Basic Computation (Savitch, Chapter 2)

TOPICS

• Variables and Data Types
• Expressions and Operators
• Integers and Real Numbers
• Characters and Strings
• Input and Output

Variables

• Variables store information
  • You can think of them like boxes
  • They “hold” values
  • The value of a variable is its current contents

• Note that this differs from variable in math
  • In math, a variable is an “unknown”
    • It has a fixed value (or set of values)
    • Solving an equation reveals its value
    • They don’t actually “vary”
  • In programming, variables change values
    • Their current value is always known
    • The program changes their values to achieve a goal

Data Types

• Variables are like boxes: they hold values
  • But you can’t put an elephant in a shoe box
  • Different boxes hold different types of things

• Therefore, variables have data types
  • The data type describes the set of values a variable might contain
  • The value of a variable is a member of the set defined by its data type
  • Examples: int, char, String

Creating Variables

• You create new variables through declarations
  • Examples:
    int daysPerYear;
    char vowel;

• You assign values using =
  • Examples:
    daysPerYear = 365;
    vowel = ‘a’;
More about Variables

- An uninitialized variable is useless
  - So it’s good practice to initialize when declaring variables, can be done with one statement:
    ```java
    int daysPerYear = 365;
    ```
- Variables can be re-used:
  ```java
  int daysPerYear = 365;
  // random code here
  daysPerYear = 110;
  ```

Literals

- Literals are values that are directly recognized by Java:
  - numbers
    - 237, 10, 9, 1.5, 5.8, 99.999
  - characters
    - ‘a’, ‘Z’, ‘0’, ‘$’
  - strings
    - "hello", "there"

Java Identifiers

- An identifier is a name, such as the name of a variable.
- Identifiers may contain only
  - Letters
  - Digits (0 through 9)
  - The underscore character (_)
  - And the dollar sign symbol ($) which has a special meaning
- The first character cannot be a digit.

- Identifiers may not contain any spaces, dots (.), asterisks (*), or other characters:
  - 7-11 netscape.com util.* (not allowed)
- Identifiers can be arbitrarily long.
- Since Java is case sensitive, stuff, Stuff, and STUFF are different identifiers.
Keywords or Reserved Words

- Words such as if are called keywords or reserved words and have special, predefined meanings.
- Cannot be used as identifiers.
- See Appendix 1 for a complete list of Java keywords.
- Example keywords: `int`, `public`, `class`

Naming Conventions

- Class types begin with an uppercase letter (e.g. String).
- Primitive types begin with a lowercase letter (e.g. int).
- Variables of both class and primitive types begin with a lowercase letters (e.g. myName, myBalance).
- Multiword names are "punctuated" using uppercase letters.

Where to Declare Variables

- Declare a variable
  - Just before it is used or
  - At the beginning of the section of your program that is enclosed in {}:
    ```java
    public static void main(String[] args) {
        // declare variables here */
        // code starts here */
    }
    ```

Java Types

- In Java, there are two different types of data types:
  - Primitive data types
    - Hold a single, indivisible piece of data
    - Pre-defined by the language
    - Examples: int, char, double, boolean
  - Classes
    - Hold complex combinations of data
    - Programs may define new classes
    - Examples: String, System

Let's start with these
**Primitive Types**

- Integer types: byte, short, int, and long
  - int is most common
- Floating-point types: float and double
  - double is more common
- Character type: char
- Boolean type: boolean

---

**Primitive Data Types**

- The 4 most common primitive data types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integer values</td>
<td>5</td>
</tr>
<tr>
<td>double</td>
<td>floating-point values</td>
<td>3.14</td>
</tr>
<tr>
<td>char</td>
<td>characters</td>
<td>‘J’</td>
</tr>
<tr>
<td>boolean</td>
<td>either true or false</td>
<td>true</td>
</tr>
</tbody>
</table>

Notice – there are no quotes around true!

---

**Assignment Statements**

- An assignment statement is used to assign a value to a variable.
  
  ```
  answer = 42;
  ```

- The "equal sign" is called the assignment operator.

