Object Oriented Programming Practicum

ICT2132

Control Statements and Arrays

P.H.P. Nuwan Laksiri
Department of ICT
Faculty of Technology
University of Ruhuna

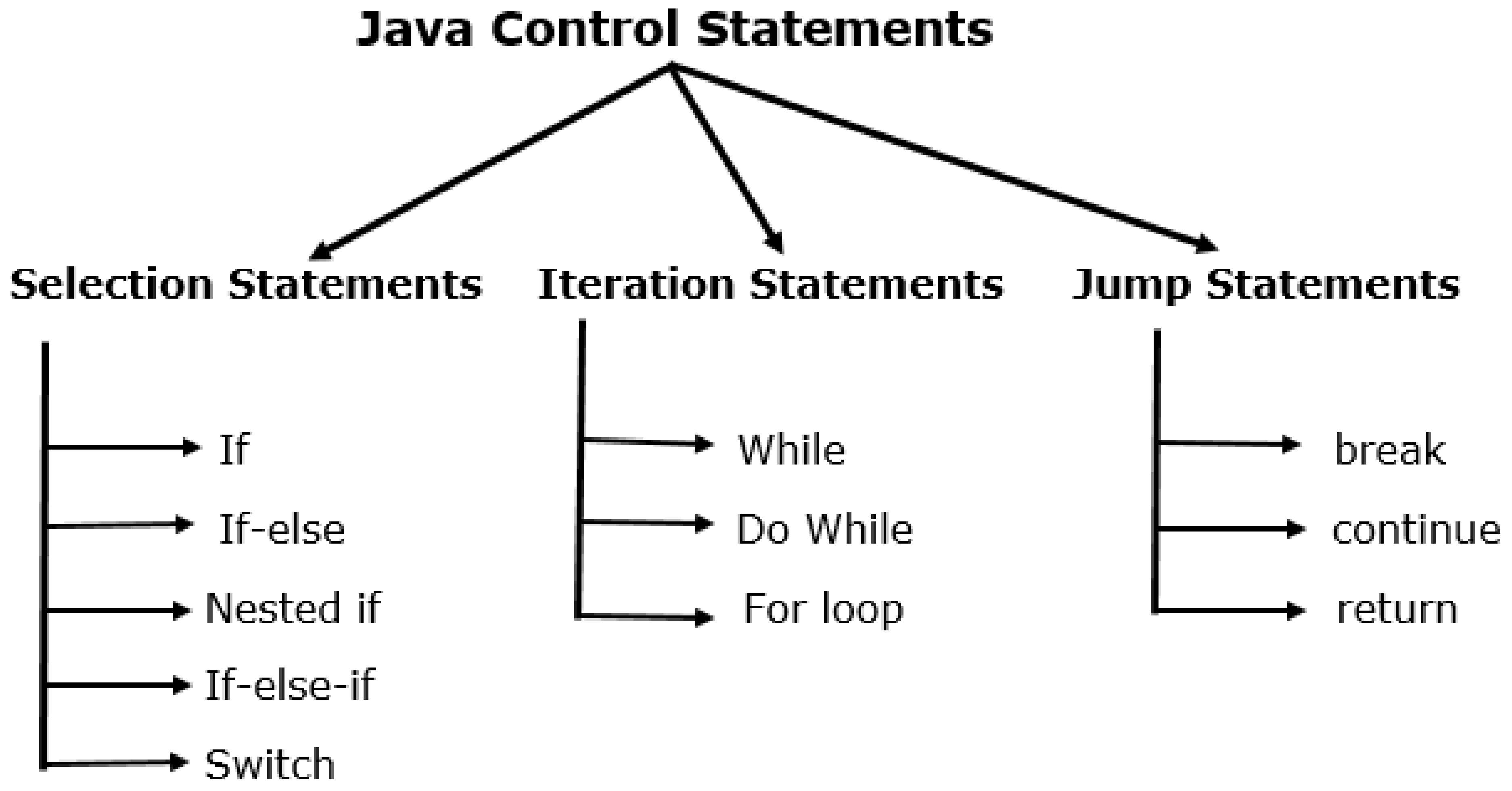
Lesson 02

Recap

- JAVA – Primitive Types
- JAVA – Reference Types
- Primitive vs Reference
- JAVA – Wrapper Classes for Primitives
- JAVA – Literals
  - Integer Literals
  - Floating point Literals
  - String Literals
  - Character Literals
  - Boolean Literals
- JAVA – Constants
- JAVA –Operators
- JAVA - Type Casting

Outline

- Sequence Control
- Selection Control
  - *if* Statement
  - *if – else* Statement
  - Nested *if* Statement
  - *switch* Statement
- Repetition Control
  - *while* Statement
  - *do – while* Statement
  - *for* Statement
    - Enhanced *for* Statement
- Branching Statements
  - *Break*
  - *Continue*
- Arrays
  - Declaring Arrays
  - Creating Arrays
  - Initializing Array values
  - Single and Multidimensional Arrays
  - Advantages/Disadvantages


Java Control Statements
Selection Statements
Iteration Statements
Jump Statements
If
If-else
Nested if
If-else-if
Switch
While
Do While
For loop
break
continue
return

Sequence Control

- The sequence structure is trivial
- Simply list the statements to execute in the order in which they should execute

if Statement (selection)

```
        if(condition){
                statement(s);
        }
```

Ex :

Assign your age into variable, if the age is greater than or equal18, display "You are an adult" message.

```
if(age>=18){
        System.out.println("You are an adult");
}
```

if – else Statement(selection)

```
if(condition)
{
        statement(s);
}
else
{
        statement(s);
}
```

if – else Statement(selection)

Ex :

Modify the above exercise to display "You are a teenager", if the age is less than 18.

```
if(age<18)
{
	System.out.println("You are a teenager");
}
else
{
	System.out.println("You are an adult");
}
```

Nested *if* Statement(selection) ???

```
if(condition1)
{
        statement(s);
}
```

if – else – if Statement(selection)

```
if(condition1)
{
        statement(s);
}
else if(condition 2)
{
        statement(s);
}
else
{
        statement(s);
}
```

If –else –if Statement(selection)

Ex :

Modify the above exercise to display “You are a kid”, if the age is less than 10.

```
if(age>=18)
{
        System.out.println(“You are an adult”);
}
else if(age >=10)
{
        System.out.println(“You are a teenager”);
}
else
{
        System.out.println(“You are a kid”);
}
```

switch Statement(selection)

```
switch (var)
{
        case val1:
                statement(s);
                break;
        case val2:
                statement(s);
                break;
        ….
        default:
                statement(s);
}
```

switch Statement(selection) ???

Ex :

Modify the above implemented code (using *if –else-if)* segment with *switch* statement.

```
int age = 17;
switch (age)
{
        case (age>=18):
                System.out.println(“You are an adult”);
                break;
        case (age>=10):
                System.out.println(“You are a teenager”);
                break;
        default:
                System.out.println(“You are a kid”);
}
```

while Statement(repetition)

```
while (expression)
{
        statement(s);
}

Ex :
int count = 1;
while (count < 11)
{
        System.out.println("Count is: " + count);
        count++;
}
```

do - while Statement(repetition)

```
do
{
        statement(s);
} while (expression) ;
```

Ex :

```
int count = 1;
do
{
        System.out.println("Count is: " + count);
        count++;
} while (count < 11);
```

for Statement(repetition)

```
for (initialization; termination; increment)
{
        statement(s);
}

Ex:
for(int i=1; i<11; i++)
{
        System.out.println("Count is: " + i);
}
```

Enhanced *for* Statement(*foreach*)

HOMEWORK……..☺

break Statement

- The break keyword can be used in any of the loop control structures to cause the loop to *terminate immediately*.
- When a break occurs, no matter what the value is of the loop counter or the Boolean expression, the flow of control will jump to the next statement past the loop.

break Statement

Ex :

Try *for*, *while* and *do while* loop statement with “break”.

continue Statement

- The continue keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.
- In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
- In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

continue Statement

Ex :

Try *for*, *while* and *do while* loop statement with “continue”.

