Program Fundamentals

/* HelloWorld.java
 * The classic “Hello, world!” program
 */

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

/* HelloWorld.java … <etc.> */

- /* … */ and // …
- These are comments
- Everything between /* and */ or after // is ignored by the compiler
- They explain the program and its parts to humans
  - You, me, the TA, your teammate, your colleagues, and anyone else that might want to understand your program
class HelloWorld {

• “class” is a java *keyword*
  – keywords are words with special meaning in a programming language
• A *class* is a named collection of
  – data objects, and
  – operations on those data objects
• Note how this matches our design!
  – This is object oriented programming!
• The braces { } surround the things in the class
public static void main (String[] args) {

• main() is a java method
  – A method is a named set of operations within a class
  – The parentheses ( ) follow the name of the method
  – The braces surround the body of the method
  – Program vs. applets
  – Every program has a main( ) function
    • That’s where the program starts executing
System.out.println("Hello, world!");

• This is the body of the main( ) method
• This is the instructions that run when main( ) is executed
• This code prints something on the screen
  – Whatever is between the quotes “ ”
Compiling and Running

- Compiler translates human-readable code into machine-readable code
- The name of the .java file usually matches the name of the class it contains
- Java bytecode is machine independent
  - machine code, binaries, and executables are not
Lexical Elements

- Composed of characters
- The lowest-level components of a program
  - White space
  - Comments
  - Keywords
  - Identifiers
  - Literals
  - Operators
  - Punctuation

Thrown out by the compiler

Converted into tokens by the compiler
White space

- Space, tab, and newline
- Separate tokens not otherwise separated by punctuation
- Make the code readable
- Can’t appear in a keyword, identifier, or literal
- Otherwise ignored by the compiler
Comments

• Provide additional information to a person reading the code
• Separates tokens like white space
• Single-line comment: // …
• Multi-line comment: /* … */
• Ignored by the compiler
• Important part of any good program!
Keywords (aka Reserved words)

- Special words that can’t be used for anything else: *abstract, boolean, byte, case, catch, char, class, const, continue, default, do, double, else, extends, final, finally, float, for, goto, if, implements, import, instanceof, int, interface, long, native, new, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, try, void, volatile, while*

- *null, true, false* – predefined like literals
Identifiers

• Names for different elements of a java program: classes, methods, and variables
• Defined by the programmer
• Any sequence of letters and digits starting with a letter (including $ and _)
  – Except Java keywords and null, true, and false
• Examples
  – Ok: HelloWorld, println, data, first_name, a7, java
  – Not ok: 123, x+y, int, data?, first name
Literals

• **Constants** – primitive program elements with a fixed value
  
  • Five types of constants in java
  
  – int – 1, 79, -23, 0
  
  – double – 1.5, 2.7, 3.14159, -0.3
  
  – boolean – true, false
  
  
  – String – “Hello”, “foo”, “123”, “123(*&T^%”
Operators and Punctuation

- **Operators** specify an action to take on data
  - +, -, *, /, %, ++, --, etc.
  - Really just shorthand for specific methods on that data
- **Punctuation** separates or encloses program elements or parts
  - ; , ( ) { }
- **Type, Precedence, and Associativity**
- By the way: ., !, *, #, $, &, ^, @, ~, |, /, ->
Data Types and Variable Declarations

• Every data object has an associated type that specifies
  – What it is
  – What operations it supports

• Primitive types
  – Numeric: byte, short, int, long, float, double – numbers in different sizes and formats
  – Character: char - characters
  – Logical: boolean – true, or false
  – Can be created using literals or as the result of operations (17, 2+3, etc.)
Data Types and Variable Declarations (cont.)

• Class types
  – String, Button, Point, etc.
  – Composed of other class types and primitive types
  – Created with the class keyword
  – Over 1500 classes in standard Java
Variables

• Data objects
  – Have a specified type
  – Have a value of that type

• Variable declaration
  `<type> <identifier>;
  <type> <identifier1>, <identifier2>,
  <identifiern>;
`
Variable Initialization

• Examples

  int age;
  boolean flag1;
  double hang_time; // C style identifier
  String firstname;
  Button clickToExit;  // Java style identifier
  int first, second, third;
// HelloWorld2.java - simple variable declarations

class HelloWorld2 {
    public static void main(String[] args) {
        String word1, word2, sentence;

        word1 = "Hello, ";
        word2 = "world!";
        sentence = word1.concat(word2);
        System.out.println(sentence);
    }
}

strings vs Strings vs. Identifiers vs. Variables

- *string* – a particular data value that a program can manipulate
- *String* – a Java type - data objects of this type can contain strings
- *Variable* – a data object, has an identifier, a type, and a value
- *Identifier* – the name of a particular class, variable, or method
- Example: String animal = “elephant”;