- We say, "The variable named answer is assigned a value of 42," or more simply, "answer is assigned 42."
Operators

- Operators act on primitive data types
- You have already seen =
  - No, it does not test for equality
  - The ‘=’ operator assigns a value to a variable
    - Example: int x = 7;
- The other operators are more intuitive:
  - ‘+’ adds two numbers
  - ‘*’ multiplies two numbers
  - ‘-’ subtracts two numbers
  - ‘/’ divides two numbers
  - ‘==’ tests for equality

Expressions

- A program is a sequence of expressions
  - Well, it also needs a header
  - But the program body lists expressions
- A simple expression looks like:
  \[ \text{data_type var1} = \text{var2 op var3}; \]
  - Where ‘var1’, ‘var2’ and ‘var3’ are variables
  - ‘op’ is any operator (e.g. +, -, *)
  - If var1 is a new variable, then ‘data_type’ is the type of the new value

Variations on Expressions

- Note that variables can be re-used across expressions:
  \[
  \text{int } x = 7; \\
  x = x + 1; \\
  \]
- Variables can be re-used within expressions:
  \[
  x = x + x; \\
  \]

More variations on expressions

- The right hand side of an assignment can be any mathematical expression:
  \[
  \text{int } y = x + (2 \times z); \\
  \]
- When more than one operator appears
  - Parenthesis disambiguate
    - See above
  - Without parenthesis, operator precedence rules apply
    - E.g. multiply before add, left before right
    - Better to rely on parentheses
Example Problem

- How would you write a program to print all the numbers from 1 to 100, one per line?
- You could write 100 println(...) commands, but that would be a long program!
- Or could use one variable and keep incrementing it...

Integers

- Numbers without fractional parts
  3, 47, -12
- Variables store integers with an assignment statement
  int size = 7;
- Integer variables may be used like integer literals (i.e., number), e.g.,
  size = size + 1;

Example

```java
// Simple example of loop in Java
public class SimpleExample {
    public static void main(String[] args) {
        int counter = 1;
        while (counter <= 100) {
            System.out.println(counter);
            counter = counter + 1;
        }
    }
}
```

Integer Arithmetic Operations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Example</th>
<th>Evaluates to</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>45 + 5</td>
<td>50</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>657 – 57</td>
<td>600</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>7000 * 3</td>
<td>21000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>13 / 7</td>
<td>1</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>13 % 7</td>
<td>6</td>
</tr>
</tbody>
</table>
Remainder Reminder

- % determines what is left after integer division.
- For integers, $x \% y$ or $x / y$
  
  $$x = qy + r$$

  where $r = x \% y$ and $q = x / y$

Java Integer Examples

```java
int i = 10/3;
- What's $i$? 3

int j = 10 % 3;
- What's $j$? 1

int k = 13 % 5 / 2;
- What's $k$? 1
```

Additional Integer Operators

- Self-assignment
  ```java
  int temperature = 32;
  temperature = temperature + 10;
  ```

  What is temperature? 42

- Increment
  ```java
  cent++; equivalent to cent = cent + 1;
  ```

- Decrement
  ```java
  cent--; equivalent to cent = cent - 1;
  ```

Specialized Assignment Operators

- Assignment operators can be combined with arithmetic operators including -, *, /, %.
  ```java
  amount = amount + 5;
  ```

  can be written as
  ```java
  amount += 5;
  ```

  yielding the same results.
Parentheses and Precedence

- Parentheses can communicate the order in which arithmetic operations are performed.
- Examples:
  - \((\text{cost} + \text{tax}) \times \text{discount}\)
  - \(\text{cost} + (\text{tax} \times \text{discount})\)
- Without parentheses, an expression is evaluated according to the rules of precedence.

Precedence Rules

- The binary arithmetic operators *, /, and %, have lower precedence than the unary operators +, -, ++, --, and !, but have higher precedence than the binary arithmetic operators + and -.
- When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.

Sample Expressions

<table>
<thead>
<tr>
<th>Ordinary Math</th>
<th>Java (Preferred Form)</th>
<th>Java (Parenthesized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{rate} \times \text{delta})</td>
<td>rate * rate + delta</td>
<td>(rate * rate) + delta</td>
</tr>
<tr>
<td>2*(salary + bonus)</td>
<td>2 * (salary + bonus)</td>
<td>2 * (salary + bonus)</td>
</tr>
<tr>
<td>(\text{time} / \text{mass})</td>
<td>1 / (time + 3 * mass)</td>
<td>1 / (time + (3 * mass))</td>
</tr>
<tr>
<td>(\text{a} / \text{b})</td>
<td>(a - t) / (t + 9 * v)</td>
<td>(a - t) / (t + (9 * v))</td>
</tr>
</tbody>
</table>
Real Numbers

- Also called floating-point numbers
- Numbers with fractional parts
  3.14159, 7.12, 9.0, 0.5e001, -16.3e+002

- Declared using the data type `double`
  ```java
double pricePerPound = 3.99,
taxRate = 0.05,
shippingCost = 5.55;
double pctProfit = 12.997;
```

double Arithmetic Operations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>45.0 + 5.30</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>657.0 – 5.7</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>70.0 * 3.0</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>96.0 / 2.0</td>
</tr>
</tbody>
</table>

Numbers in Java

- `int` is of fixed size;
  a value that is too large to be stored in an `int` variable will not match the mathematical value.
- Example:
  ```java
  int x = 100000 * 100000;
  out.println( x );
  ```
  Will print: 1410065408
  - Numerical overflow!
  - No warning messages!

