SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. Ridha Khedri

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Acknowledgments: Material based on *Java actually: A Comprehensive Primer in Programming* (Chapter 2)

Topics Covered

- To write even simple computer programs, we need some basic programming elements
- Many of these elements are also found in other programming languages
- Understanding them is therefore also useful for learning languages other than Java
- We will focus on basic programming elements provided in Java
- I will try to demonstrate their use in a simple way

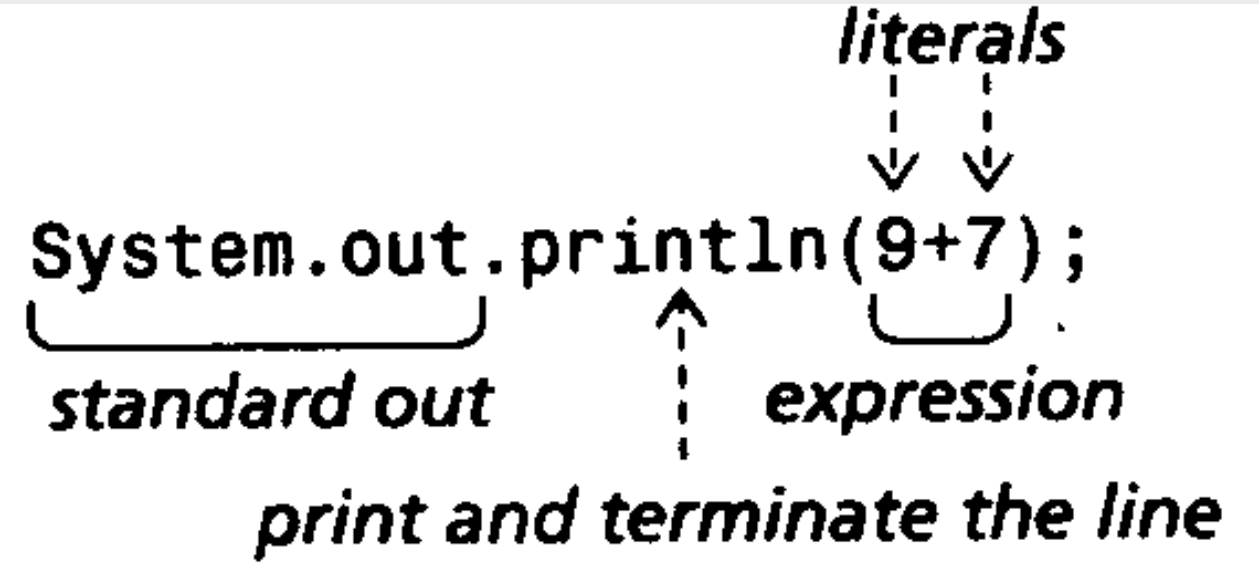
Learning Objectives:

- How to print strings and numerical values to the terminal window
- How to store values in your programs
- What a primitive data type is, and which are Java primitive data types
- How to write arithmetic expressions
- How to format program output
- How to read numbers and strings from the keyboard

- Programs can print directly to the terminal window

Example

Printing the sum of two integers to the terminal window



```
1 // Printing text strings and numerical values to the terminal window.
  public class SimplePrint {
3   public static void main(String[] args) {          // (1)
      System.out.println("The value of 9 + 7 is ");   // (2)
5     System.out.println(9+7);                        // (3)
      System.out.print("The value of 9 + 7 is ");     // (4)
7     System.out.println(9+7);                        // (5)
    }
9 }
```

Listing 1: SimplePrint Program

Program Output

The␣value␣of␣9␣+␣7␣is␣

16

The␣value␣of␣9␣+␣7␣is␣16

```
1 // Printing text strings and numerical values to the terminal window.
  public class SimplePrint {
3   public static void main(String[] args) {            // (1)
    System.out.println("The value of 9 + 7 is ");      // (2)
5   System.out.println(9+7);                           // (3)
    System.out.print("The value of 9 + 7 is ");       // (4)
7   System.out.println(9+7);                           // (5)
  }
9 }
```

Listing 2: SimplePrint Program (showing string spaces)

- It prints strings and numerical values to the terminal window at (2) to (5) using the System.out object

- The object **System.out** is called standard out
- It is by default connected to the terminal window
- The System.out object offers
  - **println()** method: for printing a string and terminating the line (moving the cursor to the beginning of the next line)
  - **print()** method: does not move the cursor to the next line after printing on the current line

