Chapter 2 Elementary Programming

Contents

[Algorithm 2](#_Toc42105794)

[Pseudocode 2](#_Toc42105795)

[Code 2](#_Toc42105796)

[Tracing a program 3](#_Toc42105797)

[Variables 3](#_Toc42105798)

[Declaring Variables 3](#_Toc42105799)

[Assigning Variables 4](#_Toc42105800)

[Initialize Variables / Assignment Operator 4](#_Toc42105801)

[Assignment Statements: 4](#_Toc42105802)

[Identifiers 4](#_Toc42105803)

[Naming Conventions 5](#_Toc42105804)

[Camel Back 5](#_Toc42105805)

[Underscore 5](#_Toc42105806)

[Named Constants = final keyword 6](#_Toc42105807)

[Numeric Literals 6](#_Toc42105808)

[Data Type 6](#_Toc42105809)

[Primitive Data Types 7](#_Toc42105810)

[Reference Data Types 7](#_Toc42105811)

[Numeric Data Types 7](#_Toc42105812)

[Scope 8](#_Toc42105813)

[Input 8](#_Toc42105814)

[Reading using the Scanner class 9](#_Toc42105815)

[Operations 10](#_Toc42105816)

[Exponents 12](#_Toc42105817)

[Evaluating Expressions and Operator Precedence 12](#_Toc42105818)

[Augmented Assignment Operators 13](#_Toc42105819)

[Increment and Decrement Operators 13](#_Toc42105820)

[Numeric Type Conversions 14](#_Toc42105821)

[Conversion Rules 14](#_Toc42105822)

[Software Development Process 15](#_Toc42105823)

[Common Errors & Pitfalls 15](#_Toc42105824)

[Undeclared/Uninitialized Variables and Unused Variables 15](#_Toc42105825)

[Integer Overflow 15](#_Toc42105826)

[Round-off Errors – Rounding Error 15](#_Toc42105827)

[Unintended Integer Division 15](#_Toc42105828)

[Pitfalls 16](#_Toc42105829)

# Algorithm

Algorithm: describes how a problem is solved by listing the actions that need to be taken and the order of their execution

* Help the programmer plan a program before writing it in a programming language
* Can be described in natural languages or in pseudocode
* Example:
  1. Read in the circle’s radius
  2. Compute the area using the following formula:

Area = radius \* radius \* π

* 1. Display the result

# Pseudocode

A mix of natural language and program language.

# Code

Code: write a program; translating the algorithm/pseudocode into a programming language.

* Translate an algorithm into a program
* Example:

public class ComputeArea{

public static void main(String[] args){

double radius; // declare a radius

double area; // declare an area

// circle’s radius

radius = 20; // the variable, radius, is assigned a value of 20

// Compute the area using the following formula:

area = radius \* radius \* 3.14159; // the variable, area, is assigned the value of the result from the mathematical expression

// Display the result

System.out.println(“The area for the circle of radius” + radius + “ is “ + area);

}// end main

}// end ComputeArea

Tracing a program: reviewing how a program works

* Helpful for understanding how programs work
* Helpful for finding errors
* You do this by evaluating your code line by line – keep a running chart of the memory/ the values that are stored in memory
  + This helps with logic errors and to double check your work for validity. You want your programs to be valid and to keep the integrity of the variables.

# Variables

Variables: represents a value stored in the computer’s memory

* The value can be changed in the program
* Represent data of a certain data type
* The variable must be declared before it can be assigned a value
* When possible, declare and assign its initial value in one statement
* Datatype variableName;
* Examples:
  + int count;
  + char letter;

## Declaring Variables

Declaring variables: telling the computer (compiler) what the variable is & allocates memory space for the variable based on its data type

* Example: int number;

## Assigning Variables

Assigning a variable is give that variable a value that is not the default value of that programming language.

* Example: number = 102;

Initialize Variables / Assignment Operator: = *assigning a value to a variable*

* when you use Java, you will **not** think of the = as the equal sign, you will **now** think of it as the **assignment** sign
* variable = expression;
* Example: int y = 1;

Assignment Statements: designates a value for a variable

* Is an expression in Java – also called assignment expression
* After a variable is declared, you can assign a value to it by using an assignment statement.
* Use the assignment operator =

You can use a variable in an expression: x = x + 1;

System.out.println(x = 1); is equivalent to x = 1;

System.out.println(x);

i =j = k = 1; is equivalent to k = 1;

j = k;

i = j;

Identifiers: are the names that identify the elements such as classes, methods, and variables in a program.