HOMEWORK…☺

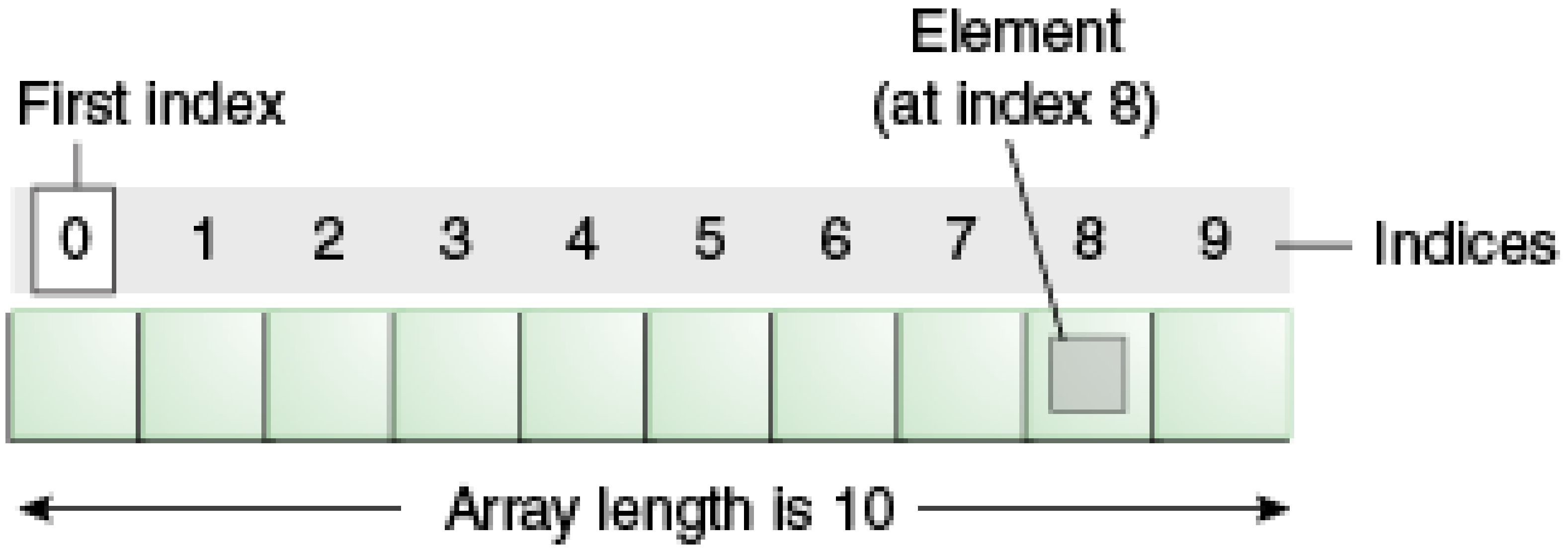
What happens with switch ???

return Statement

HOMEWORK…☺

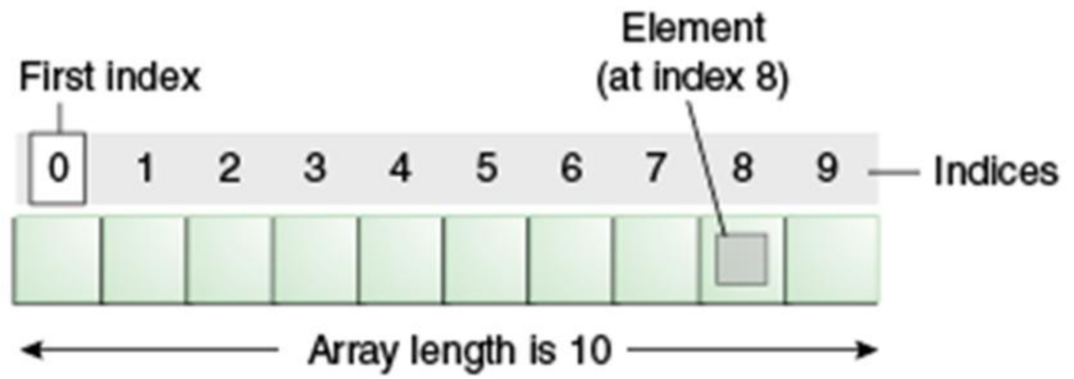
What happens with return in control statements ???

Arrays


Element
(at index 8)
First index
0 1 2 3 4 5 6 7 8 9
Indices
Array length is 10

Arrays

- A container object that holds a fixed number of values of similar data type.
- The length of an array is established when the array is created.
- Each item in an array is called an element, and each element is accessed by its numerical index.



Steps in Making an *Array*

- Three distinct steps in making an array.
  - Declare the array name.
  - Create the array.
  - Initialize the array values.

Declaring the Array Name

```
int[] anArray;
double[] anArrayOfDouble;
byte[] anArrayOfBytes;
short[] anArrayOfShorts;
long[] anArrayOfLongs;
float[] anArrayOfFloats;
boolean[] anArrayOfBooleans;
char[] anArrayOfChars;
String[] anArrayOfStrings;
```

- You can place the brackets after the array name also.

```
float anArrayOfFloats[];
```

Creating the array

- Use new operator to create an array with fixed size.

```
anArray = new int[10];
```

Initializing the array

- Initialize in declaration.

```
int[] anArray={10, 20, 30, 40, 50, 60} ;
```

  - The length of the array is determined by the number of values provided between braces and separated by commas.

- Initialize elements one by one.

```
anArray[0]=100;
anArray[1]=10;
```

- Using a loop.

```
for (int i = 0; i < 10; i++)     // elements are indexed from 0 to 9
        anArray[i] = 0;      // initialize all elements to 0.0
```

Accessing array elements

- System.out.println("Element 1 at index 0: " + anArray[0]);
- System.out.println("Element 2 at index 1: " + anArray[1]);
- System.out.println("Element 3 at index 2: " + anArray[2]);

for each Loop

- Check if there’s any relationship between for each and arrays

Arrays of Strings

- Strings can be part of an array
- Create array of Strings with 3 elements

Ex:

```
String[] deptName = {"Accounting", "Human Resources",
"Sales"};
for(int a = 0; a < deptName.length; ++a)
        System.out.println(deptName[a]);
```

Try Out… ☺

- Suppose list[] is an array holding double values and find the maximum element of the array.

- Search a specific value from the array without knowing the exact index.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Multidimensional Array

- A multidimensional array is an array whose components are themselves arrays
- Using two or more sets of brackets

```
String[][] names;
String[][] names = { {"Mr. ", "Mrs. ", "Ms."},
        {"Smith", "Jones"} };
String[][] names = new String[3][2];
```

- Accessing elements

```
System.out.println(names[0][0] + names[1][0]);
```

Advantages/Disadvantages

- Advantages
  - Code Optimization:

    It makes the code optimized, we can retrieve or sort the data easily.
  - Random access:

    We can get any data located at any index position.
- Disadvantages
  - Size Limit:

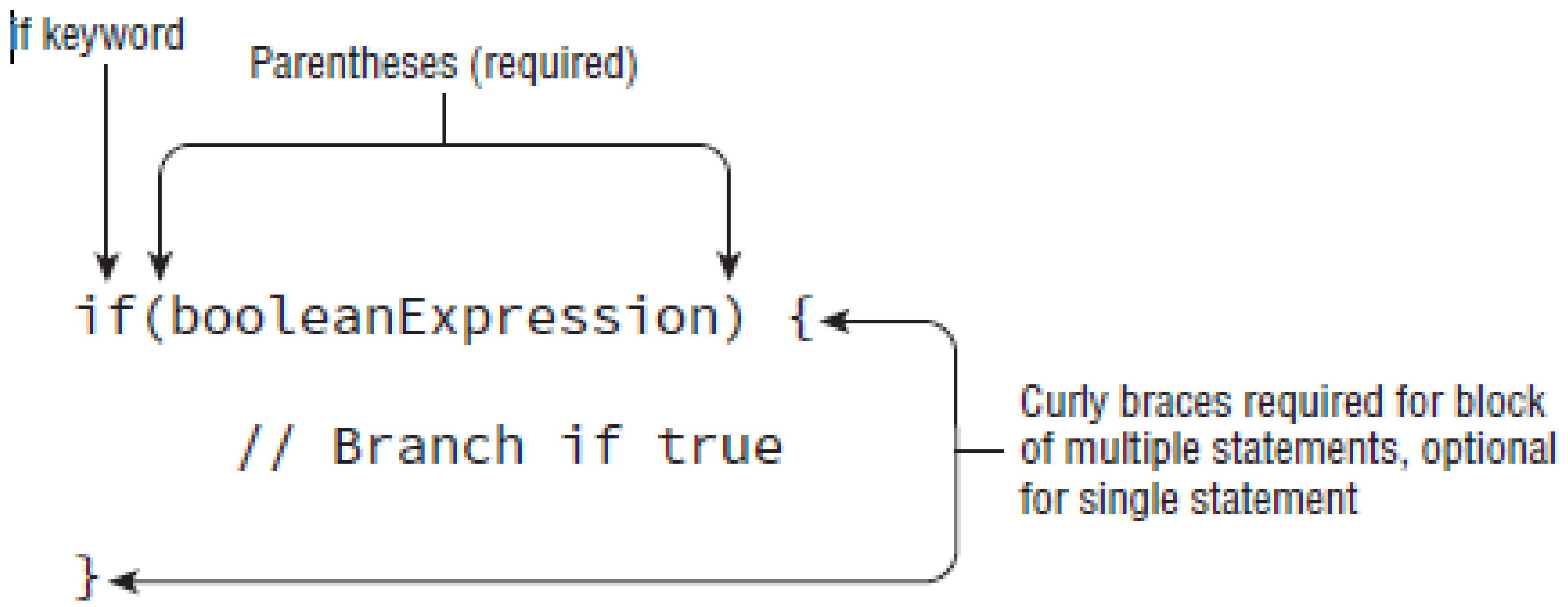
    Store only fixed size of elements in the array. It doesn't grow its size at runtime.

