Vietnam National University of HCMC
International University
School of Computer Science and Engineering

Control Flow Statement & Array

(IT069IU)

Nguyen Trung Ky

ntky@hcmiu.edu.vn

it.hcmiu.edu.vn/user/ntky

Previously,

This lecture, we have learnt about:

- **Class**
  - **Attributes**
  - **Method** with Parameters
    - **Getters** and **Setters** Methods
  - **Access Modifiers** (Public & Private)
  - **Constructor** with Parameters
  - **UML** Diagram
- **Object**
  - **Create objects** from class with keyword **new**
  - Call **methods** with input arguments of the object
- **Primitive** vs **Reference**
- **Useful Classes**
  - **Scanner**
    - Read input string with nextLine(), next()
    - Read input number with nextDouble()
  - **String**
    - Display string with print(), println(), printf()
  - **Math**
    - pow(), max(), random()
- Bank account application example

Agenda today

- **Control flow statements**:
  - Decision making statements:
    - **If**
    - **If...else**
    - **Switch**
  - Loop statements:
    - **While**
    - **Do while**
    - **For**
  - Jump statements:
    - **Break** statement
    - **Continue** statement
- **Array:**
  - Declare and Create Array
  - Loop through Array
  - Pass Arrays to Methods
    - Pass by Value vs Pass by Reference
  - Class Arrays for helper methods

Control Flow Statements

Decision Making Statements

If, Else, Switch

If Statement

- **Three** types of decision making statements:
  - **if** statement:
    - Performs an action, if a condition is true; skips it, if false.
    - **Single-selection statement**—selects or ignores a single action (or group of actions).
  - **if...else** statement:
    - Performs an action if a condition is true and performs a different action if the condition is false.
    - **Double-selection statement**—selects between two different actions (or groups of actions).
  - **switch** statement
    - Performs one of several actions, based on the value of an expression.
    - **Multiple-selection statement**—selects among many different actions (or groups of actions).

If Single-Selection Example

- Pseudocode:

```
If student's grade is greater than or equal to 60
    Print "Passed"
```

- If the condition is false, the Print statement is ignored.
- Indentation:
  - Optional, but recommended
  - Emphasizes the inherent structure of structured programs
- For single statement, { } can be omitted.
- Java code:

```
if ( studentGrade >= 60 )
   System.out.println( "Passed" );
```

```
if ( studentGrade >= 60 ){
   System.out.println( "Passed" );
}
```

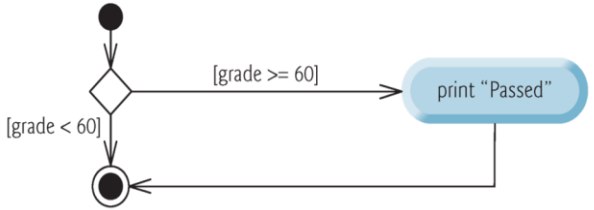


Fig. 4.2 | `if` single-selection statement UML activity diagram.

If Double-Selection Example

- Pseudocode:

```
If student's grade is greater than or equal to 60
        Print "Passed"
Else
        Print "Failed"
```

- Specify an action to perform when the condition is true and a different action when the condition is false.
- Java code:

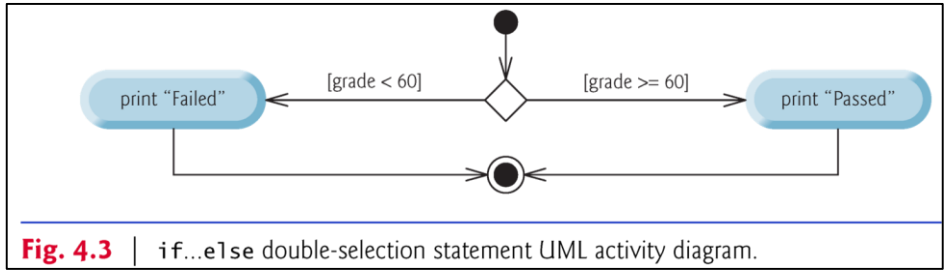


Fig. 4.3 | `if...else` double-selection statement UML activity diagram.

```
if ( grade >= 60 )
   System.out.println( "Passed" );
else
   System.out.println( "Failed" );
```

```
if ( grade >= 60 ){
   System.out.println( "Passed" );
} else{
   System.out.println( "Failed" );
}
```

Ternary Operator - If...Else short form

- Conditional operator (?:)—**shorthand if...else.**
- Ternary operator (takes three operands)
- Only suitable for simple comparison condition and make sure the conditional expression to be easy to understand.

```
System.out.println( grade >= 60 ? "Passed" : "Failed" );
```

Nested If ... Else Statements

- Both styles are corrected. But most Java developers prefer the right one.

```
if ( studentGrade >= 90 )
   System.out.println( "A" );
else
   if ( studentGrade >= 80 )
      System.out.println( "B" );
   else
      if ( studentGrade >= 70 )
         System.out.println( "C" );
      else
         if ( studentGrade >= 60 )
            System.out.println( "D" );
         else
            System.out.println( "F" );
```

```
if ( studentGrade >= 90 )
   System.out.println( "A" );
else if ( studentGrade >= 80 )
   System.out.println( "B" );
else if ( studentGrade >= 70 )
   System.out.println( "C" );
else if ( studentGrade >= 60 )
   System.out.println( "D" );
else
   System.out.println( "F" );
```

This is confusing!

Without braces { }, The Java compiler always **associates an else** with the **nearest if before it**.
Referred to as the dangling-else problem.

It looks like:

```
if ( x > 5 )
        if ( y > 5 )
                System.out.println( "x and y are > 5" );
else
        System.out.println( "x is <= 5" );
```

It actually is:

```
if ( x > 5 )
        if ( y > 5 )
                System.out.println( "x and y are > 5" );
        else
                System.out.println( "x is <= 5" );
```

No more confusing! Use braces {} for nest IF!