* Names of variables and methods that appear in the program
* Use descriptive names, not ambiguous or non-informative names
  + Examples:
    - double rad; // not descriptive – what does it stand for? For radius? For radical? For “wow! That was rad/cool”?
    - double radius; // we know that is the value of the radius for a circle
* **Must obey the following rules**:
  + An identifier is a sequence of characters that consists of letter, digits, underscores (\_), and dollar signs ($).
  + An identifier must start with a letter, an underscore (\_), or a dollar sign ($). It cannot start with a digit.
  + *The $ by convention, is used only in mechanically generated source code*
    - Meaning – we cannot use the $ for our variables = off limits
  + An identifier cannot be a reserved word.
  + And identifier cannot be true, false, or null.
  + An identifier can be of any length.

Java is case sensitive!

Class variables start with upper case letters.

All other variables start with lower case letters.

Naming Conventions: Java naming conventions

* Choose descriptive names with straightforward meanings for the variables, constants, classes, and methods in your programs
* Names are case sensitive because Java is case sensitive
* Use lowercase for variables and methods
* If a name consists of several words, concatenate them into one word
  + Make the first word lowercase, then capitalize the first letter of the second word
* Capitalize the first letter of each word in a class
* Capitalize every letter in a constant, and use underscores between words

Do not use reserved words, class names, methods, … that are already used in the Java language.

There are 2 styles to dealing with spaces: 1) camel back notation, 2) underscore notation.

Camel Back: Each word is butted together, the first character of each word after the first word is capitalized.

* Example: to name a class Program One: ProgramOne
* Examples: to name a variable first number: firstNumber

Underscore: Using the underscore character to represent the spaces

* Example: to name a class Program One: Program\_One
* Example: to name a variable first number: first\_number

I suggest trying out both ways and seeing which one you feel comfortable with, then stick to that one way. Don’t mix the two styles, it doesn’t look good, not professional.

FYI: I use the camel back notation, so when you look at my examples that is the style you will see.

# Named Constants = final keyword

Named Constants: is an identifier that represents a permanent value

* The value cannot change during the execution of a program
* Also called a constant
* Must be declared and initialized (assigned a value) in the same statement.
* The identifier must be written in all capital letters! ONLY capital letters – no lower case characters at all.
  + If you use camel back notation, you will have to use the underscore notation to represent the spaces for constants.
* Benefits:
  + You do not have to repeatedly type the same value if it is used multiple times
  + If you have to change the constant value (from 3.14 to 3.14159 for PI), you need to change it only in a single location in the source code – the spot where you declared and assigned it.
  + A descriptive name for a constant makes the program easy to read
* The word final is a Java keyword for declaring a constant
* Example: final double PI = 3.14159;

# Numeric Literals

* A literal is a value that appears directly in a program
* Has to fit within the variable data type
  + Example byte b = 128; will cause a compile error because the value 128 is too large for a byte

Data Type: the kind of data stored in a variable

Java is strongly data typed! If you declare a variable to be in int and try to assign it the value of 1.4, it will not compile. Java does this to keep the integrity of your variables.

Primitive Data Types: integers, floating-point (numbers with decimal point), characters, and boolean

* These values are what is stored in main memory

Reference Data Types: Strings, Arrays, ArrayLists, objects = ADTs (abstract data types that you will learn about in 202/2302 – for now, we don’t have to worry about them)

# Numeric Data Types

* Java has 6 numeric types for integers and floating-point number
* Operators are: +, -, \*, /, %

|  |  |  |
| --- | --- | --- |
| **Type Name** | **Range** | **Storage Size** |
| byte | -27 to 27-1 (-128 to 127) | 8-bit signed |
| short  Integer  types | -215 to 215-1 (-32768 to 32767) | 16-bit signed |
| int | -231 to 231-1 (-2147483648 to 2,147,483,647) | 32-bit signed |
| long | -263 to 263-1 (-9223372036854775808 to 9223372036854775807) | 64-bit signed |
| float  Floating-  point  types | Negative range: -3.4028235E+38 to -1.4E-45  Positive range: 1.4E-45 to 3.4028235E+38 | 32-bit IEEE 754 |
| double | Negative range: -1.7976931348623157E+308 to -4.9E-324  Positive range: 4.9E-32 to 1.7976931348623157E+308 | 64-bit IEEE 754 |