Java Statements - Highlights

- a Java statement is a complete unit of execution in Java, terminated with a semicolon (;).
- Control flow statements break up the flow of execution by using decision making, looping, and branching, allowing the application to selectively execute particular segments of code.
- a block of code in Java is a group of zero or more statements between balanced braces, ({}), and can be used anywhere a single statement is allowed.

The if-then Statement - Highlights

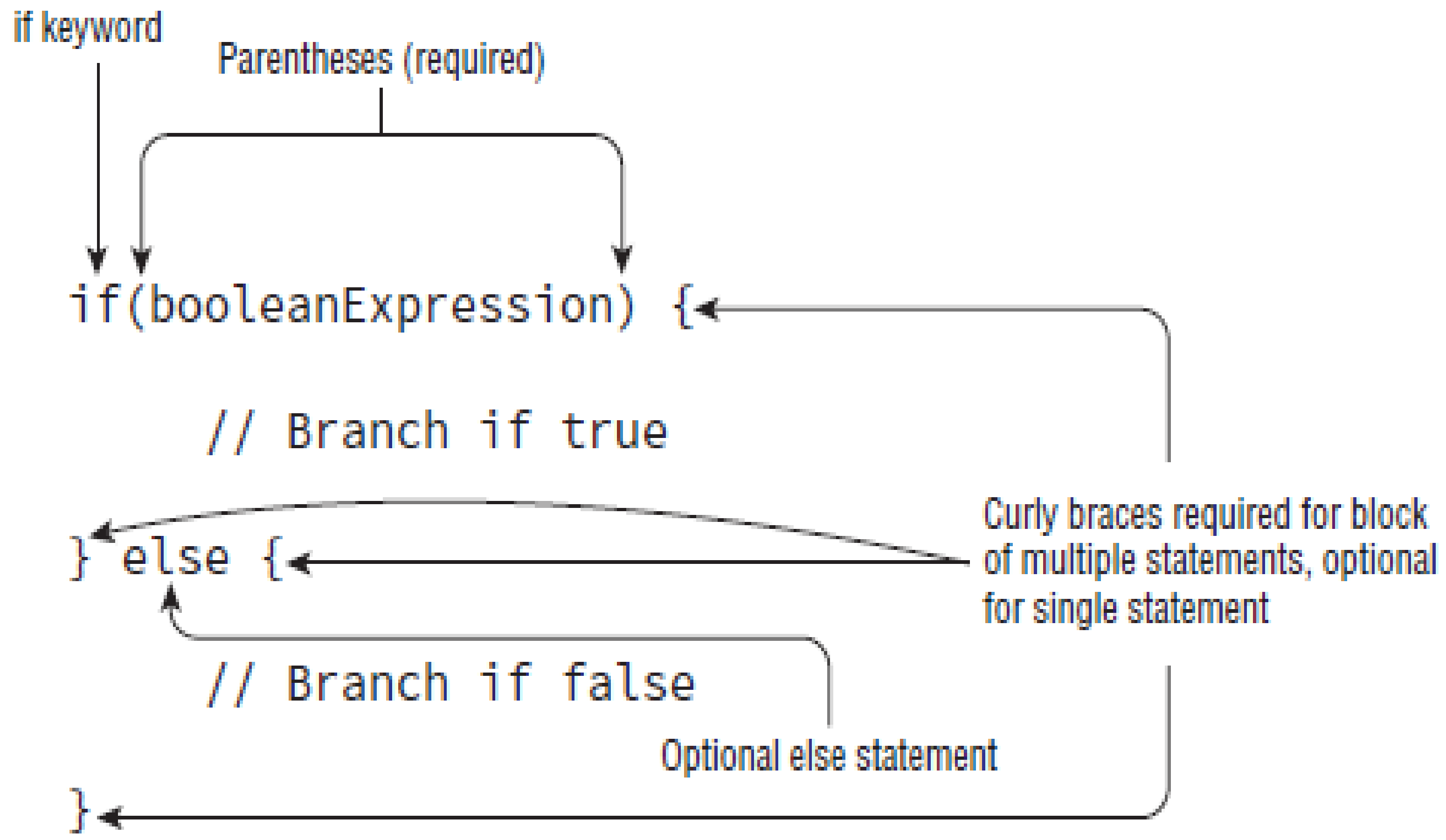
- We only want to execute a block of code under certain circumstances, The if-then statement is for it.



```
if(booleanExpression) {

      // Branch if true

}
```

if keyword
Parentheses (required)
Curly braces required for block of multiple statements, optional for single statement

The if-then-else Statement - Highlights

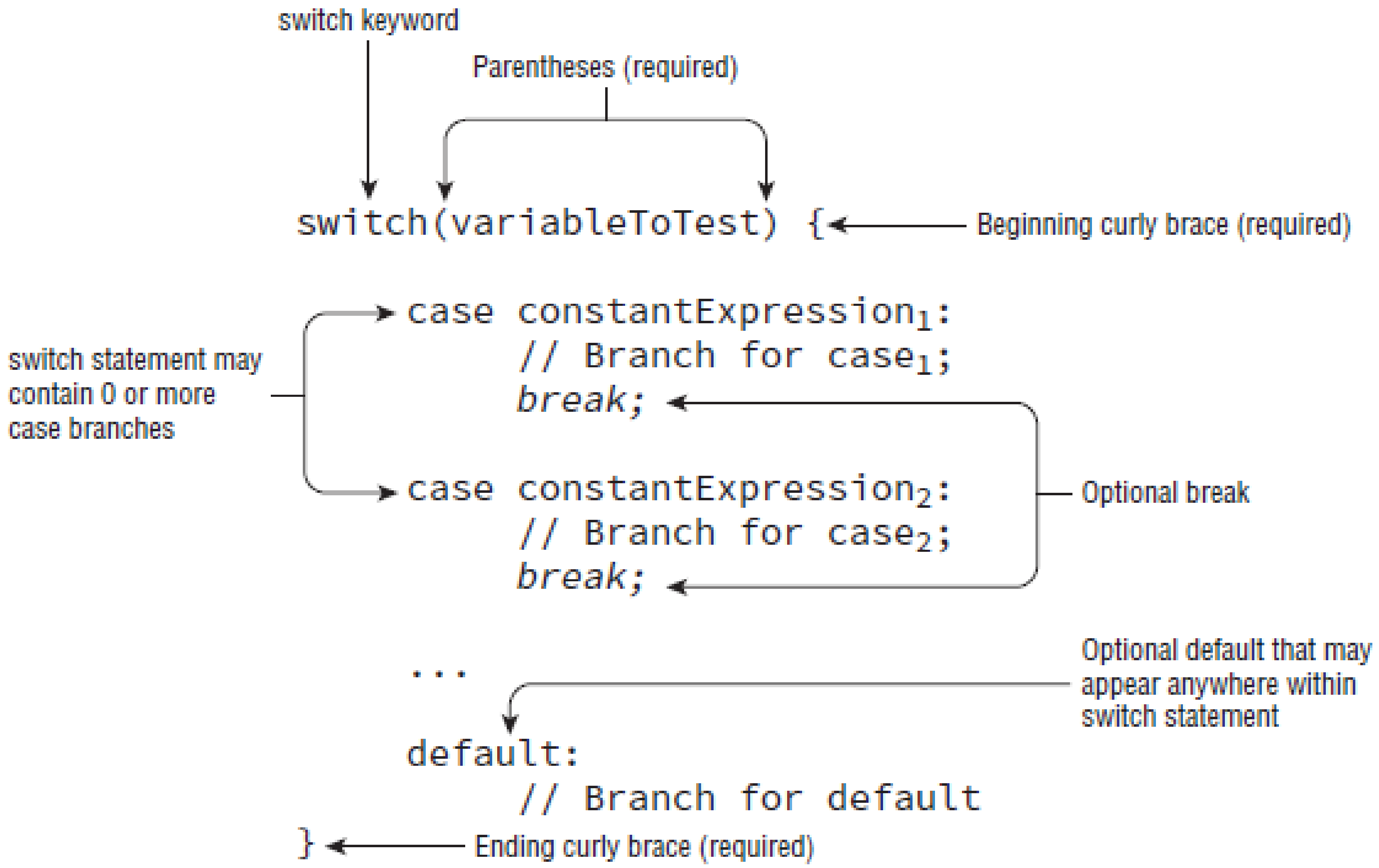

if keyword
Parentheses (required)
Curly braces required for block of multiple statements, optional for single statement
Optional else statement

```
if(booleanExpression) {

    // Branch if true

} else {

    // Branch if false

}
```

The switch Statement - Highlights

- Data types supported by switch statements include int and Integer, byte and Byte, short and Short, char and Character, int and Integer, String, enum values