[Question] the else statement belongs to which if in the first box and second box?

```
if ( x > 5 ){
        if ( y > 5 )
                System.out.println( "x and y are > 5" );
        else
                System.out.println( "x is <= 5" );
}
```

```
if ( x > 5 ){
        if ( y > 5 )
                System.out.println( "x and y are > 5" );
} else
System.out.println( "x is <= 5" );
```

Multiple statements in an IF block

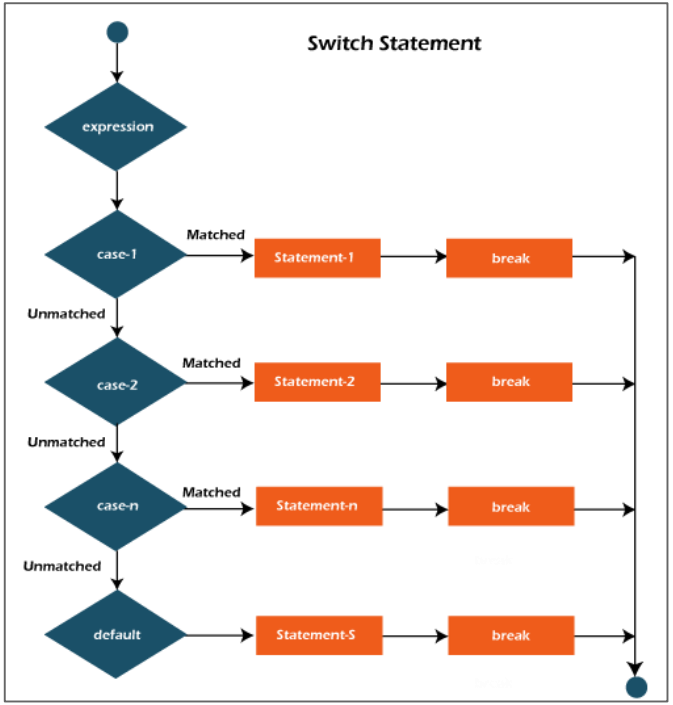
- The if statement normally expects only one statement in its body.
- To **include several statements** in the body of an if (or the body of an else for an if…else statement), **enclose the statements in braces { }.**
- Statements contained in a pair of braces form **a block.**
- A block can be placed anywhere that a single statement can be placed.

```
if ( grade >= 60 )
        System.out.println("Passed");
else {
        System.out.println("Failed");
        System.out.println("You must take this course again.");
}
```

Switch Statement

- To select one of many code blocks to be executed.
- The **value of expression** is **compared** to the **values of each case.**
- If matched, the block of that case is executed.
- **break** keyword will breaks out of the whole block.
- **default** keyword define a default block will run at the end if no case is matched.

```
switch(expression) {
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
}
```



Switch Example

```java
int day = 4;
switch (day) {
    case 6:
        System.out.println("Today is Saturday");
        break;
    case 7:
        System.out.println("Today is Sunday");
        break;
    default:
        System.out.println("Looking forward to the Weekday!");
}
```

[Question]

- What happens if you remove **“break”** keyword in one case?
- What happens if grade is **“B”**?

```java
char grade = 'C';
switch(grade) {
    case 'A' :
        System.out.println("Excellent!");
        break;
    case 'B' :
    case 'C' :
        System.out.println("Well done");
        break;
    case 'D' :
        System.out.println("You passed");
    case 'F' :
        System.out.println("Better try again");
        break;
    default :
        System.out.println("Invalid grade");
}
System.out.println("Your grade is " + grade);
```

Loop Statements

While, Do...While, For

Loop Statements

- **Repetition statements** (also called **looping statements**)
- Perform statements repeatedly while **a loop-continuation condition** remains true.
- **while** and **for** statements perform the action(s) in their bodies **zero or more times.**
  - if the loop-continuation condition is initially false, the body will not execute.
- The **do…while** statement performs the action(s) in its body **one or more times.**

While Statement

- Pseudocode

```
While there are more items on my shopping list
    Buy next item and cross it off my list
```

- **Repetition statement—repeats an action while a condition remains true.**
- The repetition statement's body may be a single statement or a block.
- Eventually, **the condition will become false. At this point, the repetition terminates**, and the statements after while block will continue.

```
while (boolean condition)
    single_statement;
```

```
while (boolean condition){
        statement_1;
    statement_2;
    …
    statement_n;
}
```

While Loop Example

```
int product= 1;
while ( product <= 100)
      product = product*3;
System.out.print(product);
```

[**Question**] What is the value of product when the while loop finishes)?

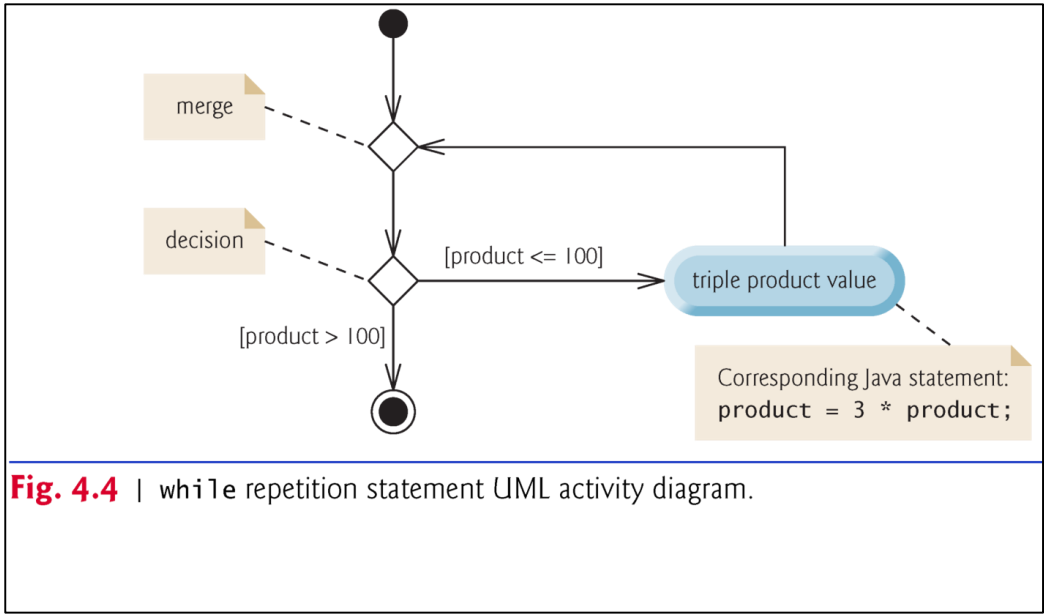


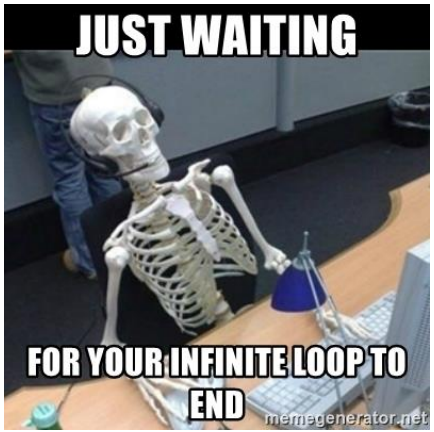
Fig. 4.4 | `while` repetition statement UML activity diagram.

Danger of Infinite Loop

- Beware about your stopping condition of any loop.

```
int product= 1;
while ( product >=0)
      product = product*3;
System.out.print(product);
```

[Question] What is the value of product when the while loop finishes)?



Analysis Class

- Write an Analysis Class:
  - Let instructor to input performance status of 10 student (pass or fail)
  - Output the number of students passed and failed
  - If the number of students passed equal or larger than 9 then give instructor an bonus.

