Lecture Set 2: Starting Java

1. Java Concepts
2. Java Programming Basics
3. User output
4. Variables and types
5. Expressions
6. User input

This Course: Intro to Procedural Programming using Java

Why Java?
- Popular modern language
- Used in web, business, telecom applications
- Developed in 1990s, incorporates many features from earlier languages
  - Object-orientation
  - Garbage collection
  - Portability of object code

Portability of Object Code?
- Object code is 2GL (assembly) / 1GL (machine code)
- Last time we said that 2GL / 1GL is architecture-specific
- How can Java have portable object code?
  Answer: Java Virtual Machine (JVM)
Java Virtual Machine

- Java includes definition of Java bytecode – “fake” machine code for Java
- Java compilers produce Java bytecode
- To run Java bytecode, must have bytecode interpreter (“Java Virtual Machine”) on client machine

Facts about JVMs

- For efficiency, JVMs often compile bytecode into native machine code
- There are also “native” Java compilers (these compile Java directly to machine code)

Method Headers

- main is a method = “operation”
  - Operations require operands – data to work on
  - Operations return new data (result)
  - Header gives information on form of operands, result for methods
    For main:
    - Operand is collection of Strings
    - Result is “void” (= unimportant)
    - More later on “public”, “static”
- Every program must have exactly one “main” method (where execution begins)
Output and Comments

- Output to console
  - `System.out.println`
  - `System.out.print`
  - String Literals always use “quotation marks”
- Comments: explanations added by programmer
  - ignored by the compiler
  - read by other people looking at the code
  - Two styles
    - `/* ... */`
    - `// to end of line…`
  - Comments are essential for good programming!

Objects

- Bundles of data (“instance variables”) and methods (“functions”)
- Created using classes as “templates”
- We’ll learn more later this semester

Java Program Organization

- Class
  - Structure around which all Java programs are based
  - A typical Java program consists of many classes
  - Each class resides in its own file, whose name is based on the class’s name
  - The class is delimited by curly braces `{ ... }`.
  - File name: `Example1.java`:
    ```java
    public class Example1 {  
      // (contents of the class go here)  
    }
    ```
  - A class consist of data (variables) and operations (methods)
Holding and calculating values

- variables
  - declaration
  - initialization
  - assignment
  - value use
- mathematical expressions
  - calculated to take on a value
  - based on values of literals and variables

Java Program Organization

- Methods
  - Where most computation takes place
  - Each method has a name, a list of arguments enclosed in (...), and body (collection of statements) in {...}
    ```java
    public static void main(String[] args) {
        ... [contents of the main method go here] ...
    }
    ```
- Variables
  - Storage locations that program can operate on
  - Variables can store data of different forms (integers, for example)
    ```java
    int secondsPerMinute = 60;
    int minutesPerLecture = 50;
    ```

Java Program Organization

- Statements: Many different types
  - Declarations – specify variable types (and optionally initialize)
    ```java
    int x, y, z; // three integer variables
    String s = "Howdy"; // a character string variable
    boolean isValid = true; // a boolean (true/false) variable
    ```
  - Assignments – assign variables new values
    ```java
    x = 13;
    ```
  - Method invocation – call other methods
    ```java
    System.out.println( "Print this message" );
    ```
  - Control flow – determine the order of statement execution.
    (These include if-then-else, while, do-while, for. More later.)
  - Built-in Operators: For manipulating values (+, -, *, /, etc.)
    - i.e. String Concatenation for output
### Built-in (Primitive) Types

<table>
<thead>
<tr>
<th>Type name</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>char</td>
<td>2</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### String Type
- Elements of String type are sequences of characters
  - “abc” “Call me Ishmael” etc.
- **String** type is not built-in
- We will use it a lot
- Useful operation:
  - concatenation (+)
  - “abc” + “def” is equivalent to “abcdef”

### Writing Programs in Java
1. **EXPRESSIONS**: computations that carry a value
2. **OPERATORS**: symbols like +, *, -, etc.
3. Statements end with a semicolon
4. Types of statements:
   - DECLARATION (where a variable is created)
   - ASSIGNMENT (where a variable is given a value)
   - METHOD INVOCATIONS (where another method is called)
   - others - later
5. You can put blank lines in almost anytime you want
   - except not in the middle of an identifier or a keyword
   - and except not in a set of quotation marks
6. Proper indenting helps readability
Variables …

- … are named storage locations

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>5</td>
</tr>
</tbody>
</table>

- Recall that memory is a sequence of bits
- Question: How much memory to allocate for a variable’s value?
- Answer: A variable must have a type specifying how much storage to allocate.