- **String literal**, or just a **string**: A sequence of characters enclosed in double quotes (")
- What is a string in general and what are the operations on strings?
- The **operator +** is concatenate on strings
  "High"+"Five" produces "HighFive"
- Concatenation operator + and the print methods help in formatting results
- Here we will only use the string operator + to create the program output to the terminal window

```
1  /*
   ␣It␣illustrates␣how␣we␣can␣use␣the␣+␣operator␣to
3  ␣create␣strings␣that␣can␣be␣printed␣by␣calling␣the␣println␣()␣method
   ␣*/
5
   public␣class␣UseOfConcatenation␣{
7  ␣␣public␣static␣void␣main(String[]␣args)␣{␣␣␣␣␣␣␣␣␣␣//␣(1)
   ␣␣␣␣System.out.println("Multiple␣strings␣can␣be␣printed"␣+␣"␣on␣the␣same
       ␣line");␣␣␣␣//␣(2)
9  ␣␣␣␣System.out.println("We␣can␣also␣print␣a␣number␣together␣with␣this␣
       string,␣e.g.␣"␣+␣2012);␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣(3)
   /*␣We␣can␣also␣write␣the␣above␣line␣as␣follows␣(with␣the␣same␣affect):
11 ␣␣␣␣␣System.out.println("We␣can␣also␣print␣a␣number␣together␣with␣this␣
       string,␣e.g.␣"␣+␣"2012");
   ␣*/
13 ␣␣}
   }
```

Program Output

Multiple strings can be printed on the same line

We can also print a number together with this string, e.g. 2012

- The operator + **converts** the value **2006** to its **string** representation **"2006"**

- **HOWEVER**, using the + operator on numerical values causes the two values to be added

```
System.out.println(9+7)
```

Statement Output

16

- The + operator can be used several times in a statement

```
/*
            THIS ILLUSTRATES SEVERAL USAGES OF THE + OPERATOR
 */

public class UsagesOfPlus {
  public static void main(String[] args) {              // (1)
      System.out.println("**********␣ATTENTION␣STUDENTS␣*******"); //
          (2)
      System.out.println(); // (3)
      System.out.println(); // (4)
      System.out.println("Observe␣how␣the␣different␣symblos␣+␣behave:␣
          The␣value␣of␣9+7␣is␣" + (9 + 7)); // (5)

  }
}
```

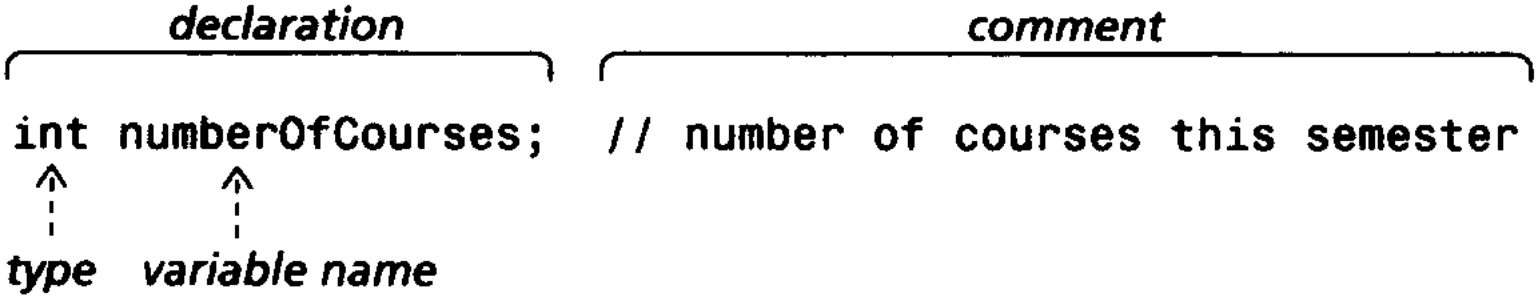
Program Output

*********** ATTENTION STUDENTS *******

Observe how the different symblos + behave: The value of 9+7 is 16

- Programs can evaluate expressions that involve variables
- We often want to store values in computer memory, to retrieve them for use during program execution
- Programming languages use variables for this purpose
- A **variable** designates a location in memory where a value of a certain data type can be stored
- A variable has a name that we use to access the memory location
- A data type (or just type)
  - is defined by a set of valid values
  - has a set of operations that can be performed on those values

Declaring a variable



- The declaration starts with a keyword/terminal **int**
- **int** specifies that the variable stores (only) **integer values**
- The keyword is followed by the variable's name
- The semicolon (;) terminates the declaration

```
1  // Using local variables to hold numerical values.
   public class LocalVariables {
3    public static void main(String[] args) {
       int numberOfCourses = 2; /* Declaration of numberOfCourses and its
            initialization to the integer value 2*/
5      System.out.println("Number of courses this semester is " +
                          numberOfCourses);
7
       int numberOfStudents = 37;     /* Declaration of numberOfStudents and
            its initialization to the integer value 37*/
9      System.out.println("The course Java-101 has " +
                          numberOfStudents + " students");
11
       numberOfStudents = 23;
13     System.out.println("The course Java-102 has " +
                          numberOfStudents + " students");
15   }
   }
```

Program Output

Number of courses this semester is 2

The course Java-101 has 37 students

The course Java-102 has 23 students

- If we need several variables of the same type, we can write them in the same declaration, separated by a comma (,)

```
int numberOfCourses, numberOfStudents;
```

instead of the two following declarations:

```
int numberOfCourses;
int numberOfStudents;
```