The switch Statement - Highlights

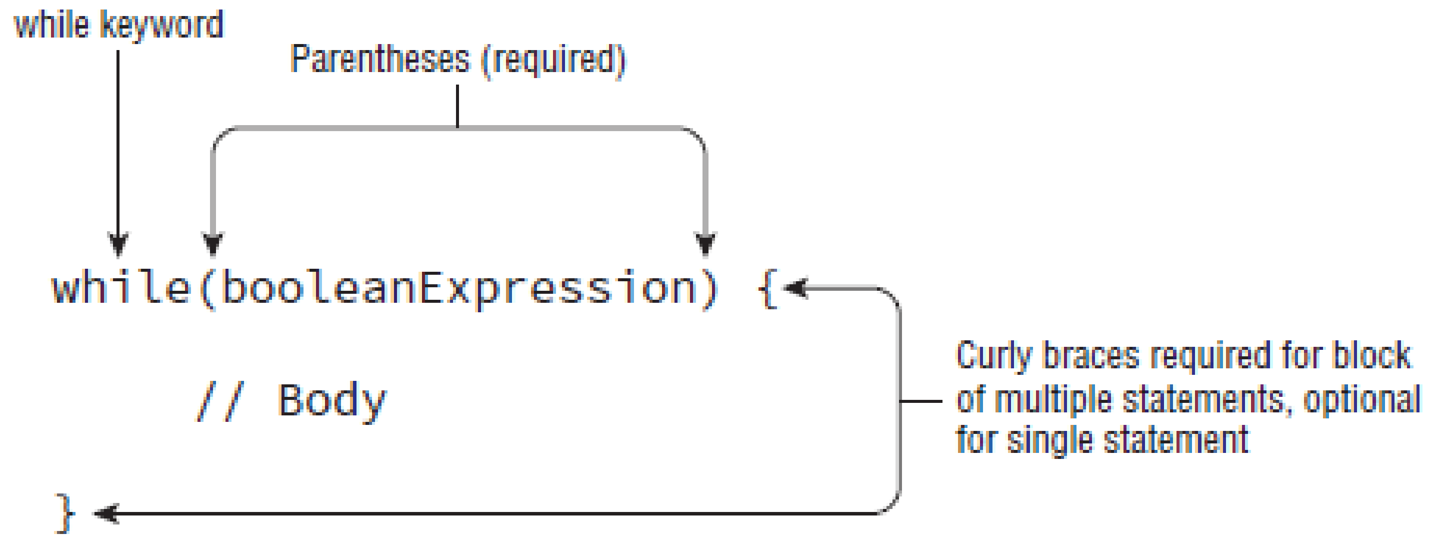


```
switch(variableToTest) {

    case constantExpression1:
        // Branch for case1;
        break;

    case constantExpression2:
        // Branch for case2;
        break;

    ...

    default:
        // Branch for default
}
```

switch keyword
Parentheses (required)
Beginning curly brace (required)
switch statement may contain 0 or more case branches
Optional break
Optional default that may appear anywhere within switch statement
Ending curly brace (required)

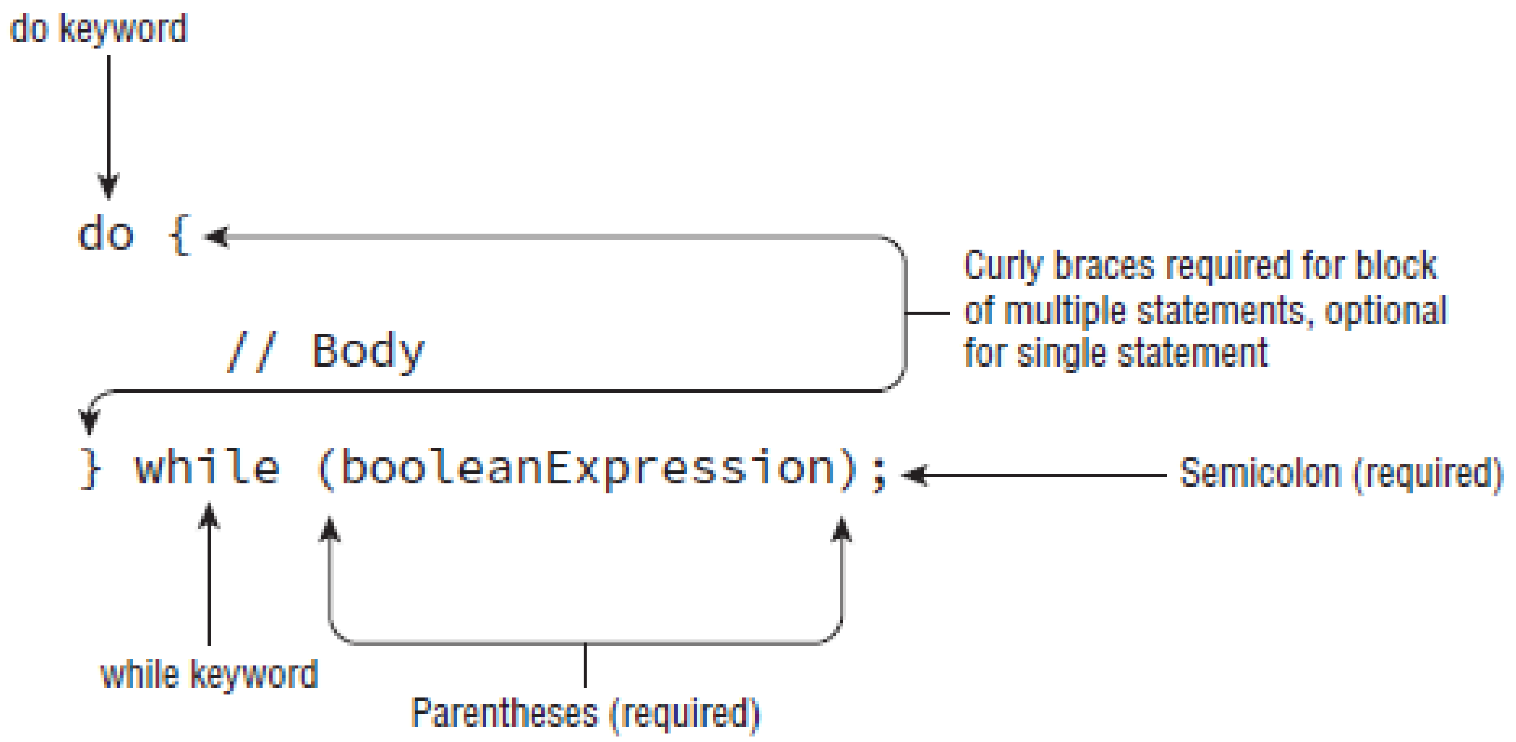
The while Statement - Highlights

- A repetition control structure, which we refer to as a loop, executes a statement of code multiple times in succession
- During execution, the Boolean expression is evaluated before each iteration of the loop and exits if the evaluation returns false.