Recall Java Built-in Types

<table>
<thead>
<tr>
<th>Type name</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integers</strong></td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td><strong>Reals</strong></td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>2</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

Primitive Data Types In Detail

**Integer Types:**

- byte 1 byte Range: -128 to +127
- short 2 bytes Range: -32,000 to +32,000
- int 4 bytes Range: -2 billion to +2 billion
- long 8 bytes Range: -9 quintillion to +9 quintillion

**Floating-Point Types:**

- float 4 bytes -3.4x10^38 to 3.4x10^38, 7 digits of precision
- double 8 bytes -1.7x10^308 to 1.7x10^308, 15 digits of precision

**Other types:**

- boolean 1 byte true, false
- char 2 bytes A single (Unicode) character
**Primitive-Type Literals/Constants**

- **Constants** are also called *literals*.
- **Integer types:**
  - `byte`
  - `short`
  - `int`
  - `long`

  - Optional sign and digits (0-9): `-1` `+123` `0` `1234567`
  - Same as above, but followed by 'L' or 'l': `-1394382953L`
- **Floating-point types:**
  - `double`
  - `float`

  - Two allowable forms:
    - Decimal notation: `3.14159` `-234.421` `0.0042` `-43.0`
    - Scientific notation: `(use E or e for base 10 exponent)` `3.145E5` `3.145 x 10^5` `1834.23e-6` `1834.23 x 10^{-6} = 0.00183423`

*Note: By default, integer constants are `int`, unless 'L'/'l' is used to indicate they are `long`. Floating constants are `double`, unless 'F'/'f' is used to indicate they are `float`.*

**Character and String Constants**

- **Char constants:** Single character in single quotes ('…') including:
  - Letters and digits: 'A', 'B', 'C', …, 'a', 'b', 'c', …, '0', '1', …, '9'
  - Punctuation symbols: '*', '#', '@', '$' (except ' and backslash '\')
  - Escape sequences: (see below)
- **String constants:** 0 or more characters in double quotes ("…")

*Escape sequences:* Allows inclusion of special characters:
- `\"` double quote
- `\n` new-line character (start a new line)
- `\'` single quote
- `\t` tab character
- `\` backslash

*Examples:* `char x = 'i'`; `x` contains a single quote
- String `s1 = "Hi there!"`; `s1` contains "Hi there!"
- String `s2 = "C:\WINDOWS"`; `s2` contains C:\WINDOWS

**Common Numeric Operators**

- **Arithmetic operators:**
  - Unary negation: `–x`
  - Addition/subtraction: `x + y` `x - y`
  - Multiplication/division: `x * y` `x / y`
    - Division between integer types **truncates** to integer: `23/4 → 5`
    - `x` divided by `y` returns the **remainder of** `x` divided by `y`: `23 % 4 → 3`
  - Division with real types yields a real result: `23.0/4.0 → 5.75`

- **Comparison operators:**
  - Equality/inequality: `x = y` `x != y`
  - Less than/greater than: `x < y` `x > y`
  - Less than or equal/greater than or equal: `x <= y` `x >= y`

*These comparison operators return a **boolean value:** true or false.*
Common String Operators

- **String Concatenation**: The `+` operator concatenates (joins) two strings.
  
  ```
  "Go" + "Terps" -> "GoTerps"
  ``

  When a string is concatenated with another type, the other type is first evaluated and converted to its string representation.

  ```
  (4 + 2) + "degrees" -> "32 degrees"
  ```

- **String Comparison**: Strings have special comparison functions.
  
  ```
  s.equals(t) : returns true if s and t have the same characters.
  s.compareTo(t) : compares strings lexicographically (dictionary order)
  ```

  ```
  result < 0 if s precedes t
  result > 0 if s follows t
  result == 0 if s is equal to t
  ```

  ```
  "dilbert".compareTo( "dogbert" ) -> -1 (which is < 0)
  ```

  Both functions are case-sensitive.

User Input in Java

- We’ve done output (System.out); what about input?
- Java 5.0 includes the **Scanner class** feature
  
  - Can use Scanner to create “scanner objects”
  - Scanner objects convert user input into data
  - To use Scanner need to *import* a library:
    ```
    import java.util.Scanner;
    ```

Scanner Class Details

- To create a scanner object:
  ```
  new Scanner( input_source );
  ```
  
  - Input source can be keyboard(System.in), files, etc.
  - Object must be assigned to a variable (e.g. `sc`)
- **Operations**
  ```
  nextBoolean()  
  nextByte()    
  nextDouble()  
  nextFloat()   
  nextInt()     
  nextLong()    
  nextShort()   
  next()        
  nextLine()    
  ```

  ```
  Returns value of indicated type (reports error if type mismatch)
  ```

  ```
  Returns sequence of characters up to next whitespace
  ```

  ```
  Returns sequence of characters up to next carriage return
  ```
Objects

- From Example 5:
  ```java
  Scanner sc = new Scanner(System.in);
  ```
  - `sc` is a variable
  - Its type is `Scanner`
- What’s going on?
  - `Scanner` is a class defined in `java.util.Scanner`
  - `System.in` is a predefined object for keyboard input
  - `new Scanner(System.in)` creates a new object in the `Scanner` class and assigns it to `sc`
- Object?
  - A bundle of data (instance variables) and operations (methods)
  - A class defines both instance variables and methods for objects
  - A class is also a type for objects
- We will learn (much) more about objects later

Debugging Java Programs

- Types of errors
  - "Compile time": caught by Eclipse / Java compiler
    - Syntax errors
    - Disobeys the rules of the language; violates language’s grammar
  - Type errors: misuse of variables
  - "Run time": appear during program execution
    - Semantic errors
    - Obey the rules of the language but does not express them meaning you intended:
      - Division by 0
      - Crash or hang or wrong outputs (because of mistakes in programming)
- Eclipse helps catch compile time errors
  - Red: error
  - Yellow: warning
- Debugging
  - Process of finding and fixing problems
  - To minimize debugging frustration – use "unit" testing
    - Write a small part, thoroughly test it, cycle back