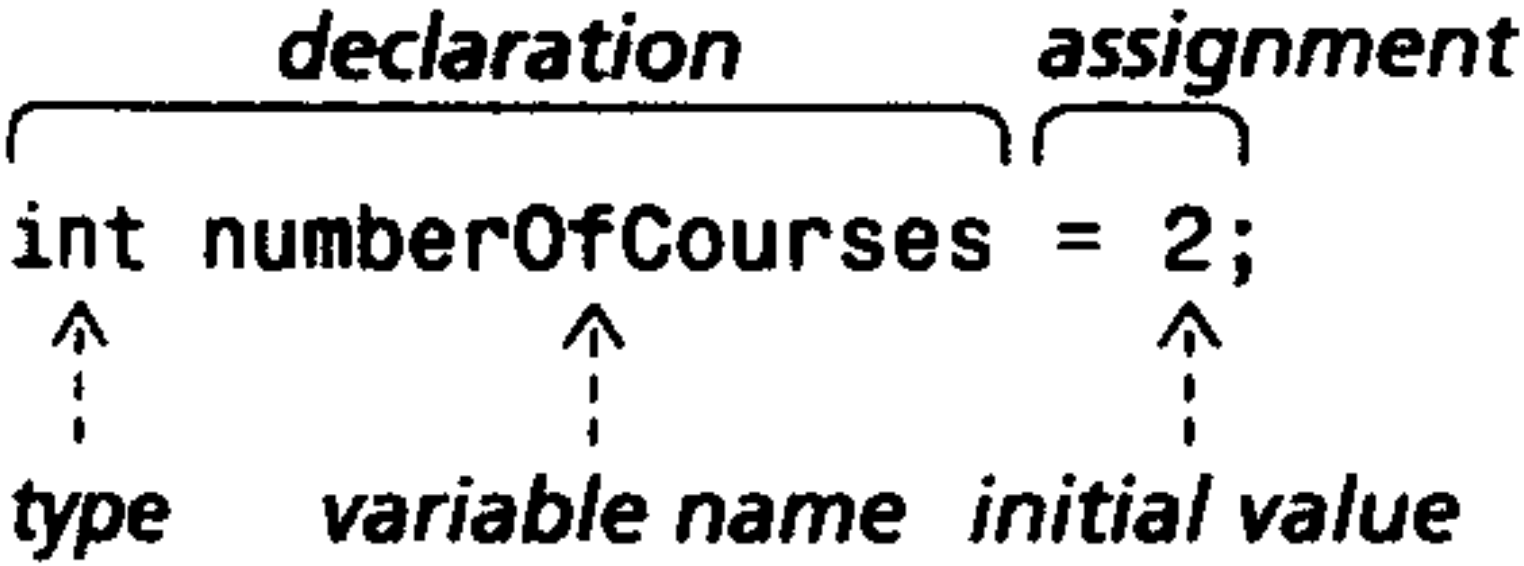
- After declaring a variable, we can assign a value to it
- The first time we assign a value to a variable, we are initializing it
- Later in the program, the value can be changed at will.
- To change the content of the variable, we simply assign a new value to it
- The old value is then overwritten by the new value
- In Java, we use the assignment operator = for assigning

Example

- **int numberOfCourses = 2;**

  can be substituted by the following statements:

  int numberOfCourses;
  numberOfCourses = 2;



```
// Using local variables to hold numerical values.
public class LocalVariables {
  public static void main(String[] args) {
    int numberOfCourses = 2; /* Declaration of numberOfCourses and its
         initialization to the integer value 2*/
    System.out.println("Number of courses this semester is " +
                       numberOfCourses);

    int numberOfStudents = 37;    /* Declaration of numberOfStudents and
          its initialization to the integer value 37*/
    System.out.println("The course Java-101 has " +
                       numberOfStudents + " students");

    numberOfStudents = 23;
    System.out.println("The course Java-102 has " +
                       numberOfStudents + " students");
  }
}
```

```
// Using local variables to hold numerical values.
public class LocalVariablesBetter {
  public static void main(String[] args) {
      // Begin declarations of variables and their initialisations
      int numberOfCourses =2;
      int numberOfStudents = 37;
      // End declarations of variables and their initialisations

      System.out.println("Number of courses this semester is " +
                       numberOfCourses);
      System.out.println("The course Java-101 has " +
                       numberOfStudents + " students");

      numberOfStudents = 23;          /* Assigning a new integer value 23
            to the variable numberOfStudents*/
      System.out.println("The course Java-102 has " +
                        numberOfStudents + " students");
  }
}
```

```
// Calculating and printing the area of a rectangle.
public class Assignment {
  public static void main(String[] args) {
    int length = 5;                                          // (1)
    int breadth = 4;                                         // (2)
    int area = length * breadth;                             // (3)
    System.out.println("The area of a rectangle with length "
                       + length + " and breadth " + breadth
                       + " is " +  area);
    length = 2;                                              // (4)
    breadth = length;                                        // (5)
    area = length * breadth;                                 // (6)
    System.out.println("The area of a rectangle with length "
                       + length + " and breadth " + breadth
                       + " is " +  area);
    length = 6;                                              // (7)
    breadth = 4;                                             // (8)
    area = length * breadth;                                 // (9)
    breadth = 5;                                             // (10)
    System.out.println("The area of a rectangle with length "
                       + length + " and breadth " + breadth
                       + " is " +  area);                    // (11)
  }
}
```

```
// Calculating and printing the area of a rectangle.
public class Assignment {
  public static void main(String[] args) {
    int length = 5;                                          // (1)
    int breadth = 4;                                         // (2)
    int area = length * breadth;                             // (3)
    System.out.println("The area of a rectangle with length "
                       + length + " and breadth " + breadth
                       + " is " +  area);
    length = 2;                                              // (4)
    breadth = length;                                        // (5)
    area = length * breadth;                                 // (6)
    System.out.println("The area of a rectangle with length "
                       + length + " and breadth " + breadth
                       + " is " +  area);
    length = 6;                                              // (7)
    breadth = 4;                                             // (8)
    area = length * breadth;                                 // (9)
    breadth = 5;                                             // (10)
    System.out.println("The area of a rectangle with length "
                       + length + " and breadth " + breadth
                       + " is " +  area);                    // (11)
  }
}
```