Numbers in Java

- It is not always possible to test `double` expressions for equality and obtain a correct result because of rounding errors (called “floating point error”).

```java
public class ProblemDoublePrecision {
    public static void main( String[ ] args ) {
        double val = 1.0/5 + 1.0/5 + 1.0/5 - 0.6;
        System.out.println( val );
    }
}
```
Numbers in Java

- How should you handle “floating point error”?
  - Test to see if the value is within a margin of error

```java
public class CheckDoubleEquality {
    public static void main(String[] args) {
        double val = 1.0/5 + 1.0/5 + 1.0/5 - 0.6;
        if (Math.abs(val) < 0.00001) {
            val = 0;
        }
        System.out.println(val);
    }
}
```

Assignment Compatibilities

- Java is said to be strongly typed.
  - You can’t, for example, assign a floating point value to a variable declared to store an integer.
  - Sometimes conversions between numbers are possible ...
    ```java
double doubleVariable = 7;
```
  - ... is possible even if `doubleVariable` is of type double, for example.

Assignment Compatibilities

- A value of one type can be assigned to a variable of a type further to the right:
  - `byte --> short --> int --> long --> float --> double`
  - But not to a variable of any type further to the left.
  - You can assign a value of type `char` to a variable of type `int`.

Type Casting

- A type cast temporarily changes the value of a variable from the declared type to some other type.
  - For example,
    ```java
double distance;
distance = 9.0;
int points;
points = (int)distance;
```
  - Illegal without `(int)`
Type Casting

- The value of (int)distance is 9.
- The value of distance, both before and after the cast, is 9.0.
- Any nonzero value to the right of the decimal point is truncated rather than rounded.

Mixing Numeric Data Types

- **Widening conversion.** Java will automatically convert `int` expressions to `double` values without loss of information
  - int i = 5;
  - double x = i + 10.5;
  - double y = i;

- **Narrowing conversion.** To convert `double` expressions to `int` requires a typecasting operation and truncation will occur
  - i = (int) (10.3 * 2)
  - To round-up instead of truncating add 0.5
  - i = (int) (10.3 * x + 0.5)

Characters

- Any key you type on the keyboard generates a character which may or may not be displayed on the screen (e.g., nonprinting characters)
- Characters are a primitive type in Java and are not equivalent to Strings
- Examples
  - char vitamin = ‘A’,
  - chromosome = ‘y’,
  - middleInitial = ‘N’;
The other meta-type: Classes

- A **primitive** data type is indivisible
  - They have no meaningful subparts
  - The primitives are defined by the language
    - int, char, double, etc.
- A **class** is a data type that contains many bits of information
  - For example, Strings (many primitive chars)
  - Many classes defined by the language
    - You can also define new ones...

Classes

- Classes have data & methods
  - The data may be primitives or other classes.
  - Used instead of operators
  - The period (\'.\') accesses methods of a class:
    ```
    String greeting = "hello";
    char c = greeting.charAt(0);
    // c now equals 'h'
    ```

More about Strings

- **String** is defined in the **java.lang** package
  - *The java.lang package is automatically included in all programs, so you do not need to import it.*
- String literals are defined in double-quotes “string”
- **Examples**
  ```
  String t1 = "To be ";
  String t2 = "or not to be";
  System.out.println(t1.concat(t2));
  // prints To be or not to be
  ```

String Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int length()</td>
<td>Returns the length of this string</td>
</tr>
<tr>
<td>int indexOf(String s)</td>
<td>Returns the index within the string of the first occurrence of the string s.</td>
</tr>
<tr>
<td>String substring (int begin, int endx)</td>
<td>Returns the substring beginning at index beginx and ending at index endx-1</td>
</tr>
<tr>
<td>String toUpperCase()</td>
<td>Converts all characters of the string to uppercase</td>
</tr>
<tr>
<td>String concat(String s)</td>
<td>Concatenates the new string to the end of the original string</td>
</tr>
<tr>
<td>char charAt(int index)</td>
<td>Returns the character at the index, which must be between 0 and length of string - 1</td>
</tr>
</tbody>
</table>
**Syntax: primitives vs classes**

