University American College Skopje

School of Computer Science and Information Technologies

Course: Programming Languages

2021/2022

Expressions. Control Structures. Loops. Arrays. Arrays of Arrays

Lecture 2

TYPES OF VALUES IN JAVA

- Java is a *strongly typed language.*
  - Every variable and expression has its type which is known at compile time.
  - Not so much in C++ – the type of pointer is realized at run time upon polymorphism.
- *The data type* determines the set of values that a variable can produce, limits the operations that are supported on those values and determines the sense of those operations.

TYPES OF VALUES IN JAVA

- Two groups of values in Java:
  - Primitive types;
    - Need not be instantiated (int, float, boolean, …).
  - Referent types.
    - Must be instantiated (all user defined classes and all Java API classes).
- Null – a special nameless type and just one value (null).
  - Can be considered a special character and/or value from any referent type.

EXPRESSIONS IN JAVA

- *Expressions* are groups of elements in Java, the execution of which produces a certain result.

- The result of the expression can be
  - Any data type (primitive, referent or null);
  - void.

EXPRESSIONS IN JAVA

- Since all operators are actually methods, it can be said that *expressions are parts of the language that include method invocation.*
- In other words:
  - Every method invocation is an expression.
    - Methods can be voidby return type, but they are still expressions that yield voidresults.
  - Every operation that involves an operator is an expression.
    - Operators are defined using methods which may be (and commonly are) overloaded.

EXPRESSIONS IN JAVA – EXAMPLES

Expression	Return type, Return value
`5 * 9`	⇒ `int`, `40`
`4 >= 9`	⇒ `boolean`, `false`
`Math.random()`	⇒ `double`, a random value between `0` and `1`
`System.out.println()`	⇒ `void`,
`"It is now " + 4 + " o`clock"`	⇒ `String`, `"It is now 4 o'clock"`
`c = '\n'`	⇒ `char`, newline (= is an assignment operator; the variable must previously be declared, using a *declaration statement*)

STATEMENTS IN JAVA

- *Statements* control the flow of program execution and affect its result.
  - Statements don't produce results.

- Nevertheless, statements depend on expressions to determine the course of subsequent program execution.

STATEMENTS AND EXPRESSIONS IN JAVA

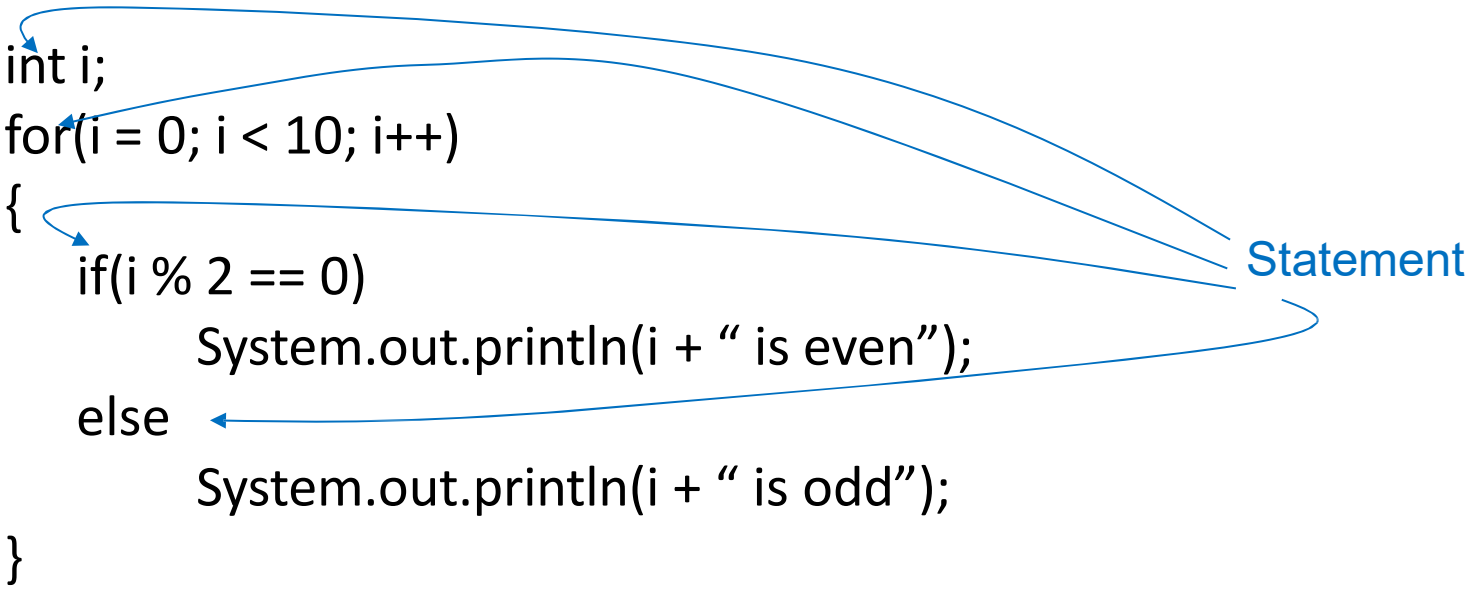
- Example:

```
int i;
for(i = 0; i < 10; i++)
{
    if(i % 2 == 0)
        System.out.println(i + " is even");
    else
        System.out.println(i + " is odd");
}
```