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Bonus to instructor!
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 6
Failed: 4
```

Analysis.java

```
1  // Fig. 4.12: Analysis.java
2  // Analysis of examination results using nested control statements.
3  import java.util.Scanner; // class uses class Scanner
4
5  public class Analysis
6  {
7     public static void main( String[] args )
8     {
9        // create Scanner to obtain input from command window
10       Scanner input = new Scanner( System.in );
11
12       // initializing variables in declarations
13       int passes = 0; // number of passes
14       int failures = 0; // number of failures
15       int studentCounter = 1; // student counter
16       int result; // one exam result (obtains value from user)
17
```

Analysis.java

```
18          // process 10 students using counter-controlled loop
19          while ( studentCounter <= 10 )
20          {
21             // prompt user for input and obtain value from user
22             System.out.print( "Enter result (1 = pass, 2 = fail): " );
23             result = input.nextInt();
24
25             // if...else is nested in the while statement
26             if ( result == 1 )           // if result 1,
27                passes = passes + 1;      // increment passes;
28             else                         // else result is not 1, so
29                failures = failures + 1; // increment failures
30
31             // increment studentCounter so loop eventually terminates
32             studentCounter = studentCounter + 1;
33          } // end while
34
```

Analysis.java

```
35          // termination phase; prepare and display results
36          System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );
37
38          // determine whether more than 8 students passed
39          if ( passes > 8 )
40             System.out.println( "Bonus to instructor!" );
41       } // end main
42    } // end class Analysis
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Bonus to instructor!
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 6
Failed: 4
```

Do...While Loop

- Instead of checking loop condition at the start, do…while will **check loop condition at the end.**
- To ensure the **loop block will start at least once** even if the loop condition is even false at the beginning.

```
1  // Fig. 5.7: DoWhileTest.java
2  // do...while repetition statement.
3
4  public class DoWhileTest
5  {
6     public static void main( String[] args )
7     {
8        int counter = 1; // initialize counter
9
10       do
11       {
12          System.out.printf( "%d  ", counter );
13          ++counter;
14       } while ( counter <= 10 ); // end do...while
15
16       System.out.println(); // outputs a newline
17    } // end main
18 } // end class DoWhileTest
```

Condition tested at end of loop, so loop always executes at least once

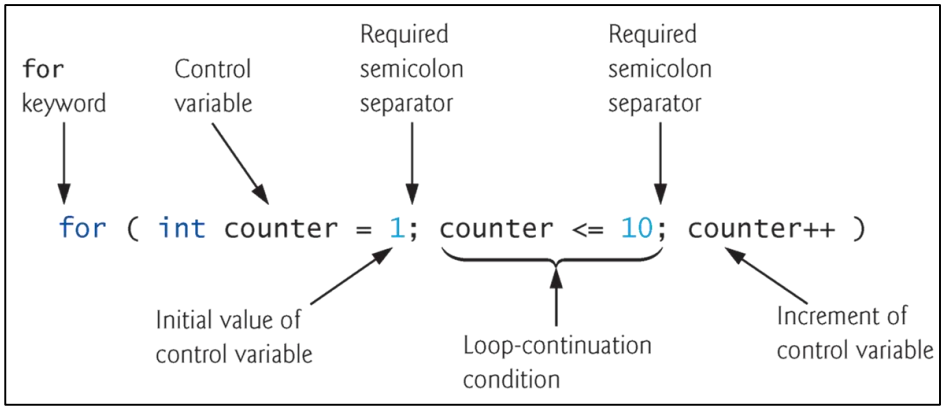
```
1  2  3  4  5  6  7  8  9  10
```

Fig. 5.7 | do...while repetition statement.

For Loop - Counter-Controlled Repetition

- For Loop is for when we **know exactly how many times we want to loop** through a block of code. (While is for when we only know stopping condition).
- **initial_statement: executed (one time)** at the beginning.
- **stop_condition: loop condition** to allow the loop to continue.
- **Update_statement**: **executed (every time)** after the code block has been executed.

```
for (initial_statement; stop_condition; update_statement) {
  // code block to be executed
}
```



For Loop Example

```
for (int i = 0; i < 5; i+=1) {
      System.out.println(i);
}
```

```
for (int i = 0; i <= 10; i=i + 2) {
   System.out.println(i);
}
```

[**Question**] What is the output for these two code?

For Loop Example

Sum.java

```
1  // Fig. 5.5: Sum.java
2  // Summing integers with the for statement.
3
4  public class Sum
5  {
6     public static void main( String[] args )
7     {
8        int total = 0; // initialize total
9
10       // total even integers from 2 through 20
11       for ( int number = 2; number <= 20; number += 2 )
12          total += number;
13
14       System.out.printf( "Sum is %d\n", total ); // display results
15    } // end main
16 } // end class Sum
```

```
Sum is 110
```

Fig. 5.5 | Summing integers with the `for` statement.

Nested For Loop

- If a loop exists inside the body of another loop, it's called a nested loop.

```java
int weeks = 3;
int days = 7;

// outer loop prints weeks
for (int i = 1; i <= weeks; ++i) {
    System.out.printf("Week %d: ",i);

    // inner loop prints days
    for (int j = 1; j <= days; ++j) {
        System.out.printf("%d ",j);
    }
    System.out.println();
}
```

```java
int weeks = 3;
int days = 7;
int i = 1;

// outer loop prints weeks
while (i<= weeks){
    System.out.printf("Week %d: ",i);

    // inner loop prints days
    for (int j = 1; j <= days; ++j) {
        System.out.printf("%d ",j);
    }
    i+=1;
    System.out.println();
}
```

```
Week 1: 1 2 3 4 5 6 7
Week 2: 1 2 3 4 5 6 7
Week 3: 1 2 3 4 5 6 7
```

- These two code uses different ways of loopings but the output is the same!

Jump Statements

Break & Continue

Keyword "break"

- Can break out of any type loop (while, do…while, for, switch)

```
for (int i = 1; i<=10; i+=1){
    if (i==7){
        break;
    }
    System.out.println(i);
}
```

```
int i = 1;
do{
    if (i == 7) {
        break;
    }
    System.out.println(i);
    i+=1;
}while (i <= 10);
```

[Question] What is the output for each loop?

```
int i = 1;
while (i <= 10) {
    if (i == 7) {
        break;
    }
    System.out.println(i);
    i+=1;
}
```

“break” in nested loop

- Each break keyword can only break out of one layer of loop.

