Introduction to Java Basics

Programming in Java

Contents

9- Arithmetic Expressions: Let’s put all together

- Purpose:
  - Conversion of degrees Fahrenheit in degrees Celsius
- Algorithm:
  - Assign the temperature in Fahrenheit (ex. 100 degrees)
  - Calculate the temperature in Celsius (1 Celsius = 5/9 (Fahr - 32))
  - Display temperature in Celsius
- Variables and constants:

9- Arithmetic Expressions: Let’s put all together

Data	Identifier	Type	var or const?
Temperature in Fahrenheit	`fahr`	`double`	

9- Arithmetic Expressions: Let's put all together

```
//*********************************************************
//  Temperature.java       Author: your name
//  A program to convert degrees Fahrenheit in degrees Celsius.
//*********************************************************
public class Temperature {
   public static void main (String[] args)
   {
     // Declaration of variables and constants


    // step 1: Assign the temperature in Fahrenheit (100)

    // step2: Calculate the temperature un Celsius

    // step3: Display temperature in Celsius


   }
}
```

filename???

10- More assignment operators

- In addition to = , we often perform an operation on a variable, and then store the result back into that variable
- Java has shortcut assignment operators:

```
variable = variable operator expression;
variable operator= expression;
```

Operator	Example	Equivalent To
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

10- More assignment operators: Shorthand Assignments

Example:	Equivalent To:
count += 2;	count = count + 2;
sum -= discount;	sum = sum – discount;
bonus *= 2;	bonus = bonus * 2;
time /= rushFactor;	time = time / rushFactor;
change %= 100;	change = change % 100;
amount *= count1 + count2;	amount = amount * (count1 + count2);

10- More assignment operators: Assignment operators

- The behavior of some assignment operators depends on the types of the operands
- ex: the +=
  - If the operands are strings, += performs string concatenation
  - The behavior of += is consistent with the behavior of the "regular" +

10- More assignment operators: Assignment operators

```
int amount = 10;
amount += 5;
System.out.println(amount);

double temp = 10;
temp *= 10;
System.out.println(temp);

String word = "hello";
word += "bye";
System.out.println(word);
word *= "bye";  // ???
```

Output

10- More assignment operators: Increment or Decrement

- In Java, we often add one or subtract one to a variable...
- Two shortcut operators:
  - The increment operator (++) adds one to its operand
  - The decrement operator (--) subtracts one from its operand
- The statement: **count++;**

  is functionally equivalent to: **count = count + 1;**
- The statement: **count--;**

  is functionally equivalent to: **count = count - 1;**

10- More assignment operators: Increment or Decrement

The increment and decrement operators can be in:

- in prefix form ex: ++count;
  - the variable is incremented/decremented by 1
  - the value of the entire expression is the new value of the variable (**after** the incrementation/decrementation)
- in postfix form: ex: count++;
  - the variable is incremented/decremented by 1
  - the value of the entire expression is the old value of the variable (**before** the incrementation/decrementation)

10- More assignment operators: Increment or Decrement

```
int nb = 50;
++nb;
```

value of nb

```
int nb = 50;
nb++;
```

value of nb

```
int nb = 50;
int x;
x = ++nb;
```

value of nb & x

```
int nb = 50;
int x;
x = nb++;
```

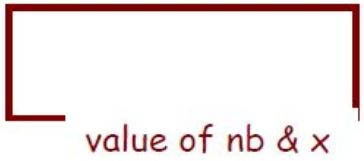
value of nb & x

```
int nb = 50;
int x;
x = nb++ + 10;
```

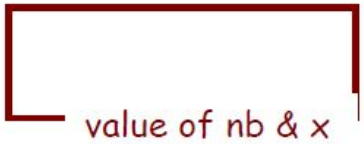
value of nb & x

10- More assignment operators: Increment or Decrement

```
int nb = 50;
int x;
x = ++nb + nb;
```




```
int nb = 50;
int x;
x = nb++ + nb;
```



10- More assignment operators: Exercise

What is stored in the integer variables num1 , num2 and num3 after the following statements?

```
int num1 = 1, num2 = 0;
int num3 = 2 * num1++  +  --num2 * 5;
```

A) num1 = 1, num2 = 0, num3 = 2

B) num1 = 1, num2 = 0, num3 = -1

C) num1 = 2, num2 = -1, num3 = 2

D) num1 = 2, num2 = -1, num3 = -3

E) num1 = 2, num2 = -1, num3 = -1

10- More assignment operators: Exercise

What is stored in the integer ***q*** after the following statements?

```
int x = 1, y = 10, z = 3;

int q = ++x  *  y--  +  z++;
```

A) 13

B) 20

C) 23

D) No idea?

10- More assignment operators: Exercise

What is stored in the integers ***a*** and ***c*** after the following statements?

```
int a = 1;

int c = a++  +  a--;
```

A) a = 1, c = 2

B) a = 1, c = 3

C) a = 3, c = 3

D) No idea?