- How many statements and how many expressions are present in this code segment?

STATEMENTS AND EXPRESSIONS IN JAVA

- Example:

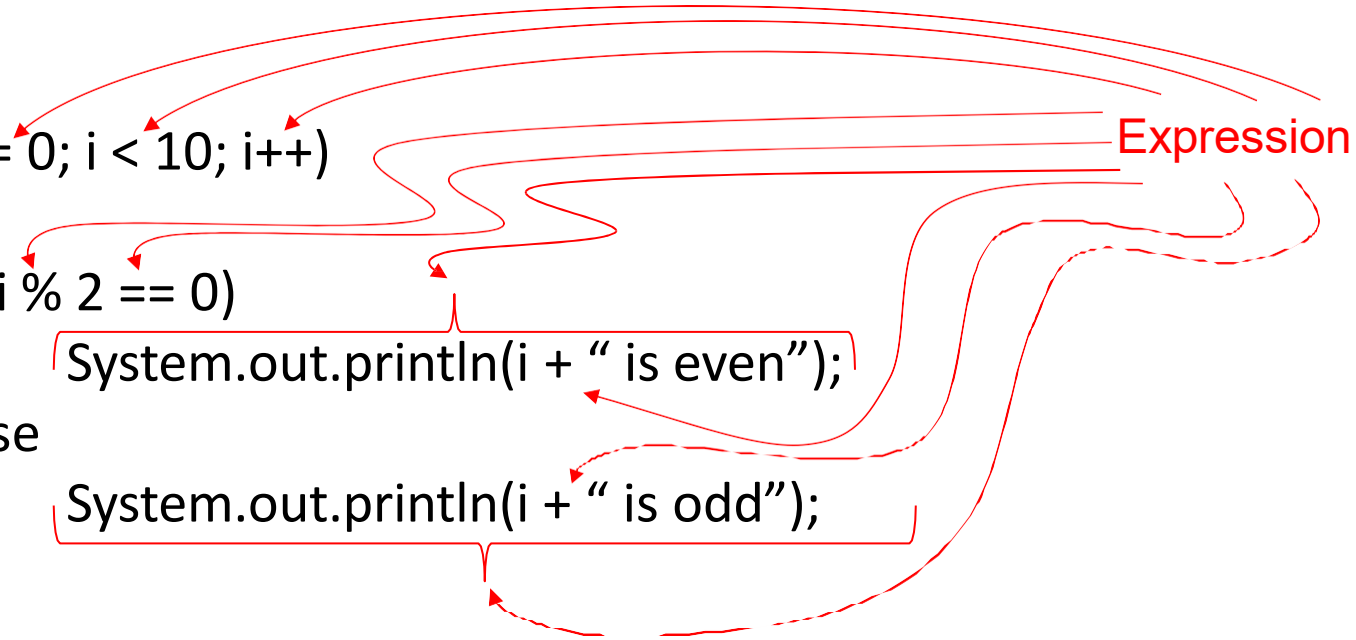


- How many statements and how many expressions are present in this code segment?