```
int weeks = 3;
int days = 7;
// outer loop prints weeks
for (int i = 1; i <= weeks; ++i) {
    System.out.printf("Week %d: ",i);

    // inner loop prints days
    for (int j = 1; j <= days; ++j) {
        if (j == 4)
            break;
        System.out.printf("%d ",j);
    }
    System.out.println();
}
```

```
Week 1: 1 2 3
Week 2: 1 2 3
Week 3: 1 2 3
```

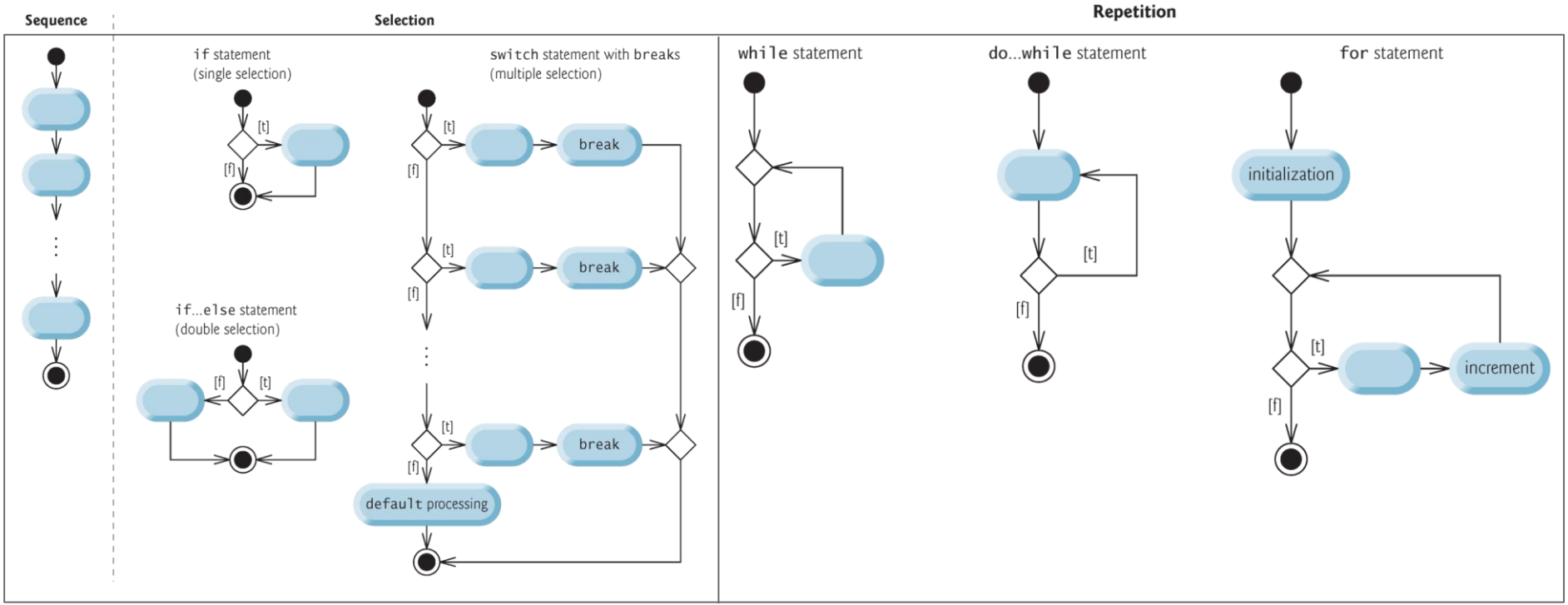
Keyword “continue”

- “Continue” will skips the remaining statements in the loop body and proceeds with the next iteration of the loop.

```
1  // Fig. 5.14: ContinueTest.java
2  // continue statement terminating an iteration of a for statement.
3  public class ContinueTest
4  {
5     public static void main(String[] args)
6     {
7        for (int count = 1; count <= 10; count++) // loop 10 times
8        {
9           if (count == 5)
10             continue; // skip remaining code in loop body if count is 5
11
12          System.out.printf("%d ", count);
13       }
14
15       System.out.printf("%nUsed continue to skip printing 5%n");
16    }
17 } // end class ContinueTest
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

Control & Loop Flow Diagram



Let's Wrap Up! GradeBook Class!

- Write an GradeBook Class:
  - Let instructor to input integer value for grade of any number of student in a class. (Only stopped if he/she input -1)
  - Output the number of grades & the total sum.
  - Also, calculate the average of grades for the whole class.

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of the 3 grades entered is 257
Class average is 85.67
```

Array

Store multiple items of the same type

- Imagine you have a high score board in a game, with 10 high scores on it. Using only the tools we know so far, you can create one variable for every score you want to keep track of:

```
int score1 = 100;
int score2 = 95;
int score3 = 86;
// more and more ...
```

- What if you had 10,000 scores? Too many variable!!!

Array is here to save you!

- with 10 high scores, you want to keep track of:

```
int score1 = 100;
int score2 = 95;
int score3 = 86;
// more and more ...
```

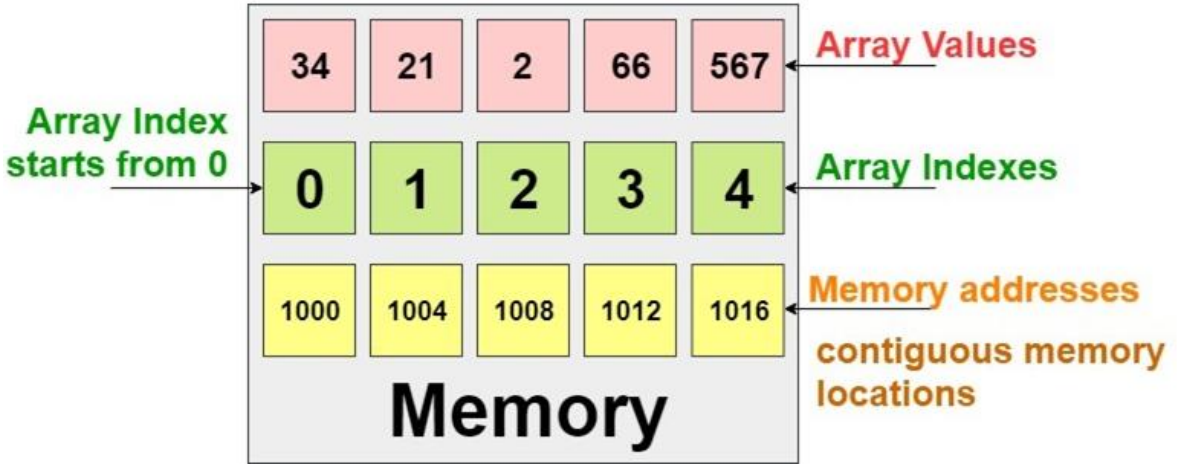


```
int[] scores = new int[10];
```

Array

- An array is **a group of variables** (called elements or components) containing **values of the same type.**
- Arrays are **objects**, so they're considered **reference types.**