10- More assignment operators: Exercise

What is stored in the integer ***c*** after the following statements?

```
int a = 1, b = 2;

int c = a++  +  a  +  2 * (--b) + 3 / b--;
```

A) 7

B) 8

C) 8.5

D) No idea?

10- More assignment operators: Summary of ++ and --

Expression	Operation	Value Used in Expression
`count++`	add 1	old value
`++count`	add 1	new value
`count--`	subtract 1	old value
`--count`	subtract 1	new value

11- Assignment Compatibility

- In general, the value of one type cannot be stored in a variable of another type

```
int  intVariable  =  2.99;      //Illegal
```

- However, there are exceptions to this

```
double  doubleVariable  =  2;
```

For example, an int value can be stored in a double type

11- Assignment Compatibility

- An expression has a <u>value</u> and a <u>type</u>

```
2 / 4     (value = 0, type = int)
2 / 4.0   (value = 0.5, type = double)
```

- The type of the expression depends on the type of its operands

- In Java, type conversions can occur in 3 ways:
  - arithmetic promotion
  - assignment conversion
  - casting

11- Assignment Compatibility: Arithmetic promotion

- Happens automatically, if the operands of an expression are of different types

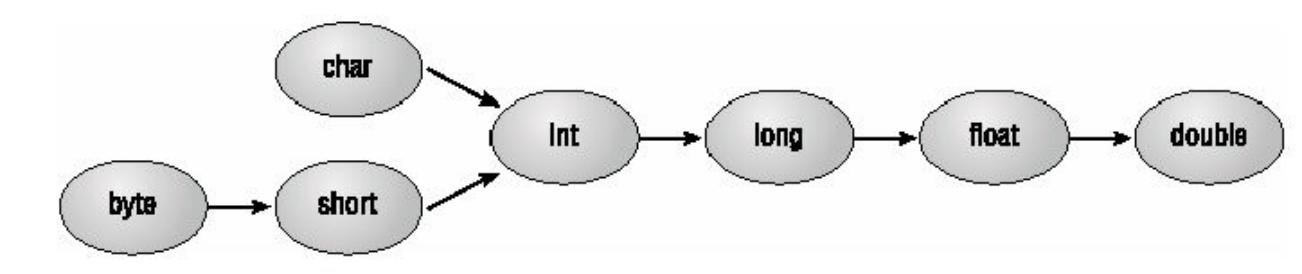
```
aLong + anInt * aDouble
```

- Operands are promoted so that they have the same type
- Promotion rules:
  - if one operand is of type… the others are promoted to…

double	**double**
float	**float (double)**
long	**long**

- **short**, **byte** and **char** are always converted to **int**

11- Assignment Compatibility: Arithmetic promotion




```
(aByte + anotherByte)      --> int
(aLong + anInt * aDouble)  --> ??
(aFloat - aBoolean)        --> ??
```

11- Assignment Compatibility: Arithmetic promotion

- Value and type of these expressions?

```
2 / 4
int / int      2/4
int            0
```

11- Assignment Compatibility: Arithmetic promotion

- What is the value and type of this expression?

2 / 4 * 1.0

A) 0 (int)

B) 0.0 (double)

C) 0.5 (int)

D) 0.5 (double)

11- Assignment Compatibility: Arithmetic promotion

- What is the value and type of this expression?

```
1.0 * 2 / 4
```

A) 0 (int)

B) 0.0 (double)

C) 0.5 (int)

D) 0.5 (double)

11- Assignment Compatibility: Arithmetic conversions

- Occurs when an expression of one type is assigned to a variable of another type

```
var = expression;
```

- *widening* conversion
  - if the variable has a wider type than the expression
  - then, the expression is widened automatically

```
long aVar;
aVar =
5+5;
```

```
byte aByte;
int anInt;
anInt = aByte;
```

```
double aDouble;
int anInt = 10;
aDouble = anInt;
```

  - integral & floating point types are compatible
  - **boolean** are not compatible with any type

11- Assignment Compatibility: Arithmetic conversions

- *narrowing* conversion
  - if the variable has a smaller type than the expression then, compilation error, because possible loss of information

```
int aVar;
aVar = 3.7; ok?
```

```
int aVar;
aVar = 10/4; ok?
```

```
int aVar;
aVar = 10.0/4; ok?
```

11- Assignment Compatibility: Casting

- The programmer can explicitly force a type conversion
- Syntax: **(desired_type) expression_to_convert**

```
int aVar;
aVar = (int)3.7;
(aVar is 3… not 4!)
```

```
byte aByte;
int anInt = 75;
aByte = anInt;              // ok?
aByte = (byte)anInt;        // ok?
```

```
double d;
d = 2/4;  // d is 0
d = (double)2/4;  // d is 0.5
                  // 2.0 / 4
d = (double)(2/4); // d is 0.0
```