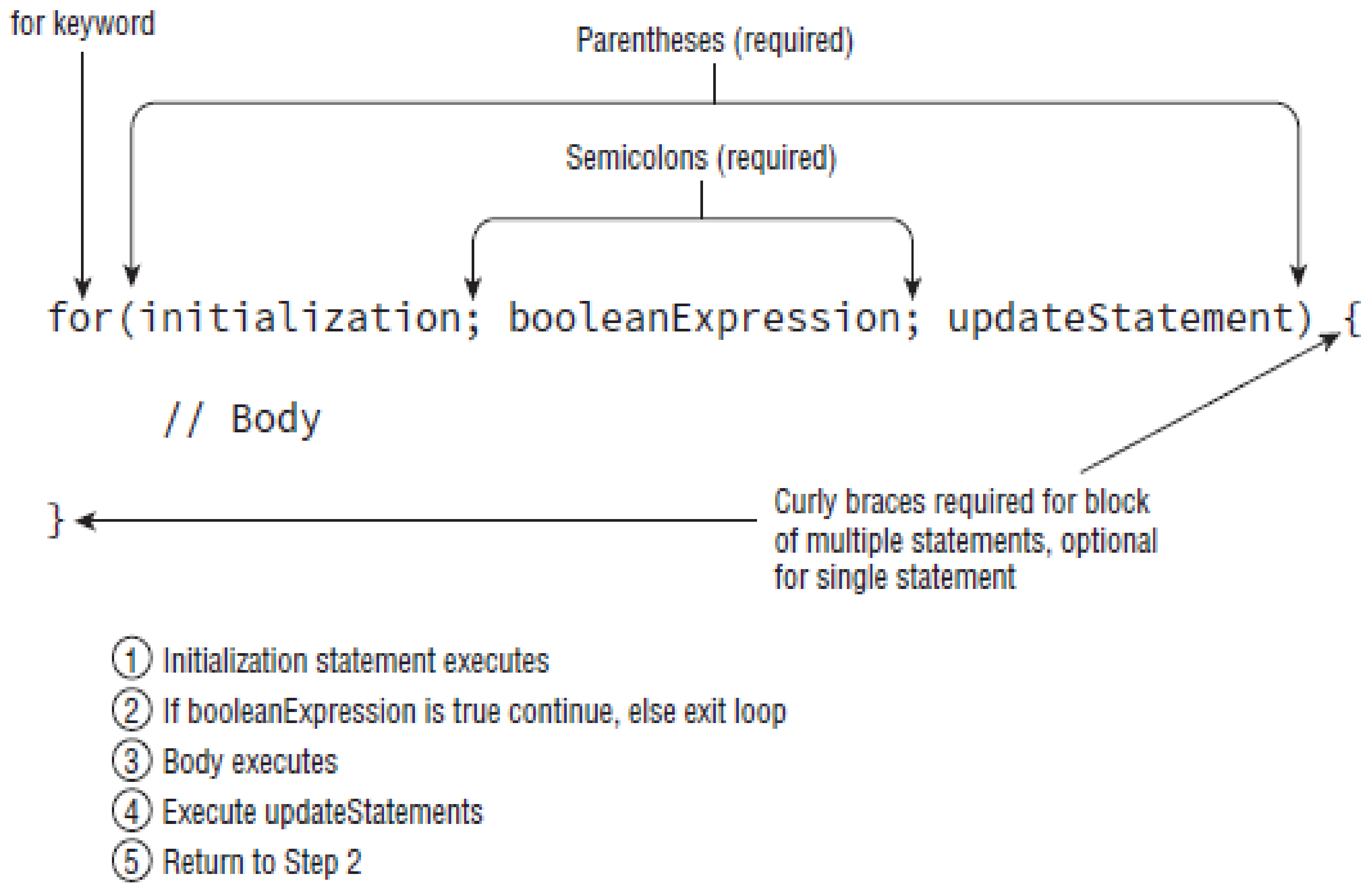

while keyword
Parentheses (required)
while(booleanExpression) {
// Body
}
Curly braces required for block of multiple statements, optional for single statement

The do-while Statement - Highlights

- Unlike a while loop, though, a do-while loop guarantees that the statement or block will be executed at least once.


```
do {

    // Body

} while (booleanExpression);
```
do keyword
Curly braces required for block of multiple statements, optional for single statement
Semicolon (required)
while keyword
Parentheses (required)

The for Statement - Highlights


for keyword
Parentheses (required)
Semicolons (required)
Curly braces required for block of multiple statements, optional for single statement

```
for(initialization; booleanExpression; updateStatement) {

    // Body

}
```

1. Initialization statement executes
2. If booleanExpression is true continue, else exit loop
3. Body executes
4. Execute updateStatements
5. Return to Step 2

The for Statement - Highlights

- Creating an infinite loop

```
for( ; ; ) {
  System.out.println("Hello World");
}
```

- Adding Multiple Terms to the for Statement

```
int x = 0;
for(long y = 0, z = 4; x < 5 && y < 10; x++, y++) {
  System.out.print(y + " ");
}
System.out.print(x);
```

The for Statement - Highlights

- Redeclaring a Variable in the Initialization Block

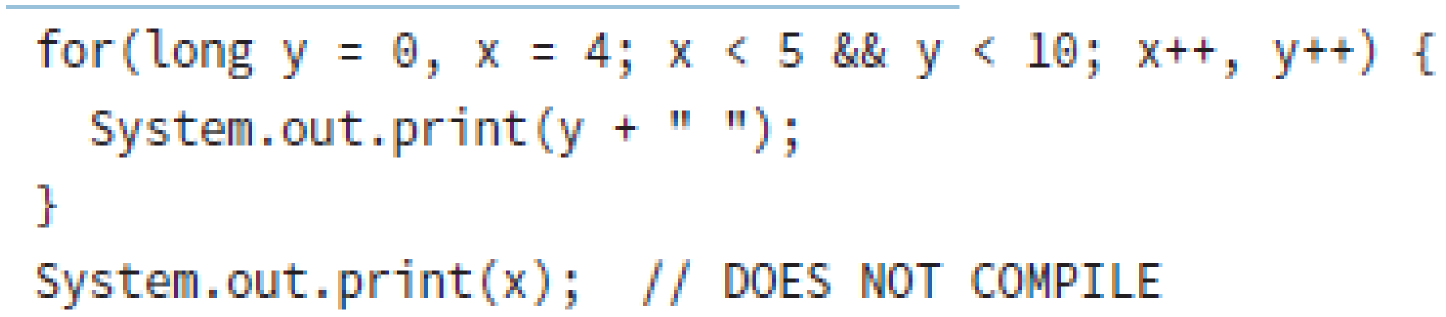
```
int x = 0;
for(long y = 0, x = 4; x < 5 && y < 10; x++, y++) {   // DOES NOT COMPILE
  System.out.print(x + " ");
}
```

- Using Incompatible Data Types in the Initialization Block

```
for(long y = 0, int x = 4; x < 5 && y<10; x++, y++) {   // DOES NOT COMPILE
  System.out.print(x + " ");
}
```

The for Statement - Highlights

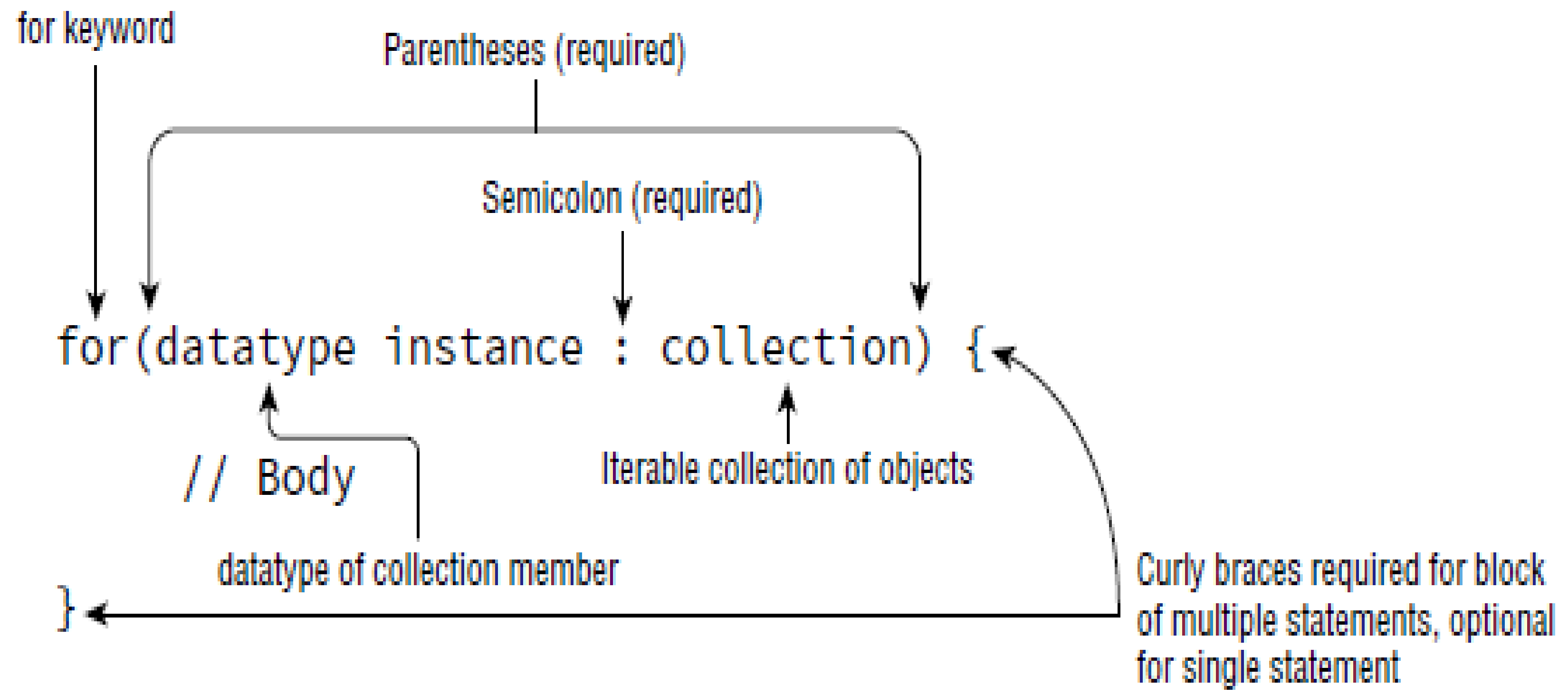
- Using Loop Variables Outside the Loop



```
for(long y = 0, x = 4; x < 5 && y < 10; x++, y++) {
  System.out.print(y + " ");
}
System.out.print(x);  // DOES NOT COMPILE
```