```
int[ ] x = new int[5];
x[0] = 34;
x[1] = 21;
x[2] = 2;
x[3] = 66;
x[4] = 567;
```



Array Declaration Syntax

- Array Declaration (array variable points to null):

data_type[] *arrayName*;

- Allocate actual memory for array members:

arrayName = new *data_type*[*size*];

- You can do both steps in one line:

data_type[] *arrayName* = new *data_type*[*size*];

For Example:

- **you declare a new array with two steps:**

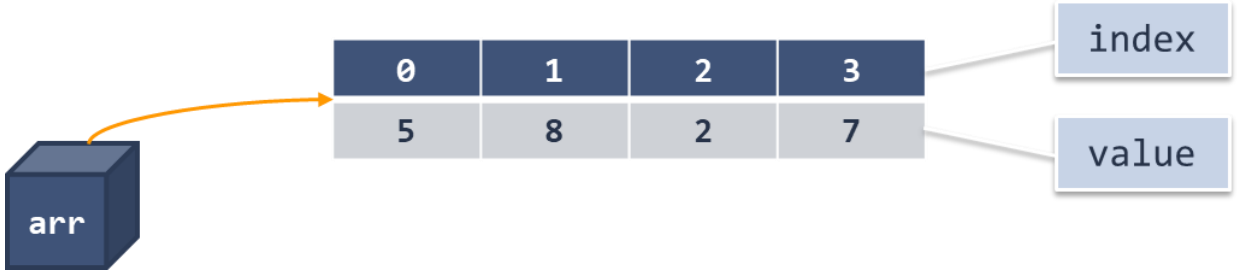
```
int[] c; // declare the array variable
c = new int[12]; // create the array; assign to array variable
```

- **You can declare a new array of 12 integers in one step like this:**

```
int[] c = new int[12];
```

Array Initializer Example

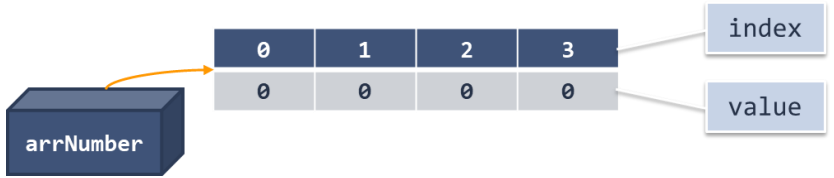
```
int[] arr;
arr = new int[4];
arr[0] = 5;
arr[1] = 8;
arr[2] = 2;
arr[3] = 7;
```



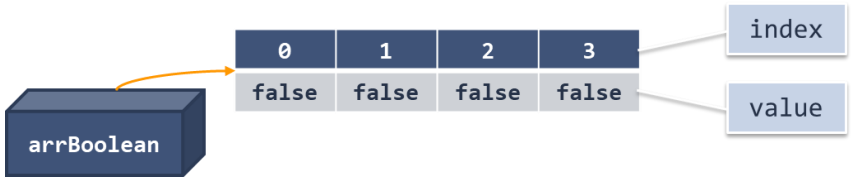
Default Value for Array Element

- When an array is created, each of its elements receives a default value. For example:
  - zero for the numeric primitive-type elements
  - false for boolean elements
  - null for references

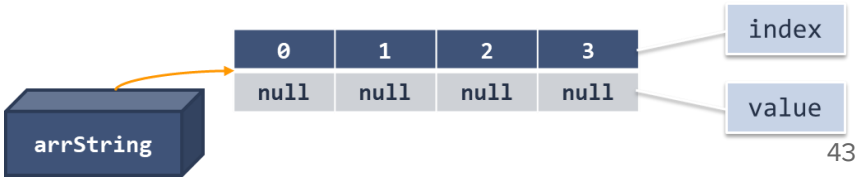
```
int[] arrNumber = new int[4];
```




```
boolean[] arrBoolean = new boolean[4];
```




```
String[] arrString = new String[4];
```



Readability for Array Declaration

- A program **can create several arrays in a single declaration**. The following declaration reserves 100 elements for b and 27 elements for x:

```
String[] b = new String[100], x = new String[27];
```

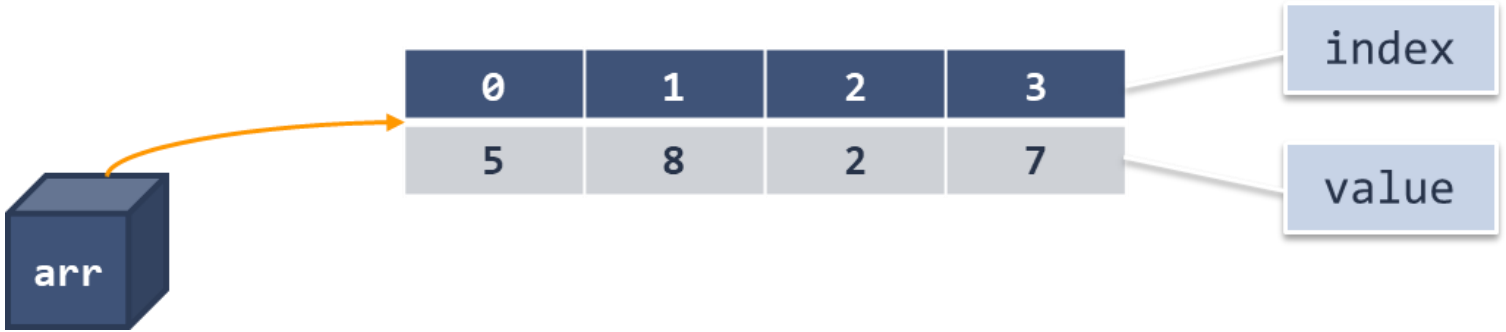
- **For readability, we prefer to declare only one variable at a time**:

```
String[] b = new String[100]; // create array b
String[] x = new String[27]; // create array x
```

Array Initializer

- You can create an array and initialize its elements with an **array initializer.**
- An **array initializer** is a comma-separated list of expressions (called an initializer list) enclosed in braces.

```
int[] arr = {5, 8, 2, 7};
```



Access Element in Array using Index

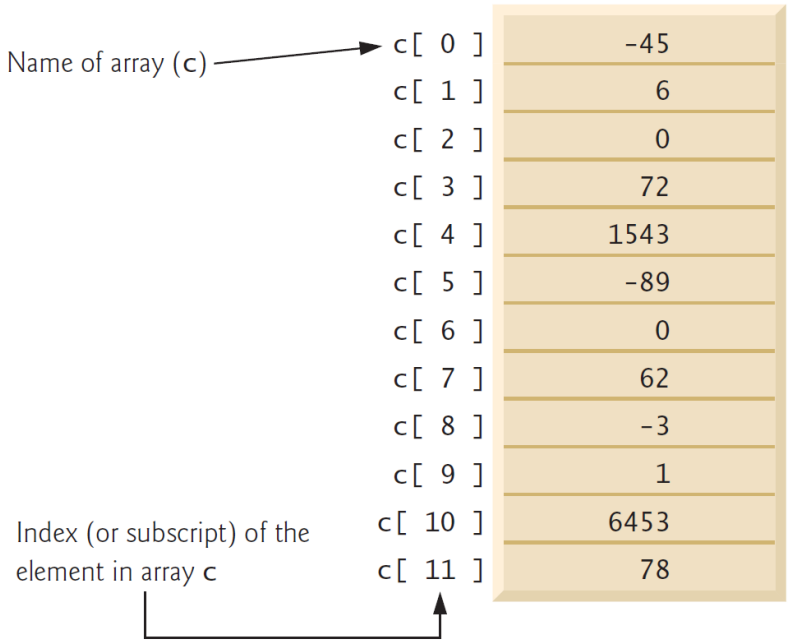
int[ ] c = {-45, 6, 0, 72, 1543, -89, 0, 62, -3, 1, 6453, 78}

```
int a =5, b = 6;

c[a+b] +=2;
```

What will be changed in the array?