Program Output

The area of a rectangle with length 5 and breadth 4 is 20

The area of a rectangle with length 2 and breadth 2 is 4

The area of a rectangle with length 6 and breadth 5 is 24

Tracing the error(s):

	length	breadth	area
After (1)	5		
After (2)	5	4	
After (3)	5	4	20
After (4)	2	4	20
After (5)	2	2	20
After (6)	2	2	4
After (7)	6	2	4
After (8)	6	4	4
After (9)	6	4	24
After (10)	6	5	24

- A **literal** is written directly in the program
- Literals can also be used to define mathematical constants (e.g. pi)
- HOWEVER, if the same literal is used in several places in a program, it is a good idea to define it as a constant
- A **constant** is a variable that cannot change its value after initialization
- In Java, a constant is defined like this:

```
final double INTEREST_RATE = 3.5;
```

- The keyword **final** prefixes the declaration
- It indicates that the variable's value cannot be changed
- Any attempt to change its value in the program leads to the compiler reporting an error and the compilation will be terminated
- To easily distinguish constants from variables, names of constants are usually written with uppercase letters (e.g., INTEREST_RATE)

- When choosing names for variables we need to remember two things
  1. Rules for what Java accepts as valid names
  2. Rules set as conventions for variable names

Example

- Names can contain letters and digits, but cannot start with a digit
  - **int agent007;** — **This is OK**
  - **int 1001nights;** — **This is NOT OK**

Example

- Other Rules for variable names . . .
  - **int my_lucky_number;** **OK: Underscore (_) is allowed**
  - **int _number_drawn;** **OK: Underscore can be the first character**
  - **int numberofminutes;** **OK**
  - **int numberOfminutes;** **OK, but different from the previous (Java is case sensitive)**

Example

Conventions for variable names

- **Use lower-case letters, except for the first letter of each consecutive word**
  - **int size;** **OK**
  - **int Size;** **NOT OK**
  - **int numberOfHours;** **OK**
  - **int itemPrize;** **OK**
  - **int discountedItemPrize;** **OK**
- **Constants are always in upper-case letters, and underscore is used to separate words**

final int DAYS_IN_WEEK = 7, HOURS_IN_WEEK = 168;

- Many programming languages provide primitive data types for numerical values
  - define the range of valid values
  - provide a set of operators to perform calculations on these values
- Java provides six different data types for **integers** and floating-point numbers
- Java provides the data type **char** for values that are single characters

- The primitive data type int in Java can hold values within the following range:

$$-2^{31} \leqslant \text{int values} < +2^{31} - 1$$

$$-2,147,483,648 \leqslant \text{int value} < +2,147,483,648$$

- The language provides the common arithmetic operators

$$+, -, *, /$$

- Integer values are commonly used for counting purposes

- Integer variables are also used to hold values that must be whole numbers

Overflow

```
public class Overflow {
  public static void main(String[] args) {
    int myNumber1 = -2147483648;
    int myNumber2 = +2147483647;

    System.out.printf("The value of myNumber1 is  %d%n", myNumber1);
    System.out.printf("The value of myNumber2 is  %d%n", myNumber2);
    myNumber1 = myNumber1 + (-2);
    myNumber2 = myNumber2 + 2;
    System.out.printf("The NEW value of myNumber1 is  %d%n", myNumber1);
    System.out.printf("The NEW value of myNumber2 is  %d%n", myNumber2);
    myNumber1 = myNumber1 * 2;
    myNumber2 = myNumber2 * 2;
    System.out.printf("After multiplying by 2, the value of myNumber1 is
          %d%n", myNumber1);
    System.out.printf("After multiplying by 2, the value of myNumber2 is
          %d%n", myNumber2);
  }
}
```

Program Output

```
The value of myNumber1 is  -2147483648
The value of myNumber2 is  2147483647
The NEW value of myNumber1 is  2147483646
The NEW value of myNumber2 is  -2147483647
After multiplying by 2, the value of myNumber1 is  -4
After multiplying by 2, the value of myNumber2 is  2
```

Explosion of the rocket Ariane 5 (On June 4, 1996)