The for-each Statement - Highlights

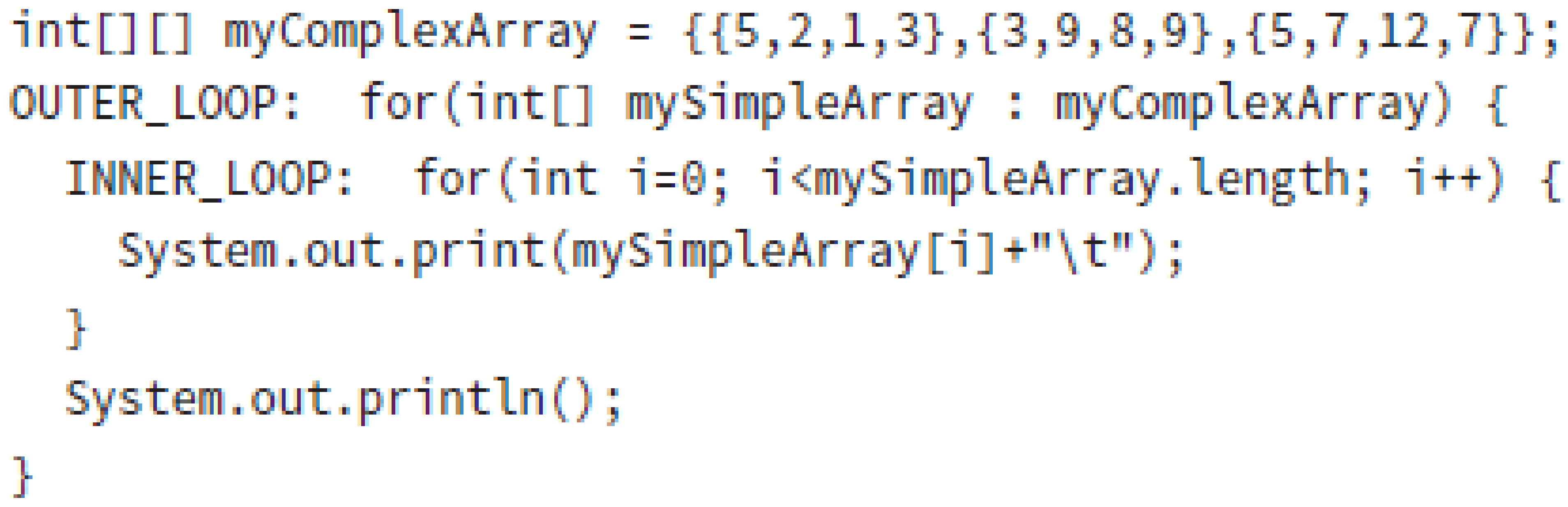
- one specifically designed for iterating over arrays and Collection objects.


for keyword
Parentheses (required)
Semicolon (required)
for(datatype instance : collection) {
// Body
}
datatype of collection member
Iterable collection of objects
Curly braces required for block of multiple statements, optional for single statement

Adding Optional Labels - Highlights

- A label is an optional pointer to the head of a statement that allows the application flow to jump to it or break from it.
- The fact is if-then statements, switch statements, and loops, they can all have optional labels.

Adding Optional Labels - Highlights

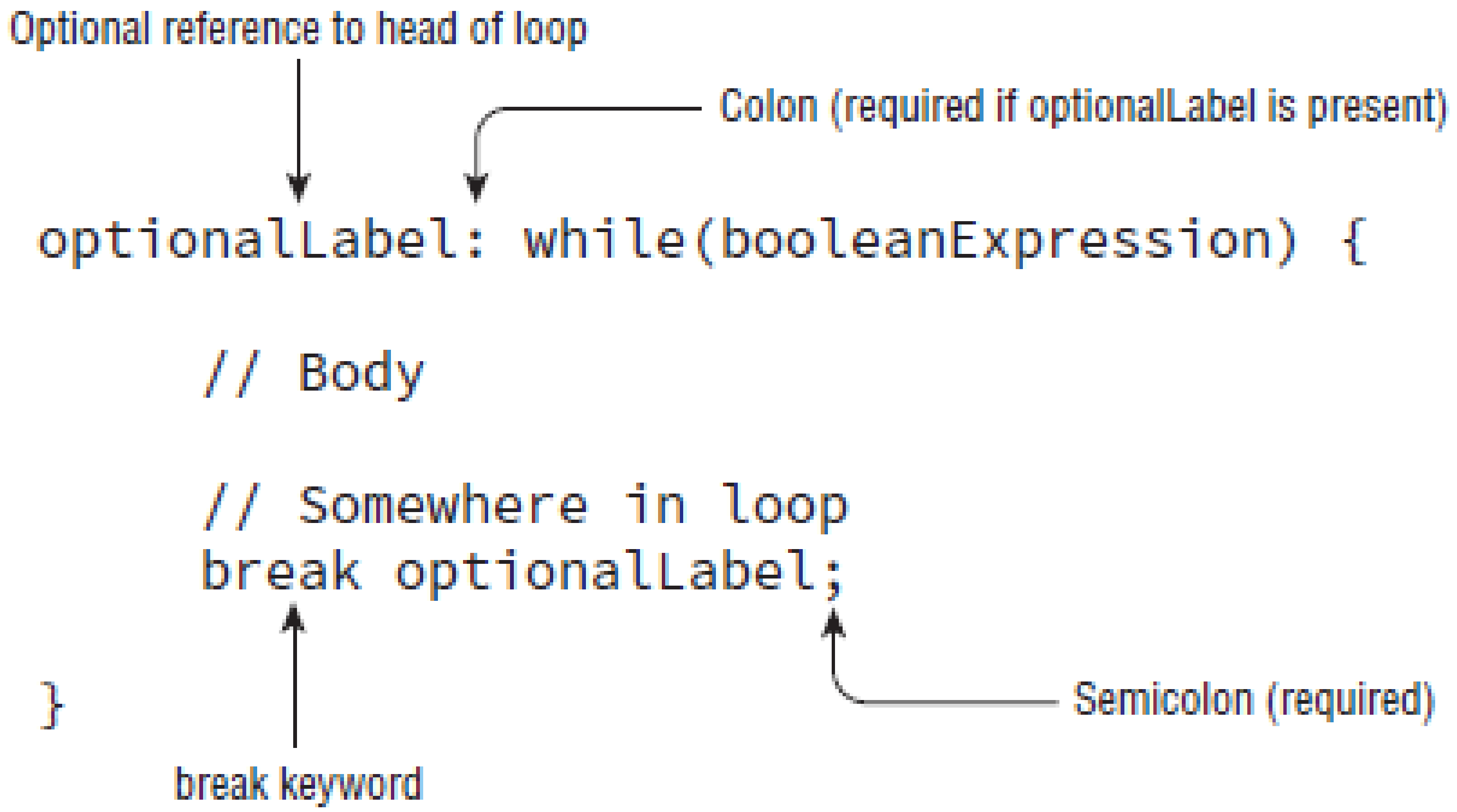


```
int[][] myComplexArray = {{5,2,1,3},{3,9,8,9},{5,7,12,7}};
OUTER_LOOP:  for(int[] mySimpleArray : myComplexArray) {
  INNER_LOOP:  for(int i=0; i<mySimpleArray.length; i++) {
    System.out.print(mySimpleArray[i]+"\t");
  }
  System.out.println();
}
```

The break Statement - Highlights

- As you saw when working with switch statements, a break statement transfers the flow of control out to the enclosing statement. The same holds true for break statements that appear inside of while, do-while, and for loops, as it will end the loop early.