```
int sum = c[0] + c[1] + c[2];
```

What is the output of sum?



Array Length Usefulness

```java
int[] scores = new int[5];
System.out.println("The size of is " + scores.length);
System.out.println("Old Array:");
for (int i=0; i<scores.length; i++){
    System.out.printf("%d ",scores[i]);
}
scores[1]=99;
scores[4]=88;
System.out.println("\nNew Array:");
for (int i=0; i<scores.length; i++){
    System.out.printf("%d ",scores[i]);
}
```

Output:

```
The size of is 5
Old Array:
0 0 0 0 0
New Array:
0 99 0 0 88
```

Loop Through 1-Dimensional Array

- Use for loop:

```
int[] myArray= {1, 2, 3, 4, 5, 6};

for (int index=0; index < myArray.length; index++){
        System.out.printf("%d ", myArray[index]);
}
```

The Same Output:

```
1 2 3 4 5 6
```

- Use enhanced for loop:

```
int[] myArray= {1, 2, 3, 4, 5, 6};

for (int element: myArray){
        System.out.printf("%d ",element);
}
```

[Question] What is the advantage and disadvantage of for loop and enhanced for loop to loop through array?

Initialize Array Example

InitArray.java

```
1  // Fig. 7.2: InitArray.java
2  // Initializing the elements of an array to default values of zero.
3
4  public class InitArray
5  {
6     public static void main(String[] args)
7     {
8        // declare variable array and initialize it with an array object
9        int[] array = new int[10]; // create the array object
10
11       System.out.printf("%s%8s%n", "Index", "Value"); // column headings
12
13       // output each array element's value
14       for (int counter = 0; counter < array.length; counter++)
15          System.out.printf("%5d%8d%n", counter, array[counter]);
16    }
17 } // end class InitArray
```

Index	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Array Initializer Example

Index	Value
0	32
1	27
2	64
3	18
4	95
5	14
6	90
7	70
8	60
9	37

InitArray.java

```
1  // Fig. 7.3: InitArray.java
2  // Initializing the elements of an array with an array initializer.
3
4  public class InitArray
5  {
6     public static void main(String[] args)
7     {
8        // initializer list specifies the initial value for each element
9        int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10
11       System.out.printf("%s%8s%n", "Index", "Value"); // column headings
12
13       // output each array element's value
14       for (int counter = 0; counter < array.length; counter++)
15          System.out.printf("%5d%8d%n", counter, array[counter]);
16
17 } // end class InitArray
```

Sum Array Example

```
1  // Fig. 7.4: InitArray.java
2  // Calculating the values to be placed into the elements of an array.
3
4  public class InitArray
5  {
6     public static void main(String[] args)
7     {
8        final int ARRAY_LENGTH = 10; // declare constant
9        int[] array = new int[ARRAY_LENGTH]; // create array
10
11       // calculate value for each array element
12       for (int counter = 0; counter < array.length; counter++)
13          array[counter] = 2 + 2 * counter;
14
15       System.out.printf("%s%8s%n", "Index", "Value"); // column headings
16
17       // output each array element's value
18       for (int counter = 0; counter < array.length; counter++)
19          System.out.printf("%5d%8d%n", counter, array[counter]);
20    }
21 } // end class InitArray
```

Index	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

Enhanced for Statement

- The enhanced for statement **iterates through the elements of an array without using a counter**, thus **avoiding the possibility of "stepping outside" the array.**
- where **parameter** has a **type** and an **identifier** (e.g., int number), and **arrayName** is the **array through which to iterate.**

```
for (parameter : arrayName)
    statement
```

- Instead, using counter in a for loop to loop through an array:

```
for (int counter = 0; counter < array.length; counter++)
   total += array[counter];
```

- We can just use:

```
for (int number : array)
   total += number;
```

Enhanced For Loop Example

```
1  // Fig. 7.12: EnhancedForTest.java
2  // Using the enhanced for statement to total integers in an array.
3
4  public class EnhancedForTest
5  {
6     public static void main(String[] args)
7     {
8        int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for (int number : array)
13          total += number;
14
15       System.out.printf("Total of array elements: %d%n", total);
16    }
17 } // end class EnhancedForTest
```

```
Total of array elements: 849
```

Pass Arrays to Methods

- You can **pass an array to a method** just like **you pass a variable to a method.**

For example:

- If **array hourlyTemperatures** is declared as:

```
double[] hourlyTemperatures = new double[24];
```

- The **method call** can be:

```
modifyArray(hourlyTemperatures);
```

- **Method header** might be written as:

```
void modifyArray(double[] b)
```

Pass-By-Value vs. Pass-By-Reference

```java
public static void main(String[] args) {
    int x = 8;
    System.out.println("X Start: " + x);
    System.out.println("Local X: " + addTwo(x));
    System.out.println("X End: " + x);
}

public static int addTwo(int local_x){
    local_x+= 2;
    return local_x;
}
```

```
X Start: 8
Local X: 10
X End: 8
```

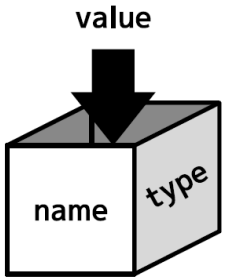
```java
public static void main(String[] args) {
    int[] x = {8};
    System.out.println("X Start: " + x[0]);
    System.out.println("Local X: " + addTwo(x)[0]);
    System.out.println("X End: " + x[0]);
}