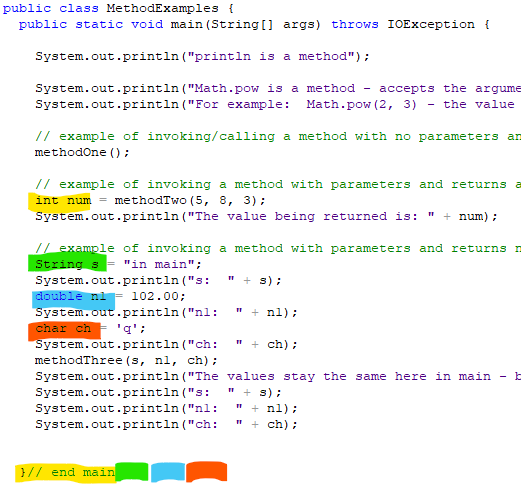
* int & double are the standard
* be careful of overflows, Java does not report nor warn of overflows
* calculations involving floating-point numbers are approximated because these numbers are **not** stored with complete accuracy
  + meaning the variables with double data types are not accurate!
* int **are** stored with complete accuracy

# Scope

Scope: the scope of a variable is the part of the program where the variable is referenced; where the program has access to that variable.

* Scope of a variable is from where it is declared in a block to the closing } of that block

Examples:



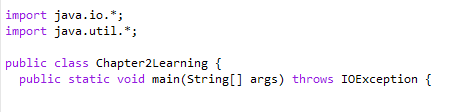
num’s scope is from where it is declared to the closing } of main

Same with s, n1, and ch

Input: receiving information from the user or from a file

* Java has a predefined object that will do this for us, this object is the Scanner object
* to be able to receive information/data from the keyboard – the user – then we need an object/Scanner assigned to read in the keystrokes
  + Example: Scanner input = new Scanner(System.in);
* In order to use the Scanner object, you have to have that class imported into your program; this is accomplished with the following statement, located prior to the class statement: 

Like so:



FYI: I always have these two statements at the beginning of all my programs. The util class has lots of “goodies” that are used often. I am not one who likes to piece meal a job, I like to get things done. So, I keep this statement, that way, whenever I need anything from the util class, the JVM will get is and I do not have to worry about nitpicking every little thing I need.

The io class is used when a program performs reading (i = input) and writing (o = output) to file, not just to the screen. We will use this starting chapter 5, for now, we will just have it there waiting.

* to be able to receive information/data from a file, then we need an object/Scanner assigned to read the file values
  + Example:

Scanner inputFromFile = new Scanner(new File(“data.txt”));

Reading using the Scanner class (an object that reads in data)

To be able to get those values – the input – then we need methods that will do this for us.

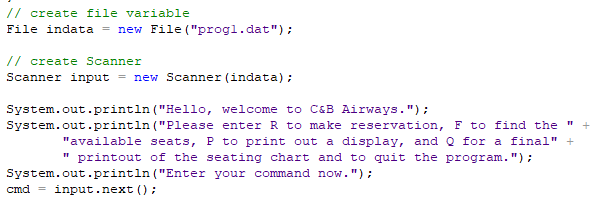
Remember that Java is strongly data typed, so there is a method for each data type

* from the keyboard
  + Scanner input = new Scanner(System.in);
  + Methods for Scanner Objects:

|  |  |
| --- | --- |
| ***Method*** | ***Description*** |
| nextByte() | Reads an integer of the **byte** type |
| nextShort() | Reads an integer of the **short** type |
| nextInt() | Reads an integer of the **int** type |
| nextLong() | Reads an integer of the **long** type |
| nextFloat() | Reads an integer of the **float** type |
| nextDouble() | Reads an integer of the **double** type |
| next() | Reads a string that ends before a whitespace character |
| nextLine() | Reads a line of text (i.e. a string ending with the *Enter* key pressed) |