// StringVsId.java – contrast Strings & Identifiers

class StringVsId {
    public static void main(String[] args) {
        String hello = "Hello, world!";
        String stringVary;
        stringVary = hello;
        System.out.println(stringVary);
        stringVary = "hello";
        System.out.println(stringVary);
    }
}

User Input

• Most interesting programs get input from the user
• Lots of ways to do this
• For now we will use tio (terminal I/O)
  – If this doesn’t work, see
    http://www.cse.ucsc.edu/~charlie/java/tio/doc/install.html
// SimpleInput.java-read numbers from the keyboard
import tio.*; // use the package tio

class SimpleInput {
    public static void main (String[] args) {
        int width, height, area;
        System.out.println("type two integers for" +
                " the width and height of a box");
        width = Console.in.readInt();
        height = Console.in.readInt();
        area = width * height;
        System.out.println("The area is ");
        System.out.println(area);
    }
}
Calling Predefined Methods

• A *method* is a named group of instructions
  – We’ve seen `main()`, `System.out.println()`,
• We execute a method by *calling* it
  – We call a method by putting its name in the program where we want it to be executed
• Method names don’t have to be unique
  – Identified by the object name - `System.out.println()`
• *function* is another name for method
Passing Parameters to Methods

• Many methods take inputs: parameters
• Parameters are passed to the method by placing them between the parentheses
• Example: System.out.println(“Hello”);
  – “Hello” is the parameter passed to System.out.println( )
• Multiple parameters are separated by commas
print( ) and println( )

• System.out.print( ) and System.out.println( ) print out strings and the primitive types
• Difference: println( ) puts a newline at the end
• Explicit newline is represented by ‘\n’, as in System.out.print("Hi\nScott\n");
  – Same as System.out.println("Hi");
More on print( ) and println( )

• Concatenation with ‘+’
  – ‘+’ allows multiple things in a print( ) statement
  – System.out.print(“The value is: ” + value);

• Be careful with numeric types
  – Given int a = 5, b = 7;
  – System.out.println(“The value is: ” + a + b); prints out “The value is: 57”
  – System.out.println(“The value is: ” + (a+b)); prints out “The value is 12”
  – System.out.println(a + b); prints out “12”
Number Types

• Two basic representations for numbers
  – Integer: whole numbers
  – Floating point: fractional numbers and very big numbers

• Bit
  – The smallest element of storage in a computer
  – Can be either 0 or 1
  – Bigger numbers are stored as a sequence of bits
Representing Numbers with Bits

• A sequence of bits is interpreted as a binary number
  – 00, 01, 10, 11 binary = 0,1,2,3 in decimal
  – Read Appendix A

• A byte is 8 bits
  – Smallest addressable unit in a computer
  – Can contain any number between –128 and 127
## Integer Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Number of Bits</th>
<th>Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>-32768 to 32767</td>
</tr>
<tr>
<td>char</td>
<td>16</td>
<td>0 to 65536</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>-2147483648 to 2147483647</td>
</tr>
<tr>
<td>long</td>
<td>64</td>
<td>9223372036854775808 to 9223372036854775807</td>
</tr>
</tbody>
</table>
## Floating point types

<table>
<thead>
<tr>
<th>Type</th>
<th>Number of bits</th>
<th>Approximate Range of Values</th>
<th>Approximate Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>32</td>
<td>+/-10^{-45} to +/-10^{+38}</td>
<td>7 decimal digits</td>
</tr>
<tr>
<td>double</td>
<td>64</td>
<td>+/-10^{-324} to +/-10^{+308}</td>
<td>15 decimal digits</td>
</tr>
</tbody>
</table>
Char

• char is a special integer type
  – Holds numeric values that represent Unicode characters
  – Examples:

<table>
<thead>
<tr>
<th>‘a’</th>
<th>‘b’</th>
<th>‘c’</th>
<th>‘A’</th>
<th>‘B’</th>
<th>‘C’</th>
<th>‘0’</th>
<th>‘1’</th>
<th>‘9’</th>
<th>‘&amp;’</th>
<th>‘*’</th>
<th>‘+’</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>98</td>
<td>99</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>48</td>
<td>49</td>
<td>57</td>
<td>38</td>
<td>42</td>
<td>43</td>
</tr>
</tbody>
</table>

• Special characters
  • ‘\’, ‘\b’, ‘\r’, ‘\”’, ‘\f’, ‘\t’, ‘\n’, ‘\’’, ‘‘’
Numeric Literals

• Integer literals
  – Default to type int
  – Can be specified as long by ending with ‘L’
  – 24, 1003, 123887699888L
  – Octal: begin with ‘0’, as in 0217
  – Hex: begin with “0x” as in 0xB3D