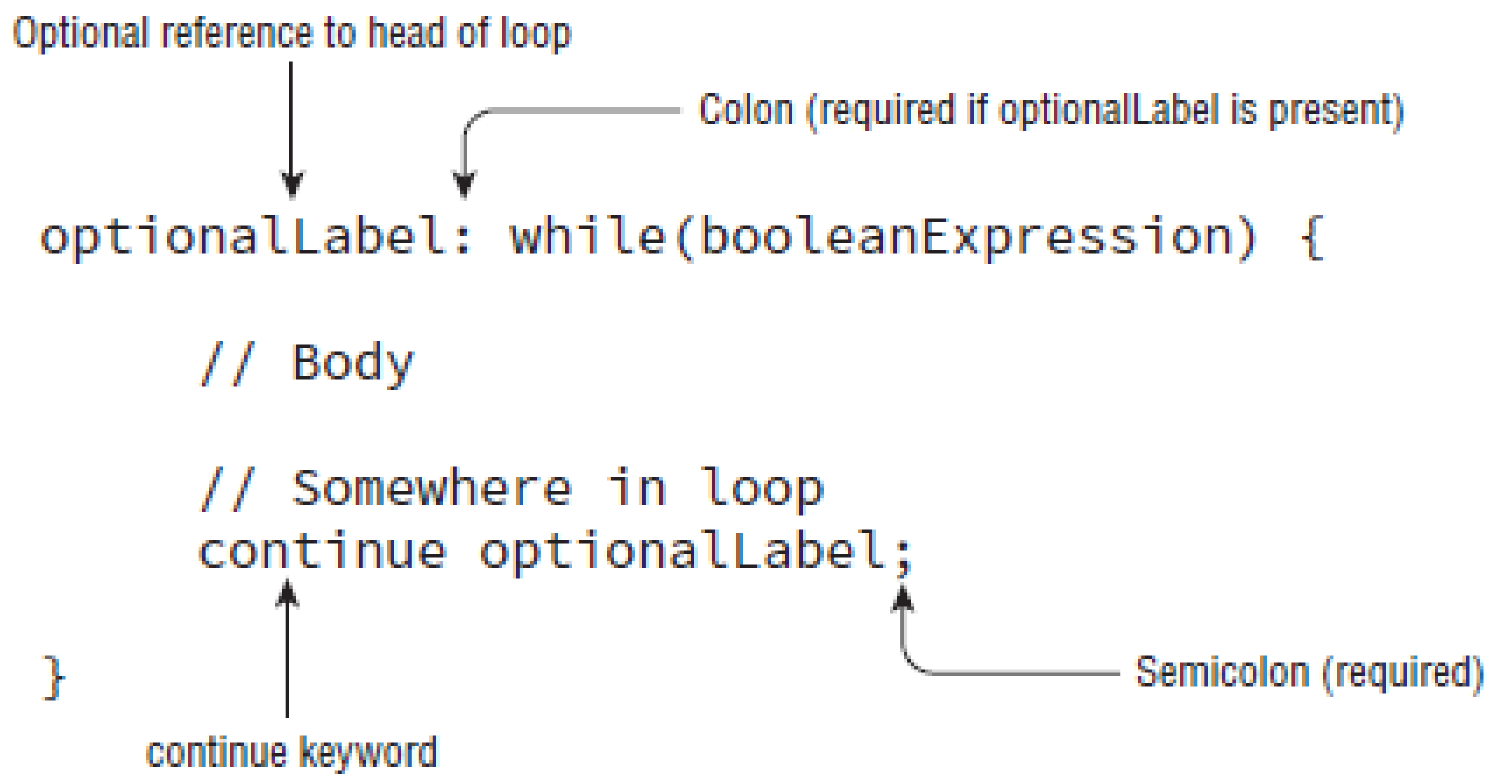
The break Statement - Highlights


Optional reference to head of loop
Colon (required if optionalLabel is present)
break keyword
Semicolon (required)

```
optionalLabel: while(booleanExpression) {

     // Body

     // Somewhere in loop
     break optionalLabel;

}
```

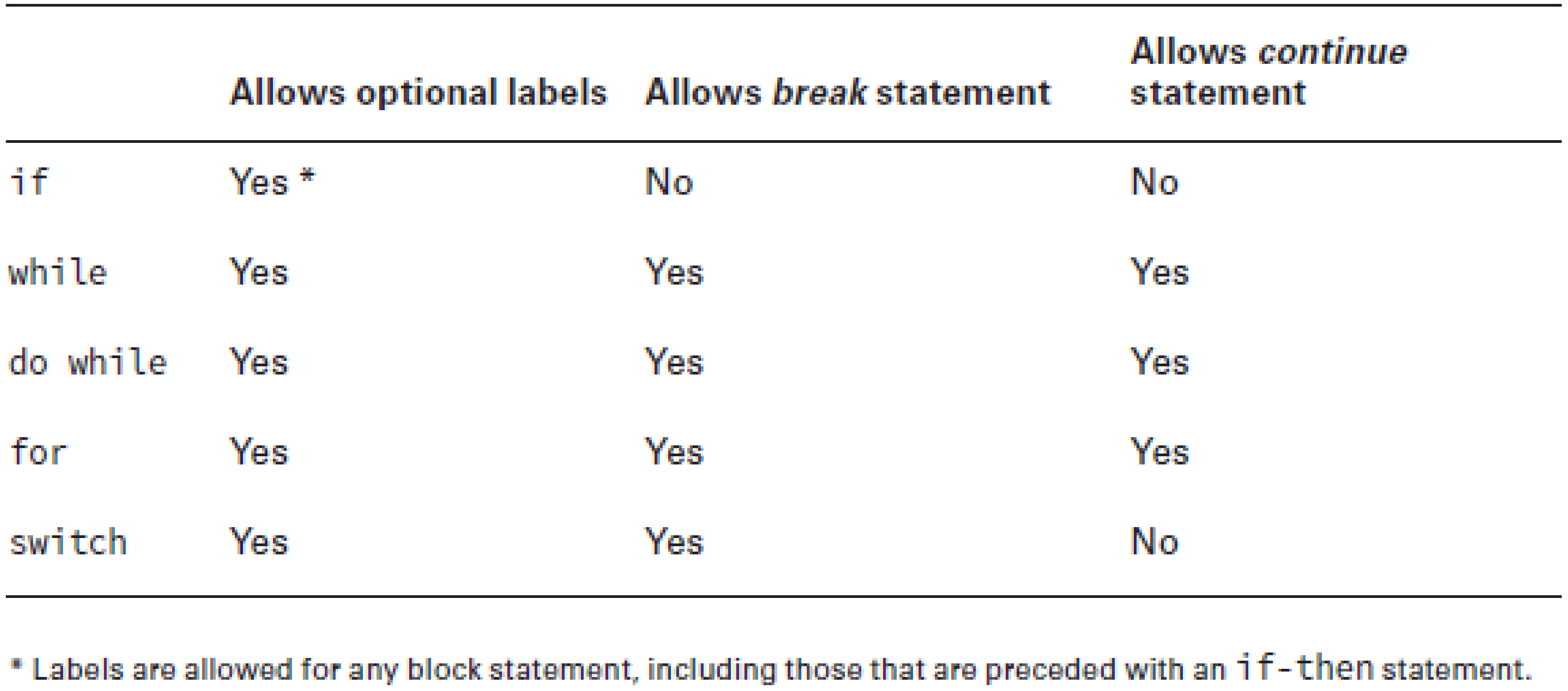
The continue Statement - Highlights

- continue statement, a statement that causes flow to finish the execution of the current loop.


Optional reference to head of loop
Colon (required if optionalLabel is present)
Semicolon (required)
continue keyword

```
optionalLabel: while(booleanExpression) {

    // Body

    // Somewhere in loop
    continue optionalLabel;

}
```

Advanced flow control usage - Highlights



	Allows optional labels	Allows *break* statement	Allows *continue* statement
`if`	Yes *	No	No
`while`	Yes	Yes	Yes
`do while`	Yes	Yes	Yes
`for`	Yes	Yes	Yes
`switch`	Yes	Yes	No

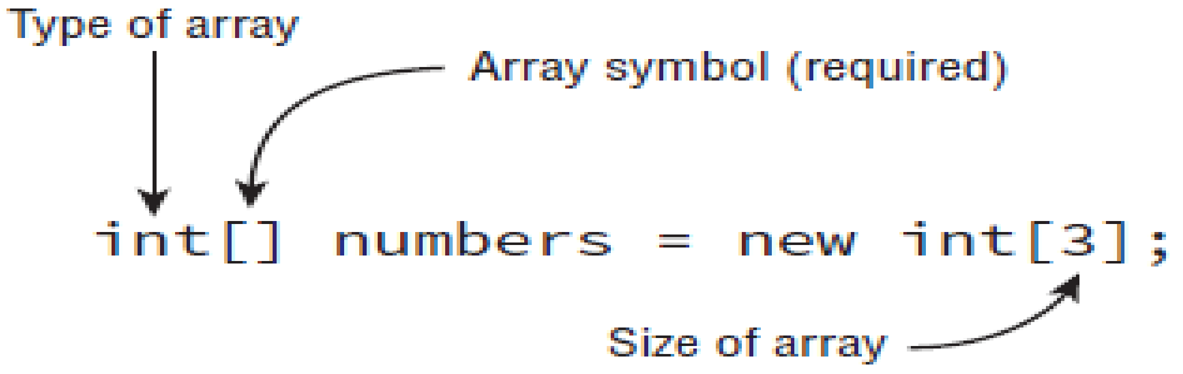
* Labels are allowed for any block statement, including those that are preceded with an `if-then` statement.

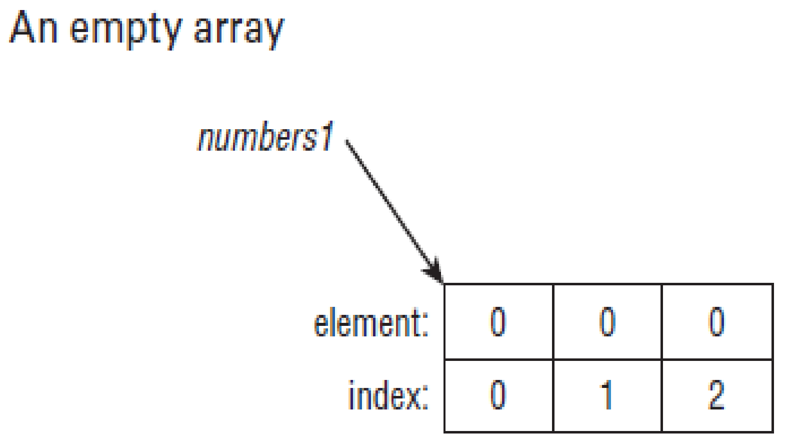
Understanding Java Arrays - Highlights

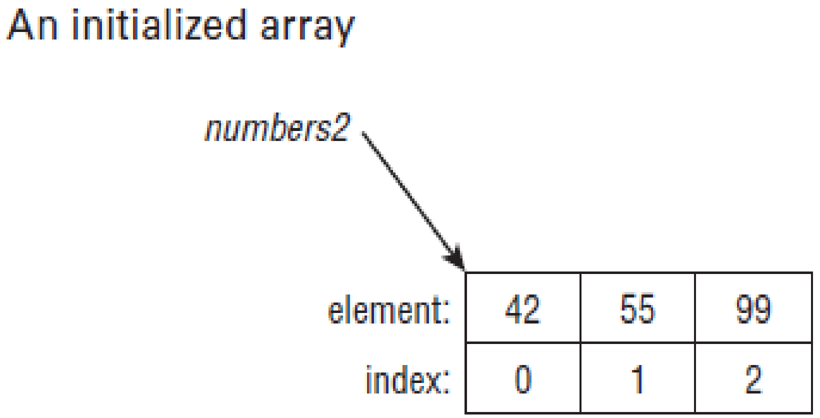
- An array is an area of memory on the heap with space for a designated number of elements.
- A String is implemented as an array with some methods that you might want to use when dealing with characters specifically.