public static int[] addTwo(int[] local_x){
    local_x[0] += 2;
    return local_x;
}
```

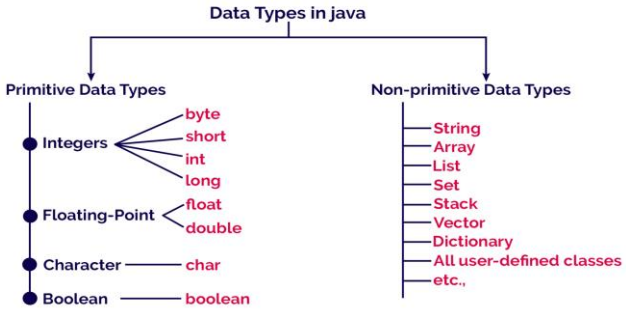
```
X Start: 8
Local X: 10
X End: 10
```

[Question] Which example passes by value to a method or which passes by reference to a method?

Let's revisited: Variable

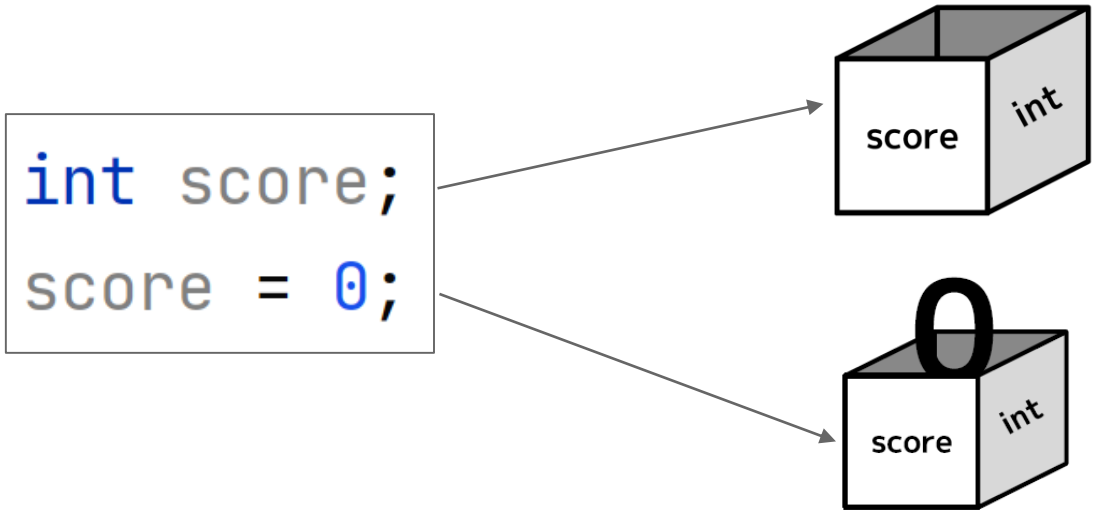
- Remember, a variable is a place in memory where you can store information.
- It's like a little box or bucket to put stuff in.
- Each **variable** has a **name** and a **type**.
- For **primitive** data type:
  - The value of a variable stores the actual value of the data.
- For **reference** data type:
  - The value of a variable stores the memory address (reference) to the location of the data. Similar to pointer address in C.





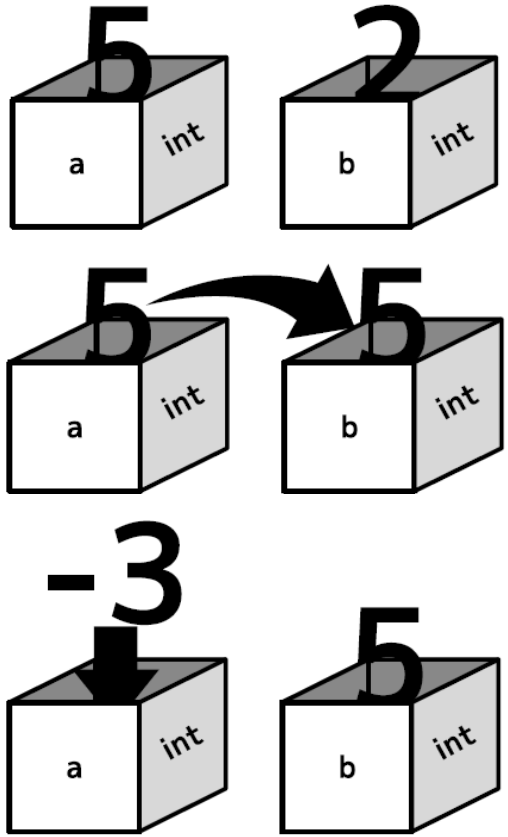
Primitive Data Type

- For **primitive** data type:
  - The value of a variable stores the **actual value of the data.**



Primitive Data Type

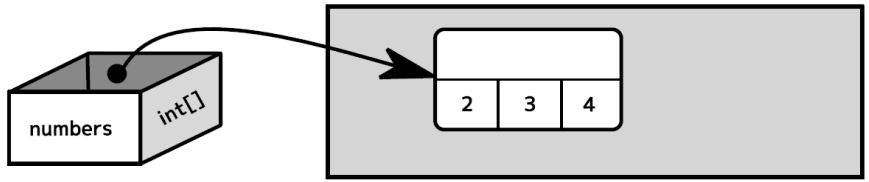
```
int a = 5;
int b = 2;
b = a;
a = -3;

System.out.println(a);
System.out.println(b);
```



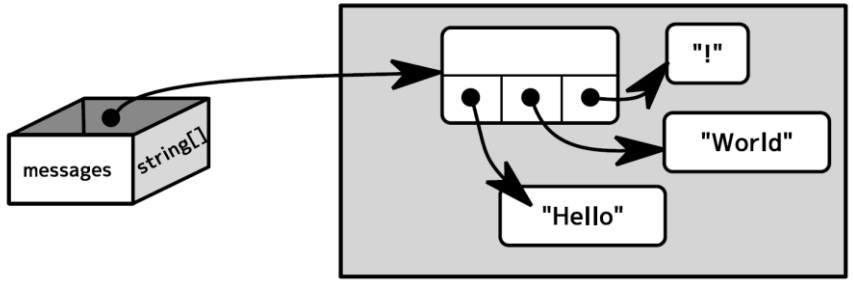
Array is a Reference Data Type

- For **reference** data type:
  - The value of a variable stores the **memory address (reference)** to the location of the data. **Similar to pointer address in C.**

```
int[] numbers =  { 2, 3, 4 };
```




```
String[] messages = {"Hello", "World", "!"};
```

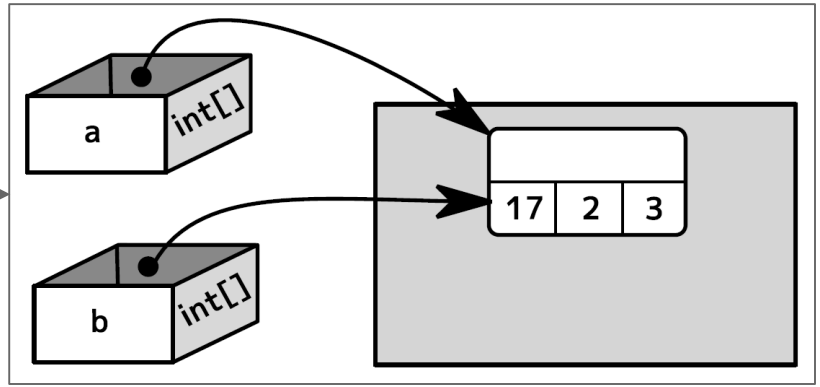


Compared This

```
int a = 3;
int b = a;
b++;
System.out.println(a);
System.out.println(b);
```

[Question] Can you guess what is the output for each case?