- Operators act on primitive variables
  - Examples: +, -, *, %
  - Standard math in-fix notation
    - \( x + y \);
    - \( y / 7 \);
- Methods act on class variables
  - Example: length()
  - Notation: class.method(arguments)
    - String s1 = “foo”;
    - int x = s1.length();

**String Method Examples**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // Keyboard input examples
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Hello, World!");
        System.out.println(keyboard.next());
        System.out.println(keyboard.nextLine());
        // String methods
        String string = "Hello in a text string";
        System.out.println(string.substring(1)); // prints "ello in a text string"
        System.out.println(string.length()); // prints 18
        System.out.println(string.replace(" ", ""); // prints 6
        System.out.println(string.toUpperCase()); // prints 6
    }
}
```

**Object Examples**

- **Scanner** instance is an object (not primitive)
- Methods for the **Scanner** class include
  - nextInt \( \rightarrow \) returns next sequence as integer
  - nextDouble \( \rightarrow \) returns next sequence as double
  - next \( \rightarrow \) returns next sequence of chars
    - read until the next whitespace (spaces, tabs, end of line)
  - nextLine \( \rightarrow \) returns next line up until enter key
    - reads

**Input/Output**

- From util package
  - import java.util.Scanner;
- Create a new instance:
  - Scanner in = new Scanner(System.in);
- Input (depends on data type reading in)
  - String input = in.nextInt();
  - String line = in.nextLine();
  - int intVal = in.nextInt();
  - double dblVal = in.nextDouble();
Reading Integers

import java.util.*;

public class getInput
{
    public static void main( String[] args )
    {
        Scanner in;
        int intVal;
        in = new Scanner( System.in );
        System.out.println("Enter an integer: ");
        intVal = in.nextInt();
        System.out.println( intVal );
    }
}

Reading double Numbers

import java.util.*;

public class getDoubleInput
{
    public static void main( String[] args )
    {
        Scanner in;
        double temp;
        in = new Scanner( System.in );
        System.out.println("Enter a real number: ");
        temp = in.nextDouble();
    }
}

Reading Strings

import java.util.*;

public class getStringInput
{
    public static void main( String[] args )
    {
        Scanner in;
        String name;
        in = new Scanner( System.in );
        System.out.println("Enter your name: ");
        name = in.next();
        System.out.println( name );
    }
}

Printing Integers

public class Forecast
{
    public static void main (String args[])
    {
        System.out.print("The temperature will be ");
        System.out.print(-10);
        System.out.print(" degrees…");
        System.out.println(" and that’s cold, folks!");
    }
}
Formatting Decimal Values

- **Use DecimalFormat class**
  - Leading zeros (e.g. money: $0.25)
    - "0.##"
  - Trailing zeros (e.g. money: $5.30)
    - ".00"
  - Truncate to 3 decimal values
    - ".###"
  - Add comma for thousands
    - ",###"

- **Import package**
  
  ```java
  import java.text.*;
  ```

- **Create the object**
  
  ```java
  DecimalFormat fmt = new DecimalFormat("#.##");
  ```

- **Specify which numbers to format when printing by calling format method**
  
  ```java
  System.out.println( fmt.format( 45.6789 ) );
  System.out.println( fmt.format( 345.6 ) );
  System.out.println( fmt.format( 67.0 ) );
  DecimalFormat fmt2 = new DecimalFormat("000.00");
  System.out.println( fmt2.format( 45.6789 ) );
  System.out.println( fmt2.format( 5.6 ) );
  DecimalFormat fmt6 = new DecimalFormat("#.###");
  System.out.println( fmt6.format( 12345 ) );
  ```

- **What could go wrong?**
  - If you mis-type a variable name or a data type...
  - When you try to compile & run it in Eclipse
    - Eclipse will tell you there was an error
    - The editor will put a red ‘X’ at the left of the line with the error
  - This is an example of a compile-time error
What else could go wrong?

- You can specify an illegal operation
  - E.g. try to divide a string by a string
  - Again, a compile-time error with a red ‘x’

- You can forget a ; or a }
  - Same as above

More Errors

- Capitalization errors
  - Java is case sensitive, identifier names must use the same capitalization rules each time

- Logic Errors
  - Program appears to run correctly, but on closer inspection the wrong output is displayed

Debugging Hints

- Let Eclipse help you!
  - Gives suggestions on methods to use
  - Provides warning and error messages as you type… even provides suggestions of how to fix the problem.

- Add debugging statements to check the computation
  System.out.println(...);