STATEMENTS AND EXPRESSIONS IN JAVA

- Example:

```
int i;
for(i = 0; i < 10; i++)
{
    if(i % 2 == 0)
        System.out.println(i + " is even");
    else
        System.out.println(i + " is odd");
}
```



- How many statements and how many expressions are present in this code segment?

STATEMENTS AND EXPRESSIONS IN JAVA

REMEMBER

- A statement **NEVER** produces a result.

- Expression **ALWAYS** produces a result, even though it might be void or null.
  - void= no value.
  - null= a value that is a reference to nothing (null≠ void).

CONTROL STRUCTURES

- if
- if – else
- switch
- while
- do-while
- for

IF

```
if (expression) {
    statement1;
    statement2;
    ….
}
```

- **expression** is an expression that can be either true or false.
- **if** is a statement that makes a decision based on logical conditioning.

COMPARISON OPERATORS

==	e.g.,	`x==y`	means x is equal to y
!=	e.g.,	`x!=y`	means x is not equal to y
>	e.g.,	`x>y`	
<	e.g.,	`c<d`	
>=	e.g.,	`e>=f`	
<=	e.g.,	`g<=k`	

OPERATOR PRIORITY

1. * / %
2. + -
3. < <= > >=
4. == !=
5. =

AN EXAMPLE FOR `IF`

If the condition (boolean) is true, the statements are executed; otherwise nothing is executed.

Examples:

```
if (percent >= 60.0)
        System.out.println("You have passed the
   exam!");
```

```
if (percent >= 60)
   {
        System.out.println("You have passed with " +
   percent + " percent.");
        System.out.println("Congratulations!");
   }
```

IF .. ELSE

If the condition (boolean) is true, the first statement (or block of statements) is executed, otherwise the next statement (or block of statements) is executed.

```
if (I_have_a_ticket) //same as: if (I_have_a_ticket == true)
   {
       System.out.println("I'm going to the concert!");
       System.out.println("How nice!");
   }
   else /*if (I_have_a_ticket != true) or if (!I_have_a_ticket)
   or if (I_have_a_ticket == false)*/
   {
       System.out.println("I'll watch the concert on the TV.");
       System.out.println("I couldn't find a ticket.");
   }
```

NESTED IF STATEMENTS

- The ifstatements can be nested into one another:

```
if (i > k)
{
   if (j > k)
        System.out.println("i and j are larger than k");
}
else
   System.out.println(i is less or equal to k");
```

- Braces play an important role when there are more if statements than else statements:

```
if (i > k)
   if (j > k)
        System.out.println("i and j are greater than k");
else
   System.out.println(i is greater than k, but j is less or
   equal to k");
```

SWITCH

```
switch (expression) {
   case case1:
        statement1;
        statement2;
        ....
        break;//this is an interruption of the current case.
              //Without it, all cases below the selected one
              //will be executed
   case case2:
        statement1;
        statement2;
        ....
        break;
   .....
   default:  //this will be executed by defalut
        statement1;
        statement2; ....
        break;
}
```

A SWITCH EXAMPLE

```
switch (temperature/10) {
    case 3:
          System.out.println("The temperature is above 30 degrees.");
          break; //This means the case is interrupted
    case 2:
          System.out.println("The temperature is between 20 and 30 degrees.");
          break;
    case 1:
          System.out.println("The temperature is between 10 and 20 degrees.");
          break;
   case 0:
          System.out.println("The temperature is between 0 and 10 degrees.");
          break;
    default:
          System.out.println("The temperature is below 0 degrees.");
    break;
}
```

WHILE AND DO-WHILE

General form of the `while` statement:

```
counter initialization;
while (condition for continuation of the loop)
{
      statement(s);
      updating the value of the counter, to reach a
state of non-fulfillment of the condition;
}
```

General form of the `do-while` statement:

```
counter initialization;
do
{
      statement(s);
      updating the value of the counter, to reach a
state of non-fulfillment of the condition;
} while (condition for the continuation of the loop)
```

WHILE AND DO-WHILE

1.

```
int i = 10;
while (i < 100){
  System.out.println("The value of i = "+ i);
  i = i + 10;
}
```

2.

```
int  i = 10;
do  {
   System.out.println("i="+i);
   i = i + 10;
   } while (i <100);
```

Question :

How many times the value of i will be printed in the code segment 1. and how many in 2. if at the beginning of the loop the value of i is set to 100 (i=100)?