On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded.
The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the inertial reference system.
The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.*

int myNumber2 = 2147483648;

Overflow.java:9: ';' expected

- The data type **double** in Java can hold values within the following range:

$$-1.7 \times 10^{308} \leqslant \text{double value} < +1.7 \times 10^{308}$$

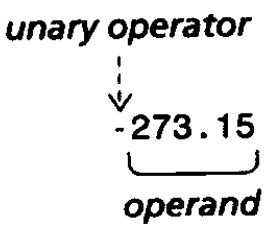
- It provides sufficient range and accuracy for floating-point values in most programming problems

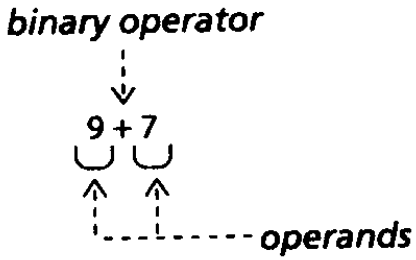
- The usual arithmetic operators on double data types

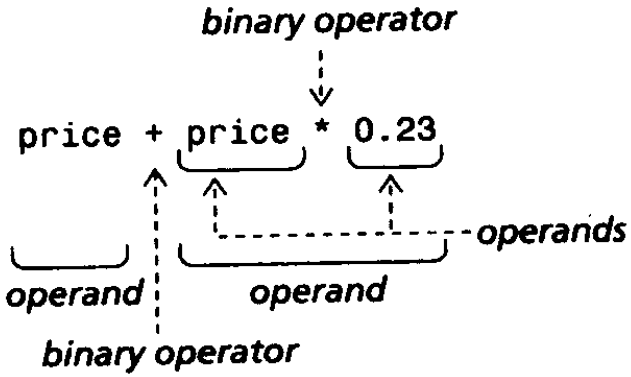
$$+, -, *, /$$

- $+, -, *, /$ are available in all common programming languages
- An operator defines an arithmetic operation
- Each operator accepts one or more operands as arguments
- $+, -, *, /$ all require two operands

Operators and operands in arithmetic expressions:







- The modulus (%) also requires two operands
- The modulus calculates the remainder after division is performed
- Examples: 17 % 3 =?, 7 % 2 =?, 1287 % 329 =?

- If the operands of a binary operator are of the same type, the result is also of the same type as the operands
- If we combine numerical values of different types using a binary operator, the two values will be converted to a common type
- This is called **type conversion**
- Type conversion can be
  - implicitly carried out (implicit conversion)
  - explicitly specified in the source code (explicit conversion)

Implicit conversions

- The range of values for the data type **double** is broader than that of the data type **int**
- We say that double is a broader data type than int
- We say that int is a narrower data type than double
- If we use both integers and floating-point values in an expression, Java automatically converts the integer value to a floating-point value
- In arithmetic expressions with binary operators, Java automatically promotes operand values to the broader type

- Java will automatically perform type conversion if we assign a value of a narrow data type to a variable of a broader data type

Example

- $4 + 8.7$: the integer value 4 will be converted to the floating point value 4.0

- Converting the integer value 10 to the floating-point value 10.0 (of type double)

```
int numberOfFullHours = 10;
double numberOfHours = numberOfFullHours;
```

Explicit conversions

- If we assign a num. value of a broad data type to a variable of a narrower type, we risk loss of information

  Example: if we assign a floating-point value to an integer variable

- To avoid loss of information, Java demands that we explicitly specify that this type of conversion is to be performed

Example

```
double numberOfHours = 40.65;
int numberOfFullHours = (int) numberOfHours;
```

The integer value 40 is assigned to numberOfFullHours

- How to evaluate the following expressions?

$$4 + 5 * 2$$

$$(4 + 5) * 2 \qquad \text{OR} \qquad 4 + (5 * 2)?$$

- Evaluation of the operands of an arithmetic operator in Java is always performed from left to right

- Operator precedence specifies the mutual ranking between different operators

- If two operators with different precedence are next to each other in an expression, the operator with the **higher precedence** is applied first
- For $4 + 5 * 2$ will be interpreted as $4 + (5 * 2)$ during compilation
- It is because the * operator has higher precedence than the $+$ operator
- The parentheses are used to change the order in which operators are applied

$$(4 + 5) * 2$$

Precedence of arithmetic operators:

Precedence level	Type operator	Operator
high	unary	+, -
	binary	*, /, %
low	binary	+, -

Exercise

Evaluate the following expressions. Explain in what order the operators are evaluated in each case.