- *Casting* can be dangerous! you better know what you're doing…

11- Assignment Compatibility: Exercise

- Which of the following assignment statements are valid?

```
byte b1 = 1, b2 = 127, b3;
b3 = b1 + b2;            // statement a)
b3 = 1 + b2;             // statement b)
b3 = (byte)1 + b2;       // statement c)
```

- A) Statements a), b) and c) are valid
- B) Only statements a) and b) are valid
- C) Only statements b) and c) are valid
- D) Only statements a) and c) are valid
- E) None of the Java statements are valid

12- Strings

- So far we have seen only primitive types
- A variable can be either:
  - a primitive type
    - ex: **int**, **float**, **boolean**,...
  - or a reference to an object
    - ex: **String**, **Array**,...
- A character string:
  - is an object defined by the **String** class
  - delimited by double quotation marks ex: "hello", "a"
    - System.out.print ("hello"); // string of characters
    - System.out.print ('a'); // single character

12- Declaring Strings

- Declare a reference to a String object

```
String title;
```

- Declare the object itself (the String itself)

```
title = new String("content of the string");
```

This calls the String constructor, which is

a special method that sets up the object

12- Declaring Strings

- Because strings are so common, we don't have to use the **new** operator to create a **String** object

```
String title;
title = new String("content of the string");
```

```
String title = new String("content of the string");
```

```
String title;
title = "content of the string";
```

```
String title = "content of the string";
```

- These special syntax works **only** for strings

12- Strings

- Once a string is created, its value cannot be modified (the object is immutable)
  - cannot lengthen/shorten it
  - cannot modify its content
- The String class offers:
  - the + operator (string concatenation)
    - ex: **String solution = "The answer is " + "yes";**
  - many methods to manipulate strings, a string variable calls …
    - **length()** // returns the nb of characters in a string
    - **concat (str)** //returns the concatenation of the string and
    - **toUpperCase()** // returns the string all in uppercase
    - **replace(oldChar, newChar)** // returns a new string

      // where all occurrences of character oldChar have been replaced by character newChar
    - And more...

12- Strings: String indexes start at zero

Display 1.5 String Indexes

The 12 characters in the string `"Java is fun."` have indexes 0 through 11.

0	1	2	3	4	5	6	7	8	9	10	11
J	a	v	a		i	s		f	u	n	.

Notice that the blanks and the period count as characters in the string.

12- Strings: Example (StringTest.java)

```
public class StringTest  {
   public static void main (String[] args)  {
     String string1 = new String ("This is a string");
     String string2 = "";
     String string3, string4, string5;



     System.out.println("Content of string1: \"" + string1 + "\"");
     System.out.println("Length of string1: " + string1.length());
     System.out.println("Content of string2: \"" + string2 + "\"");
     System.out.println("Length of string2: " + string2.length());
```

???

Output

12- Strings: Example (StringTest.java)

```
 // String string1 = new String ("This is a string");
// String string2 = "";

string2 = string1.concat(" hello");
string3 = string2.toUpperCase();
string4 = string3.replace('E', 'X');
string5 = string4.substring(3, 10);

System.out.println(string2);
System.out.println(string3);
System.out.println(string4);
System.out.println(string5);
}  }
```

???

Output

12- Strings: Example (ScannerDemo3.java)

```
//*************************************************************
// Author: W. Savitch
//
//  This program demonstrates how to read String tokens with
//  the Scanner class
//*************************************************************
import java.util.Scanner;
public class ScannerDemo3
{
 public static void main(String[] args) {
 //Let's declare our Scanner object
 Scanner scannerObject = new Scanner(System.in);
 // let's try to read 2 "words" now
 System.out.println("Next enter two words:");
 String word1 = scannerObject.next( );
 String word2 = scannerObject.next( );
 System.out.println("You entered \"" + word1 + "\" and \""
                + word2 + "\"");
```

12- Strings: Example (ScannerDemo3.java)

```
    //To get rid of '\n‘

    String junk = scannerObject.nextLine( );
    // let's try to read an entire line
    System.out.println("Next enter a line of text:");
    String line = scannerObject.nextLine( );
    System.out.println("You entered: \"" + line + "\"");

    // Close Scanner
    scannerObject.close();
   } // end of main()
}// end of class ScannerDemo3
```

12- Strings: A note on nextLine

- nextLine reads the remainder of a line of text starting where the last reading left off
- This can cause problems when combining it with different methods for reading from the keyboard such as nextInt
- ex:

```
Scanner keyboard = new Scanner(System.in);
int n = keyboard.nextInt();
String s1 = keyboard.nextLine();
String s2 = keyboard.nextLine();
```

- input:

```
2
Heads are better than
1 head.
```

- what are the values of n, s1, and s2?

need an extra invocation of **nextLine** to get rid of the end of line character after the 2