If you need to get data/values from the user, you need to inform the user. Inform the user by telling them what is going on and what to enter in. (Sometimes when we are creating programs that have user input, we’ll put the input into a file to eliminate having to enter all the data that is necessary throughout the program. )

Example:

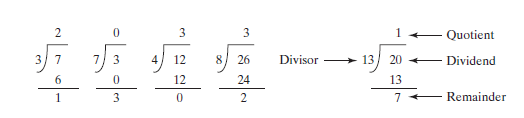


The next method will read in one single word, it terminates receiving input when the user enters a white space (such as the space bar).

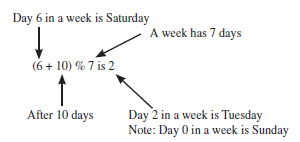
# Operations

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Meaning** | **Example** | **Result** |
| + | Addition | 34 + 1 | 35 |
| - | Subtraction | 34.0 – 0.1 | 33.9 |
| \* | Multiplication | 300 \* 30 | 9000 |
| / | Division | 1.0 / 2.0 | 0.5 |
| % | Remainder – called modulo  Abbreviated called the mod | 20 % 3 | 2 |

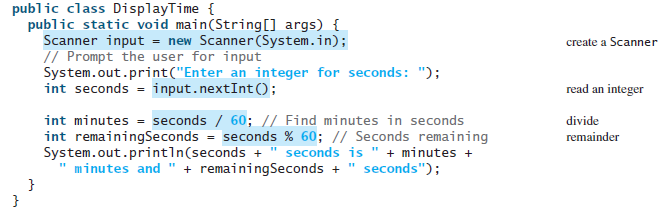
* When both operands are integers, the result is also an integer, the fractional part is truncated.
  + Example: 5 / 2 = 2 *not 2.5*
* When one operand is a double and the result is being assigned to a double (or printed to the screen), then you will get the decimal part.
  + Example 5.0 / 2 = 2.5
* Modulo operator (mod) yields the **remainder** after the division
  + Example: 20 % 13 = 7
  + This property can determine odd & even numbers, if numbers are divisible by a certain number, the day of the week, if we have to round up when order (like ordering a box of doughnuts: 15 % 12 = 3; then we have to order 2 boxes), etc.
  + We know how to add, to subtract, to multiply, to divide, it is the mod that we aren’t familiar with. In reality, we are. It is just been too long since we learned about remainders. So, let us refresh our minds. Do you remember when you were first learning about division? When you divided 7 by 3: you would write 2 r. 1 as the answer. Other examples:



* Another example: getting the days in the week:



* Another example: getting time:



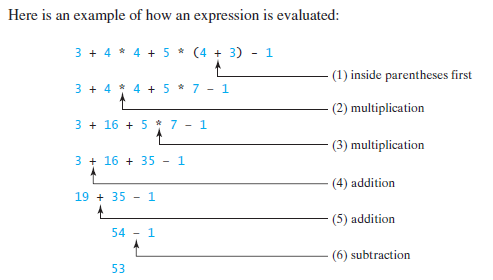
* Unary operator has only 1 operand
  + Example: -5 means negative five
* Binary operator has 2 operands
  + Example: 4 – 5 means subtracting 5 from 4

# Exponents

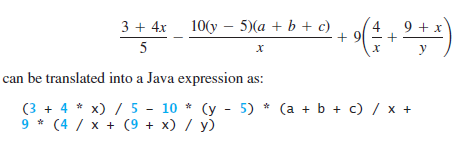
* Math.pow(a, b) is ab
* Example: 23 = Math.pow(2,3) = 8
  + FYI: pow is a method in the Math class

# Evaluating Expressions and Operator Precedence

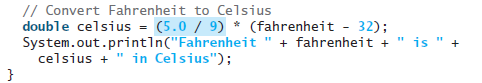
* Java expressions are evaluated in the same way as arithmetic expressions.



