while loop
Consider the following code:
if (aNumber >= 0)
if (aNumber == 0) System.out.println("first string");
else System.out.println("second string");
System.out.println("third string");

What output do you think the code will produce if aNumber is 3?

Write a test program containing the previous code snippet; make aNumber 3.

What is the output of the program? Is it what you predicted? Explain why the output is what it is; in other words, what is the control flow for the code snippet?

Using only spaces and line breaks, reformat the code snippet to make the control flow easier to understand.

Use braces, { and }, to further clarify the code.