NESTED STATEMENTS

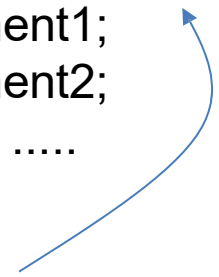
```
int     j = 5;
int num_even=0;
int num_odd=0;
while (j > 0)
{
    if (j % 2 ==0){
        System.out.println("Even number "+j);
        num_even++;
    }
    else {
        System.out.println("Odd number "+j);
        num_odd++;
   }
    j = j - 1;
}
```

FOR

Example:
```
for (int i=0; i<5; i++){
        System.out.println(i);
    }
```

General form

```
for(loop initialization; condition for continuation of the loop; post-iteration action) {
  statement1;
  statement2;
        .....
}
```



Executed once, upon the start of the loop

Tested for at the *beginning* of each iteration

Executed at the *end* of each iteration

FOR

- The forstatement has 3 parts:
  - Loop initialization;
  - Condition for loop continuation;
  - Post-iteration action.

- *None* of those parts are mandatory, and the loop continuation condition is considered true if omitted.

- Also, there can be more than one action in the loop initialization part or the post-iteration part, if the actions are delimited by commas.

FOR

- Examples – all give the same result:

Two actions delimited by a comma

```
for(int i = 1; i < 100; System.out.println(i), i++);

for(int i = 1; i < 100; )      { System.out.println(i); i++;}

for(int i = 1; ; i++)                    { System.out.println(i); if(i == 100) break; }

int i = 1; for( ; i < 100; i++)          { System.out.println(i); }

int i = 1; for( ; ; )                     { System.out.println(i++); if(i == 100) break; }
```

Infinite loop

BREAK AND CONTINUE

break is used to leave a loop, without testing for the loop termination condition.

e.g.,

```
i=1;
while (i<10){
        i++;
if (i%2 ==0) break;

System.out.println(i);
}
```

How many lines will be printed on the screen?

continue is used to ignore the current loop iteration. The loop is executed until the loop termination condition is fulfilled.

e.g.,

```
i=1;
while (i<10){
i++;
if (i%2 ==0) continue;

System.out.println(i);

}
```

How many lines will be printed on the screen?

BOOLEAN TESTING

- Boolean testing occurs in:
  - The condition of the ifstatement;
  - The condition for execution of the whilestatement;
  - The condition for execution of the do-whilestatement;
  - The condition for execution of the forstatement.

- The test is whether a boolean*expression*, i.e., a boolean*value*, is true.
  - If it is, the statement executes.
  - In the case of an else statement, it is executed if the booleanexpression, i.e., value, is false.

BOOLEAN TESTING

- Therefore, *booleanvalues* can be used to replace the expressions which are used as tests.

- For example,

```
if (6 % 2 == 0)
   System.out.println("6 is even");
```

can be replaced by

```
boolean sixIsEven = 6%2 == 0;
if (sixIsEven)
   System.out.println("6 is even");
```

It is calculated first, because the equality operator (==) has a greater priority than the assignment operator (=)

BOOLEAN TESTING

- When testing for boolean values, it is redundant to use the equality operator (==) because it produces a booleanvalue as a result, whereas the variable used for testing is itself boolean.

- Example:

```
if (sixIsEven == true) System.out.println("6 is even"); //redundant
if (sixIsEven)         System.out.println("6 is even"); //same result
```

BOOLEAN TESTING

- Furthermore, it induces errors, because of its similarity with the assignment operator (=).

- Example – the code in the loop will be executed once:

```
boolean thisIsTrue = true; while
(thisIsTrue == true)
{
    System.out.println("It is executed");
    thisIsTrue = false;
}
```

- Example – the code in the loop will be executed indefinitely:

```
boolean thisIsTrue = true;
while (thisIsTrue = true)
{
   System.out.println("It is executed");
   thisIsTrue = false;
}
```

Probably an error in the code, which can be troublesome to find

ARRAYS IN JAVA

- An array is an **object** which contains a *fixed amount of values* of one type.