- $-1 + 2 - 3$
- $2 + 6 * 7 * 2$
- $-5 + 7 - -6$
- $2 + 4.0/5$
- $-2 * 4\%2$

- **ATTENTION:** Integer division always results in an integer value
- 30/4 in a Java program gines 7 NOT 7.5
- **However,** if one of the operands of the division operator / is a floating-point value, a floating-point division is performed
- 30.0/4 in a Java program gines 7.5
- Most other arithmetic operators in Java behave as one would expect them to do

```
1  // Experimenting with the division operator.
   public class Division {
3    public static void main(String[] args) {
       System.out.println("Integer division and modulus:");
5      System.out.println(" 3/2 = " + (3/2));
       System.out.println(" 4/4 = " + (4/4));
7      System.out.println(" 3%2 = " + (3%2));

9      System.out.println("Floating-point division and modulus:");
       System.out.println(" 3.0/2.0 = " + (3.0/2.0));
11     System.out.println(" 3.0/4.0 = " + (3.0/4.0));
       System.out.println(" 3.0%2.0 = " + (3.0%2.0));
13
       System.out.println("Division by zero:");
15     System.out.println(" 2.0/0.0 = " + (2.0/0.0));
       System.out.println(" 0.0/0.0 = " + (0.0/0.0));
17     System.out.println(" 2.0/0 = " + (2.0/0));
       System.out.println(" 2/0 = " + (2/0));
19   }
   }
```

Listing 3: Division Program

Program Output:

```
Integer division and modulus:
 3/2 = 1
 4/4 = 1
 3%2 = 1
Floating-point division and modulus:
 3.0/2.0 = 1.5
 3.0/4.0 = 0.75
 3.0%2.0 = 1.0
Division by zero:
 2.0/0.0 = Infinity
 0.0/0.0 = NaN
 2.0/0 = Infinity
Exception in thread "main" java.lang.ArithmeticException: / by zero
   at Division.main(Division.java:18)
```

- Program output printed to the terminal window can be formatted using the **printf()** method of the System.out object
- **printf(String format, Object ... args)**
- The parameter format specifies how formatting will be done
- T he method **printf()** accepts zero or more parameters

```
/*
 * SimpleDisplay.java
 * To illustrate the usage of the printf() method
 *
 * Created by Ridha Khedri on 12-09-19.
 * Copyright 2012 McMaster University. All rights reserved.
 */
public class SimpleDisplay {
  public static void main(String[] args) {                    // (1)
    System.out.printf("Player | Game        %6d%6d%6d%n", 1, 2, 3);
          // (2)
    System.out.printf("%-20s%6d%6d%6d%n", "F. Reshmann", 320, 160, 235);
           // (3)
    System.out.printf("%-20s%6d%6d%6d%n", "A. King", 1250, 1875, 2500);
            // (4)
  }
}
```

Listing 4: SimpleDisplay Program

Program Output

```
Player | Game            1     2     3
F. Reshmann            320   160   235
A. King                1250  1875  2500
```

- A format string can contain both fixed text and format specifications

```
System.out.printf("Player — Game %6d%6d%6d%n", 1, 2, 3);
```

- The fixed text is printed exactly as specified in the format string
- The format specifications control how the values of the subsequent parameters are formatted and printed
- Formatting of floating-point values is localised
  - Formatting is customised to the character used as the decimal point in a country or a region (i.e., locale)
  - Comma (,) is used in France, Norway, Denmark, etc.
  - Dot (.) is used in Canada, United Kingdom, USA, etc..
  - We will use the dot (.) as decimal point

Format specification in Java (Integer):

Parameter value	Format specification	Example value	String printed	Comment
Integer	"%d"	125	"125"	Occupies as many character places as needed.
	"%6d"	125	" 125"	Occupies six character places and is right-justified. The printed string is padded with spaces to the left.
	"%02d"	3	"03"	Occupies two character places and is padded with leading zeros.

```
/*
 * FormatSpecInteger.java
 * To illustrate the usage of the format specifications with integer
     values
 *
 * Created by Ridha Khedri on 12-09-19.
 * Copyright 2012 McMaster University. All rights reserved.
 */
public class FormatSpecInteger {
  public static void main(String[] args) {
        final int NUMBER_OF_ITEMS = 12345678;                    // (1)
    System.out.printf("%10d%n", NUMBER_OF_ITEMS);      // (2)
    System.out.printf("%d%n", NUMBER_OF_ITEMS);          // (3)
    System.out.printf("%5d%n", NUMBER_OF_ITEMS);       // (4)
    System.out.printf("%1d%n", NUMBER_OF_ITEMS);       // (5)
  }
}
```

Program Output

␣␣12345678
12345678
12345678
12345678

Format specification in Java (Floating point value)

Floating point value	"%f"	16.746	"16.746000"	Occupies as many character places as needed, but always includes six decimal places.
	"%.2f"	16.746	"16.75"	Occupies as many character places as needed, but includes only two decimal places.
	"%8.2f"	16.7466	" 16.75"	Occupies eight character places, including the decimal point, and uses two decimal places.

```
1  public class FormatSpecFloatingPoint {
2    public static void main(String[] args) {
3          final double AVERAGE = 12345678.98766;                    // (1)
4      System.out.printf("%f%n", AVERAGE);                          // (2)
5      System.out.printf("%.3f%n", AVERAGE);                       // (3)
6      System.out.printf("%5.4f%n", AVERAGE);                       // (4)
7      System.out.printf("%20.8f%n", AVERAGE);                      // (5)
8      System.out.printf("%,20.8f%n", AVERAGE);                    // (5)
9    }
10 }
```