```
int[] a = new int[] { 1, 2, 3 };
int[] b = a;
b[0] = 17;
System.out.println(a[0]);
System.out.println(b[0]);
```



```java
// Fig. 7.13: PassArray.java
// Passing arrays and individual array elements to methods.

public class PassArray
{
   // main creates array and calls modifyArray and modifyElement
   public static void main(String[] args)
   {
      int[] array = { 1, 2, 3, 4, 5 };

      System.out.printf(
         "Effects of passing reference to entire array:%n" +
         "The values of the original array are:%n");

      // output original array elements
      for (int value : array)
         System.out.printf("   %d", value);

      modifyArray(array); // pass array reference
      System.out.printf("%n%nThe values of the modified array are:%n");

      // output modified array elements
      for (int value : array)
         System.out.printf("   %d", value);

      System.out.printf(
         "%n%nEffects of passing array element value:%n" +
         "array[3] before modifyElement: %d%n", array[3]);

      modifyElement(array[3]); // attempt to modify array[3]
      System.out.printf(
         "array[3] after modifyElement: %d%n", array[3]);
   }

   // multiply each element of an array by 2
   public static void modifyArray(int[] array2)
   {
      for (int counter = 0; counter < array2.length; counter++)
         array2[counter] *= 2;
   }

   // multiply argument by 2
   public static void modifyElement(int element)
   {
      element *= 2;
      System.out.printf(
         "Value of element in modifyElement: %d%n", element);
   }
} // end class PassArray
```

```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8
```

[Question] Why the modifyArray method change the original array but modifyElement method does not change the value of the original element of the array?

Class Arrays - Helper Methods for Arrays

- **Class Arrays** helps you avoid reinventing the wheel by **providing static methods for common array manipulations.**
- These methods include **sort** for sorting an array, **binarySearch** for searching an element in a sorted array, **equals** for comparing arrays and **fill** for placing values into an array.
- These methods are overloaded for primitive-type arrays and for arrays of objects.

```
int[] arr = {5, 3, 2, 6, 1, 8};

Arrays.sort(arr);

System.out.println(Arrays.toString(arr));
```

```
[1, 2, 3, 5, 6, 8]
```

https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html

Class Arrays Example

```
1  // Fig. 7.22: ArrayManipulations.java
2  // Arrays class methods and System.arraycopy.
3  import java.util.Arrays;
4
5  public class ArrayManipulations
6  {
7     public static void main(String[] args)
8     {
9        // sort doubleArray into ascending order
10       double[] doubleArray = { 8.4, 9.3, 0.2, 7.9, 3.4 };
11       Arrays.sort(doubleArray);
12       System.out.printf("%ndoubleArray: ");
13
14       for (double value : doubleArray)
15          System.out.printf("%.1f ", value);
16
17       // fill 10-element array with 7s
18       int[] filledIntArray = new int[10];
19       Arrays.fill(filledIntArray, 7);
20       displayArray(filledIntArray, "filledIntArray");
21
22       // copy array intArray into array intArrayCopy
23       int[] intArray = { 1, 2, 3, 4, 5, 6 };
24       int[] intArrayCopy = new int[intArray.length];
25       System.arraycopy(intArray, 0, intArrayCopy, 0, intArray.length);
26       displayArray(intArray, "intArray");
27       displayArray(intArrayCopy, "intArrayCopy");
28
29       // compare intArray and intArrayCopy for equality
30       boolean b = Arrays.equals(intArray, intArrayCopy);
31       System.out.printf("%n%nintArray %s intArrayCopy%n",
32          (b ? "==" : "!="));
```

Output:

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```

[Question] Can you find out which method in Arrays Class we use in this program?

```
33
34          // compare intArray and filledIntArray for equality
35          b = Arrays.equals(intArray, filledIntArray);
36          System.out.printf("intArray %s filledIntArray%n",
37             (b ? "==" : "!="));
38
39          // search intArray for the value 5
40          int location = Arrays.binarySearch(intArray, 5);
41
42          if (location >= 0)
43             System.out.printf(
44                "Found 5 at element %d in intArray%n", location);
45          else
46             System.out.println("5 not found in intArray");
47
48          // search intArray for the value 8763
49          location = Arrays.binarySearch(intArray, 8763);
50
51          if (location >= 0)
52             System.out.printf(
53                "Found 8763 at element %d in intArray%n", location);
54          else
55             System.out.println("8763 not found in intArray");
56       }
57
58       // output values in each array
59       public static void displayArray(int[] array, String description)
60       {
61          System.out.printf("%n%s: ", description);
62
63          for (int value : array)
64             System.out.printf("%d ", value);
65       }
66    } // end class ArrayManipulations
```

Output:

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```

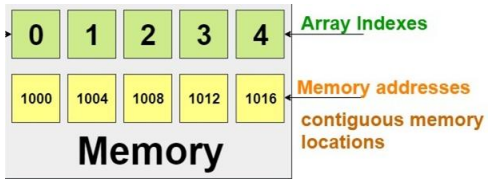
[Question] Can you find out which method in Arrays Class we use in this program?

Limitation of Array in Java

- Just like C, the size of an array can't be changed.
- The dimension of an array is determined the moment the array is created, and cannot be changed later on.
- If you want a bigger array you have to instantiate a new one, and copy elements from the old array to the new one. (Like in the below code)
- The array occupies an amount of memory that is proportional to its size, independently of the number of elements that are actually there.
- If we want to keep the elements of the collection ordered, and insert a new value in its correct position, or remove it, then, for each such operation we may need to move many elements. this is very inefficient.

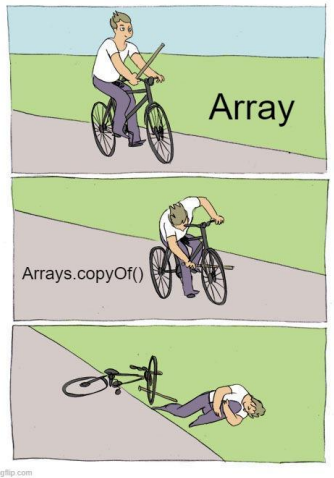
Extend array by two more elements (inconvenient way):

```
int[] oldArray = new int[] { 10, 11, 12 };

// allocating space for 5 integers
int[] newArray = Arrays.copyOf(oldArray, newLength: oldArray.length + 2);

// adding new integers at index 3, 4
newArray[3] = 13;
newArray[4] = 14;

System.out.println(Arrays.toString(newArray));
```



Output:

```
[10, 11, 12, 13, 14]
```



Recap

- Control flow statements:
  - Decision making statements:
    - **If**
    - **If...else**
    - **Switch**
  - Loop statements:
    - **While**
    - **Do while**
    - **For**
  - Jump statements:
    - **Break** statement
    - **Continue** statement
- **Array:**
  - Declare and Create Array
  - Loop through Array
  - Pass Arrays to Methods
    - Pass by Value vs Pass by Reference
  - Class Arrays for helper methods

Thank you for your listening!

“One who never asks
Either knows everything or nothing”

Malcolm S. Forbes