Creating an Array of Primitives - Highlights

- int[] numbers1 = new int[3];
- int[] numbers2 = new int[] {42, 55, 99};
- int[] numbers2 = {42, 55, 99};







Arrays - Highlights

- Using an Array

```
for (int i = 0; i <= numbers.length; i++) numbers[i] = i + 5;
```

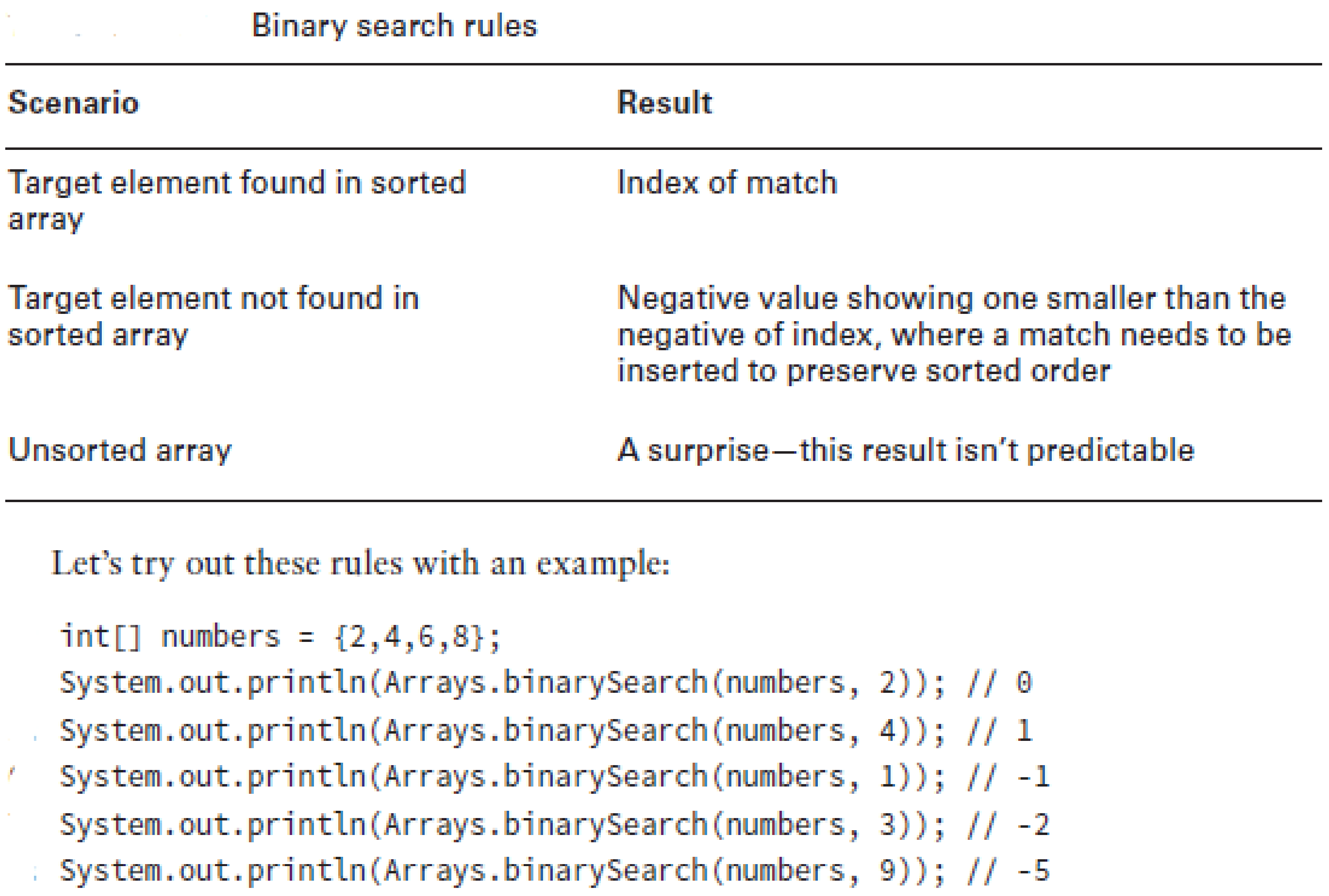
- Sorting

```
int[] numbers = { 6, 9, 1 };
Arrays.sort(numbers);
for (int i = 0; i < numbers.length; i++)
```

- Searching
  - Java also provides a convenient way to search—but only if the array is already sorted.

Arrays - Highlights

- Searching



Binary search rules

Scenario	Result
Target element found in sorted array	Index of match
Target element not found in sorted array	Negative value showing one smaller than the negative of index, where a match needs to be inserted to preserve sorted order
Unsorted array	A surprise—this result isn't predictable

Let's try out these rules with an example:

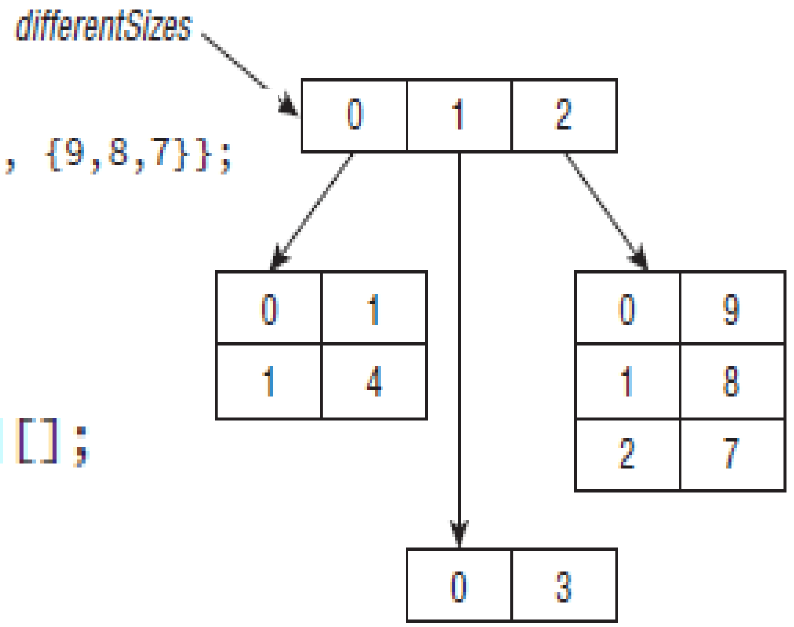
```
int[] numbers = {2,4,6,8};
System.out.println(Arrays.binarySearch(numbers, 2)); // 0
System.out.println(Arrays.binarySearch(numbers, 4)); // 1
System.out.println(Arrays.binarySearch(numbers, 1)); // -1
System.out.println(Arrays.binarySearch(numbers, 3)); // -2
System.out.println(Arrays.binarySearch(numbers, 9)); // -5
```

Multidimensional Arrays - Highlights

```
int[][] vars1;           // 2D array
int vars2 [][];          // 2D array
int[] vars3[];           // 2D array
int[] vars4 [], space [][];  // a 2D AND a 3D array
```

An asymmetric multidimensional array

```
int[][] differentSize = {{1, 4}, {3}, {9,8,7}};
```



```
int [][] args = new int[4][];
args[0] = new int[5];
args[1] = new int[3];
```

Using a Multidimensional Array - Highlights

```
int[][] twoD = new int[3][2];
for (int i = 0; i < twoD.length; i++) {
  for (int j = 0; j < twoD[i].length; j++)
    System.out.print(twoD[i][j] + " "); // print element
  System.out.println();                 // time for a new row
}
```

```
for (int[] inner : twoD) {
  for (int num : inner)
    System.out.print(num + " ");
   System.out.println();
}
```

Summary

- Sequence Control
- Selection Control
  - *if* Statement
  - *if – else* Statement
  - Nested *if* Statement
  - *switch* Statement
- Repetition Control
  - *while* Statement
  - *do – while* Statement
  - *for* Statement
    - Enhanced *for* Statement
- Branching Statements
  - *Break*
  - *Continue*
- Arrays
  - Declaring Arrays
  - Creating Arrays
  - Initializing Array values
  - Single and Multidimensional Arrays
  - Advantages/Disadvantages

References

- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/flow.html
- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

- How To Program (Early Objects)
  - By H .Deitel and P. Deitel
- Headfirst Java
  - By Kathy Sierra and Bert Bates

Questions ???

Thank You