Program Output

```
12345678.987660
12345678.988
12345678.9877
␣␣␣12345678.98766000
␣12,345,678.98766000
```

Format specification in Java (String & linefeed)

String	`"%s"`	`"Hi!"`	`"Hi!"`	Occupies as many character places as are needed.
	`"%12s"`	`"Hi Dude!"`	`"    Hi Dude!"`	Occupies twelve character places and is right-justified.
	`"%-12s"`	`"Hi Dude!"`	`"Hi Dude!    "`	Occupies twelve character places and is left-justified.
Linefeed	`"%n"`	(none)	(none)	Moves the cursor to the next line in the terminal window.

```
// Using the printf() method to prepare a nicely formatted bill.
public class SmallBill {
  public static void main(String[] args) {
    System.out.printf("Easy Data Ltd.          ");                    //
        (1)
    System.out.printf("%02d/%02d/%04d, %02d:%02d:%02d%n%n",          //
        (2)
                      20, 3, 2006, 19, 6, 9);
    System.out.printf("%-24s %8s %6s %8s%n",                         //
        (3)
                      "Item", "Price", "Count", "Sum");

    int count =2;
    double price = 132.25, sum = count*price, total = sum;
    System.out.printf("%-24s %8.2f %6d %8.2f%n",                     //
        (4)
                      "Ultraflash, USB 2.0, 1GB", price, count, sum); //
                          (5)

    count =1;
    price = 355.0; sum = count*price; total = total + sum;
    System.out.printf("%-24s %8.2f %6d %8.2f%n",                     //
        (6)
                      "Mega HD, 300GB", price, count, sum);           //
                          (7)

    count = 3;
    price = 8.33; sum = count*price; total = total + sum;
    System.out.printf("%-24s %8.2f %6d %8.2f%n",                     //
        (8)
                      "USB 2.0 cable, 2m", price, count, sum);        //
                          (9)

    System.out.printf("%40s %8.2f%n", "Total:", total);              //
        (10)
  }
}
```

Listing 5: SmallBill Program

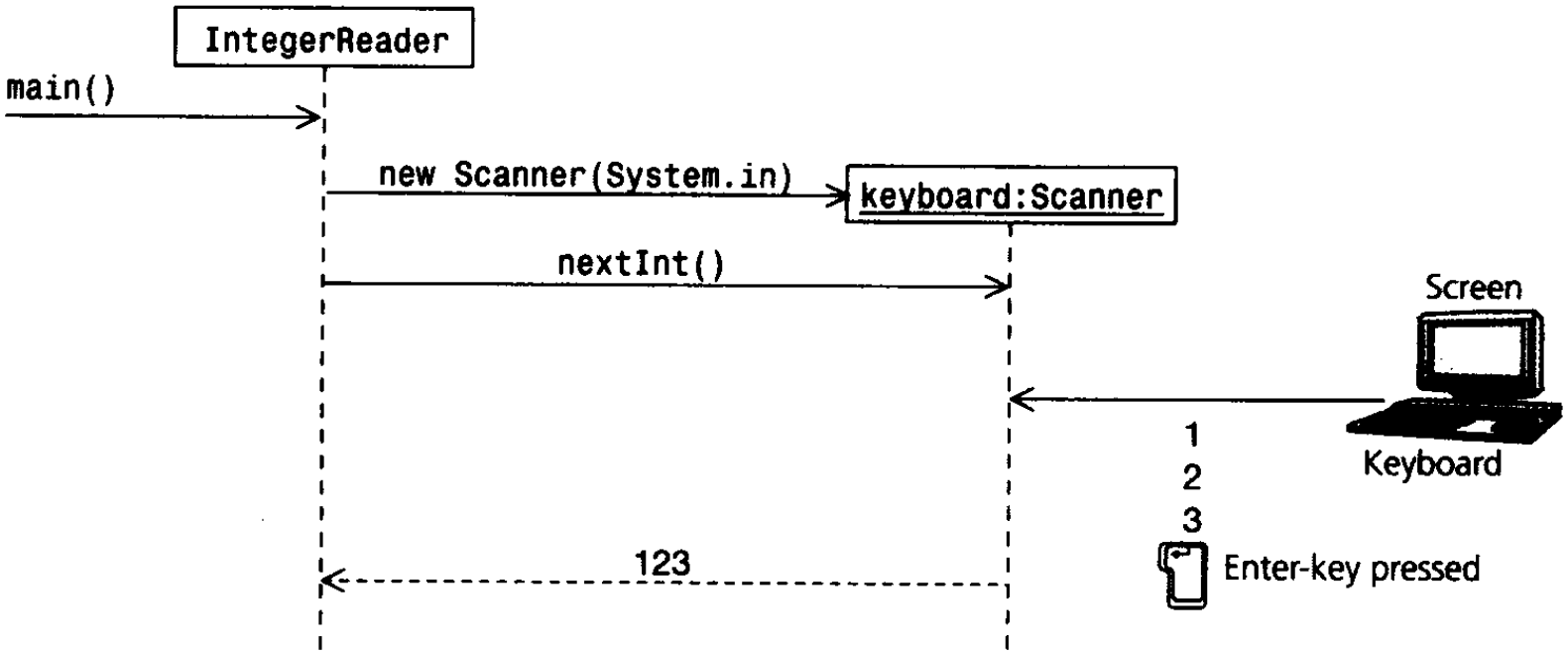
Program Output

```
Easy␣Data␣Ltd.␣␣␣␣␣␣␣␣␣␣20/03/2006,␣19:06:09

Item␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Price␣␣Count␣␣␣␣␣␣Sum
Ultraflash,␣USB␣2.0,␣1GB␣␣␣132.25␣␣␣␣␣␣2␣␣␣264.50
Mega␣HD,␣300GB␣␣␣␣␣␣␣␣␣␣␣␣355.00␣␣␣␣␣␣1␣␣␣355.00
USB␣2.0␣cable,␣2m␣␣␣␣␣␣␣␣␣␣␣8.33␣␣␣␣␣␣3␣␣␣␣24.99
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Total:␣␣␣644.49
```