• Floating point literals
  – Default to type double
  – Can be specified as float with ‘F’
  – 3.7, 2.9, 3.1416F, 1358494.34792098
Numbers vs. Chars vs. Strings

- The number 49 is different from the char 49 is different from the string “49”
- int or literal 49 = 32 bit binary number
  00000000000000000000000000110001 binary
- char 49 = 16 bit binary number and represents the Unicode character ‘1’
  0000000000110001 binary
- String “49” = ‘4’ + ‘9’ + 0 = 52 57 0 =
  0000000001010010000000000001010111
  0000000000000000 binary
Arithmetic Expressions

• Operators:
  – Addition: +
  – Subtraction: -
  – Multiplication: *
  – Division: /
  – Modulus (remainder): %

• Types: char, byte, short, int, long, float, double
Rules of Mixed-Mode Arithmetic

1. An arithmetic operation on objects of the same type yields a result of that type

2. An arithmetic operation on objects of different types first promotes smaller types to larger type
   - Any operand is a double $\Rightarrow$ promoted to double
   - Otherwise, any float $\Rightarrow$ promoted to float
   - Otherwise, any long $\Rightarrow$ promoted to long
   - Otherwise, any int $\Rightarrow$ promoted to int
   - And then rule 1 applies
Details

• Any result value that is too big for the result type will be undefined
  – Solution: force promotion when necessary by using a variable or literal of the larger type or by casting one operand to a suitably larger type
  – Example: (float)5, (long)a, 5.0, 4f

• Integer types storing floating point values will have the fractional part truncated
  – towards 0
/ MakeChange.java - change in dimes and pennies
import tio.*; // use the package tio

class MakeChange {
    public static void main (String[] args) {
        int price, change, dimes, pennies;
        System.out.println("type price (0:100):");
        price = Console.in.readInt();
        change = 100 - price; //how much change
        dimes = change / 10; //number of dimes
        pennies = change % 10; //number of pennies
        System.out.print("The change is : ");
        System.out.println(dimes + " dimes, " + pennies + " pennies");
    }
}
Type Conversion

• Implicit (in mixed-mode arithmetic)
• Explicit (casting)
• Widening
  – From “smaller” to “larger type”
  – All information is retained
• Narrowing
  – From “larger” to “smaller” type
  – Information may be lost
  – Result may be meaningless
• Common mistake: int z = 3.0/4.0;
Assignment Operators

- `<variable> = <rightHandSide>;`
- **What happens?**
  - Right hand side is evaluated
  - The result is placed in the variable `<variable>`
- **Examples:**
  
  ```
  a = 0;
  a = b + c;
  a = Console.in.readInt();
  a = b = c;
  ```
More Assignment Operators

- = , += , -= , *= , /= , %= , >>= , <<= , &= , ^= , |=

- += is pronounced “plus equals”, etc.
- All but ‘=‘ are shorthand
- Example:
  
a += b; is shorthand for  a = a + b;
- The others work the same way
Increment and Decrement Operators

• ++ is shorthand for “add one to the variable”
  
  i++; and ++i; are shorthand for i = i + 1;

• -- is shorthand for “subtract one from the variable”
  
  i--; and --i; are shorthand for i = i - 1;

• Location determines order of evaluation
  
  int a, b=0;
  a = ++b;  // result: a = 1 and b = 1;
  a = b++;  // result: a = 0 and b = 1;
Order of Evaluation

• In expressions with multiple operators, order matters!

• Example:
  
  \[
  j = (3 * 4) + 5; \quad // \text{result: } j = 17
  \]
  
  \[
  j = 3 * (4 + 5); \quad // \text{result: } j = 27
  \]
Precedence and Associativity

• *Precedence* specifies which operators are evaluated first

• *Associativity* specifies order when operators have equal precedence

• Parentheses ( ) override these
  – They force whatever is inside to be evaluated as a unit

• Example:
  \[ x = 3 + 4 \times 5; \quad \text{// result: } x = 23 \]
  \[ x = (3 + 4) \times 5; \quad \text{// result: } x = 35 \]

• Look at Appendix B for details
Programming Style

• Comments
  – At the top of every file
  – At the top of every class definition
  – At the top of every method definition
  – At the top of every non-trivial block of instructions

• Identifiers should be short and meaningful

• Readability, understandability, clarity, elegance
Java Naming Conventions

1. Class names start with uppercase and embedded words are capitalized, e.g. `HelloWorld`

2. Methods and variables start with lowercase and embedded words are capitalized, e.g. `readInt`, `data`, `toString`, `loopIndex`

3. $ should not be used and _ marks you as an old C programmer