* Pay attention when you are converting arithmetic expressions. For example, 9(4 + 5), would be in a Java statement: 9 \* (4 + 5). Forgetting the \* sign would be a wrong expression – not a valid expression.
* Example:



* + Notice that in Java, the multiplication sign is the \* not x,if you use the x then you do not have a Java expression.
* Don’t forget about int verses double division! 5 / 9 = 0, 5.0 / 9 = .5555555555556. If you forget about this then some expressions would be invalid. For example, converting Fahrenheit to Celsius, you cannot use 5 / 9, you have to use doubles to get a valid result:



If you forget the making one of the values for 5 / 9 as a double, then you will always get 0 for the answer, making your expression invalid as the output would be incorrect. Hence, your program is incorrect and nobody wants to purchase incorrect/worthless programs; which then means no job for you.

# Augmented Assignment Operators

* Augmentation is taking the current value of a variable, modify it, and then assign back to the same variable
* It is a shortcut that programmers like to use. You do not have to be a pro at this right away, but practice it until you get comfortable, we all like using shortcuts and you will see it often.
* Example: count = count + 1; can be count +=1;

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Equivalent** |
| += | Addition assignment | i += 8 | i = i + 8 |
| -= | Subtraction assignment | i -= 8 | i = i - 8 |
| \*= | Multiplication assignment | i \*= 8 | i = i \* 8 |
| /= | Division assignment | i /= 8 | i = i / 8 |
| %= | Remainder assignment | i %= 8 | i = i % 8 |

# Increment and Decrement Operators

* To increment or decrement a variable by 1
* Get to be a pro at this. It is used often.
  + Pay attention to the placement of the operator. If it is before, that means to modify the current value then use it. If it placed after, that means to use it as the current value is, then modify the value.
* Shorthand operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** i = 1 |
| ++var | preincrement | Increments by 1, & then uses the incremented var in the statement | int j = ++i;  // j is 2, i is 2 |
| var++ | postincrement | Uses the original value in the statement, then increments by 1 | int j = i++;  // j is 1, i is 2 |
| --var | predecrement | Decrements by 1, & then uses the decremented var in the statement | int j = --i;  // j is 0, i is 0 |
| var-- | postdecrement | Uses the original value in the statement, then decrements by 1 | int j = i--;  // j is 1, i is 0 |

# Numeric Type Conversions

* You can modify data types from one kind of a data type to another kind of data type.
  + For example you can make an int to a double. Or a double to an int.
* Some of these conversions will happen automatically, some you have to specify that to happen.
* Casting: an operation that converts a value of one data type into a value of another data type
* Java automatically converts integers to floating-point, or a smaller type to a larger type this is called *widening a type*; also known as implicit casting
  + Example: double number = 102;
    - What gets stored in main memory is the value of 102.0
* It does not do automatic conversions for larger data types to be smaller data types
* In order for this to happen, the programmer (you) has to *type cast* by explicit casting
* Floating-point numbers can be converted into integers using explicit casting this is called *narrowing a type*; also known as explicit casting
  + Example: System.out.println((int)1.7); will display a 1
  + Example: System.out.println( (double) 1 / 2 ); will display .5

# Conversion Rules

* When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1.    If one of the operands is double, the other is converted into double.

2.    If one of the operands is float, the other is converted into float.

3.    If one of the operands is long, the other is converted into long.

4.    If both operands are int, the result will be an int.

# Software Development Process

* The software development life cycle is a multi-stage process that includes requirements specification, analysis, design, implementation, testing, deployment, and maintenance.

# Common Errors & Pitfalls

Undeclared/Uninitialized Variables and Unused Variables

double interestRate = 0.05;

double interest = interestrate \* 45;

wrong: interestRate is assigned & declared a value but interestrate has not been declared & assigned any value

Integer Overflow

When a variable is assigned a value that is too large in size to be stored as the specified data type, it causes overflow.

int value = 2147483647 + 1; will be stored as -2147483648

Round-off Errors – Rounding Error

The difference between the calculated approximation of a number and its exact mathematical value.

See penny program example

Unintended Integer Division

Integer division results in integer number – not a decimal number (the fractional part is truncated)

To force 2 integers to perform floating-number division, make one be a floating-point number (cast)

Pitfalls:

Writing code to create multiple input objects for each input – creating a Scanner for each reading for each object to be read

Not prompting a user to enter the input