The **Scanner** class

- It offers two methods, **nextInt()** and **nextDouble()**
- **nextInt()** is for reading **integer** values
- **nextDouble()** is for reading **floating-point** values



```
1  // Reading an integer from the keyboard using the Scanner class.
   import java.util.Scanner;                                          //
       (1)
3  public class IntegerReader {
     public static void main(String[] args) {
5      Scanner keyboard = new Scanner(System.in);                     //
           (2)

7      System.out.print("Enter an integer: ");                        //
           (3)
       int numberRead = keyboard.nextInt();                           //
           (4)
9
       System.out.printf("You entered the number %d%n", numberRead);  //
           (5)
11   }
   }
```

Listing 6: IntegerReader Program

Program Output

```
Enter an integer: 123
You entered the number 123
```

- Values read from the keyboard can be stored and used in computations

```
// Calculating the area of rectangle whose sides are input from the
    keyboard.
import java.util.Scanner;
public class IntegerArea {
  public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);

    System.out.print("Enter the rectangle length [integer]: ");     //
        (1)
    int length = keyboard.nextInt();                                 //
        (2)
    System.out.print("Enter the rectangle breadth [integer]: ");    //
        (3)
    int breadth = keyboard.nextInt();                                //
        (4)

    int area = length * breadth;                                     //
        (5)
    System.out.printf("A rectangle of length %d cm and breadth" +
                      " %d cm has area %d sq. cm.%n",
                      length, breadth, area);                        //
                          (6)
  }
}
```

Listing 7: IntegerArea Program

```
1  // Calculating the area of rectangle whose sides are input from the
       keyboard.
   import java.util.Scanner;
3  public class FloatingPointArea {
     public static void main(String[] args) {
5      Scanner keyboard = new Scanner(System.in);

7      System.out.print("Enter the rectangle length [decimal number]: ");//
           (1)
       double length = keyboard.nextDouble();                              //
           (2)
9      System.out.print("Enter the rectangle breadth " +
                        "[decimal number]: ");                             //
                             (3)
11     double breadth = keyboard.nextDouble();                             //
           (4)

13     double area = length * breadth;                                     //
           (5)
       System.out.printf(
15       "A rectangle of length %.2f cm and breadth %.2f cm" +
         " has area %.2f sq. cm.%n",
17       length, breadth, area);                                           //
             (6)
     }
19 }
```

Listing 8: FloatingPointArea Program

- When entering numbers at the keyboard, a user might unintentionally enter an invalid value
- **java IntegerArea** + type in a decimal instead of an integer

Program Output

```
Enter the rectangle length [integer]: 3
Enter the rectangle breadth [integer]: 4.5
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:840)
at java.util.Scanner.next(Scanner.java:1461)
at java.util.Scanner.nextInt(Scanner.java:2091)
at java.util.Scanner.nextInt(Scanner.java:2050)
at IntegerArea.main(IntegerArea.java:10)
```

- This type of error is called an **exception** **(We will explain later)**
- For both integer and floating-point values, the syntax of the input value determines whether it is accepted
- The Scanner class has no way of catching values that are meaningless to the program
- The Scanner class is meant to be flexible, and therefore only offers syntactic validation of input data

- We can also read multiple values from the same line in the terminal window

- After the user has entered the integers,
  - the first call to the **nextInt()** method assigns the first value to the first variable
  - the first call assigns the second value to the second variable
  - etc.

```
1  // Reading multiple values from the same line on the keybouard.
   import java.util.Scanner;
3  public class ReadingMultipleValues {
     public static void main(String[] args) {
5      Scanner keyboard = new Scanner(System.in);

7      System.out.print("Enter two numbers [integer integer]: ");
       int number1 = keyboard.nextInt();                          //
           (1)
9      int number2 = keyboard.nextInt();                          //
           (2)

11     int sum = number1 + number2;
       System.out.printf("The sum of integers %d and %d is %d%n",
13                       number1, number2, sum);                  //
                             (3)
     }
15 }
```

Program Output

```
Enter two numbers [integer integer]: 12 23
The sum of integers 12 and 23 is 35
```