- This means that:
  - The array is an **object** – it is a reference data type and must be instantiated to be able to be used.
  - Contains a *fixed amount of values* – when its size is determined, it cannot be changed further.
  - Contains values of one type – values of types other than the one for which the array has been declared cannot be used.

DECLARATION OF ARRAYS IN JAVA

- The array in Java can be declared on one of the following two ways:

`int a[];` **or** `int[] a;` ← Preferred style for Java

This declares aas an array of integers.

- Examples:
  - `byte[] lotto;` **array of** `byte`**s.**
  - `char[] acronym;` **array of** `char`**acters.**
  - `String[] args;` **array of** `String`**s.**
  - …

DEFINITION OF ARRAYS IN JAVA

- After the array is declared, its default value is null.
  - As with any other reference data type.
- For an array to be defined, the **new**operator is used, as well as the array size, in square brackets (i.e., []).
  - The size must be an integer value, greater than zero.
- Example:

```
String[] greeting = new String[4];//This defines an
                                  //array of 4 Strings
```

DEFINITION OF ARRAYS IN JAVA

- After the array is declared and its size is determined, its individual elements can be accessed in order to be altered and/or used.
- Example:

```
greeting[0] = "Hello. ";
greeting[1] = "How ";
greeting[2] = "are you ";
greeting[3] = "today? ";
```

The first element of the array always has an index of 0, and the last one always has an index of *arraySize*-1.

DEFINITION OF ARRAYS IN JAVA

- It is possible to declare and define the array at the same time, without specifying the array size.
- Example:

```
String[] greeting = {"Hello. ", "How ",
  "are you ", "today? "};
```

This way, the array is automatically set to a size of 4 elements.

ARRAYS OF ARRAYS IN JAVA

- It is possible to define arrays with more than 1 dimension.
- This is accomplished by defining *arrays of arrays* in Java, i.e., multidimensional arrays.
- The declaration and definition of multidimensional arrays in Java is analogous with the declaration and definition of one-dimensional arrays.

ARRAYS OF ARRAYS IN JAVA

- Example – declaring a two-dimensional array:

```
boolean [][] isDivisibleBy = new boolean[5][5];
isDivisibleBy[0][0] = false;
isDivisibleBy[0][1] = true;
isDivisibleBy[0][2] = true;
…
isDivisibleBy[4][1] = true;
isDivisibleBy[4][2] = true;
isDivisibleBy[4][3] = false;
isDivisibleBy[4][4] = true;
```

ARRAYS OF ARRAYS IN JAVA

- A multidimensional array can also be directly defined:

```
boolean [][] isDivisibleBy = {{false, true, true, true, true},
                              {false, true, false, false, false},
                              {false, true, true, false, false},
                              {false, true, false, true, false},
                              {false, true, true, false, true}};
```

Definition of a row in a multidimensional array

Is divisible by	0	1	2	3	4
0	false	true	true	true	true
1	false	true	false	false	false
2	false	true	true	false	false
3	false	true	false	true	false
4	false	true	true	false	true

JAGGED ARRAYS

- A multidimensional array need not have all of its rows have equal lengths.
- It is possible that one row have 10 elements, another one have 3 elements etc.
- This way so called *jagged arrays* are obtained.

JAGGED ARRAYS

- Example:

```
String [][] seatReservedFor =
{
{“Guest1”, “Guest2”, “Guest3”},
{“Guest4”, “Guest5”},
{“Guest6”, “Guest7”, “Guest8”, “Guest9”},
{“Guest10”}
};
```

3 elements in a row

2 elements in a row

4 elements in a row

1 element in a row

JAGGED ARRAYS

- ATTENTION: if the multidimensional array is entered this way, it will be considered jagged and will NOT fill up with the default values on the places where elements would be needed so that a matrix NxN would be obtained.
  - This is the case in C++, where jagged arrays are not supported and all multidimensional arrays are treated as matrices.

QUESTIONS?