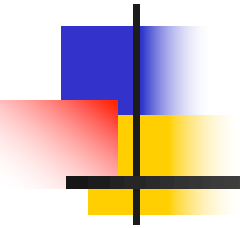
ADVANCED PROGRAMMING



CONTROL FLOW STATEMENTS

Program Control Structures

Control flow statement

When writing a program, you type statements into a file. Without control flow statements, the interpreter executes the statements in the order they appear in the file from left to right, top to bottom. You can use ***control flow statements*** in your programs to conditionally execute statements; to repeatedly execute a block of statements; and to otherwise change the normal, sequential flow of control.

Statement Type	Keyword
looping	`while`, `do-while`, `for`
decision making	`if-else`, `switch-case`
exception handling	`try-catch-finally`, `throw`
branching	`break`, `continue`, `label:`, `return`

While statement

- You use a `while` statement to continually execute a block of statements while a condition remains `true`. The following is the general syntax of the `while` statement.

```
while (logical expression) {
    statement
}
```

- First, the `while` statement evaluates expression, which must return a boolean value. If the expression returns `true`, the `while` statement executes the statement(s) in the `while` block. The `while` statement continues testing the expression and executing its block until the expression returns `false`.

Example for while statement

```
class WhileDemo {
    public static void main(String[] args){
        int count = 1;
        while (count < 11) {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
```

We can implement an infinite loop using the `while` statement as follows:

```
while (true){
   // your code goes here
}
```

do – while statement

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

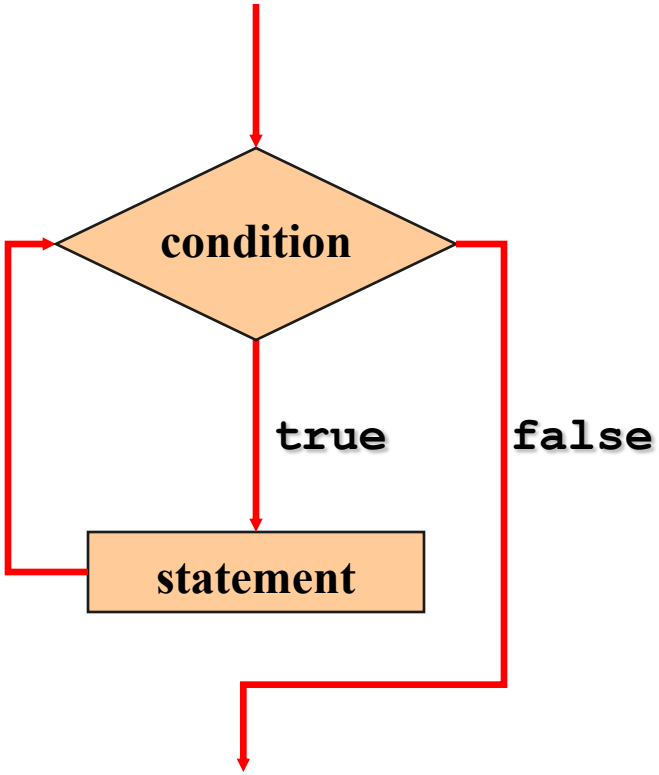
- Instead of evaluating the expression at the top of the loop, do-while evaluates the expression at the bottom. Thus, the statements within the block associated with a **`do-while`** are executed at least once.

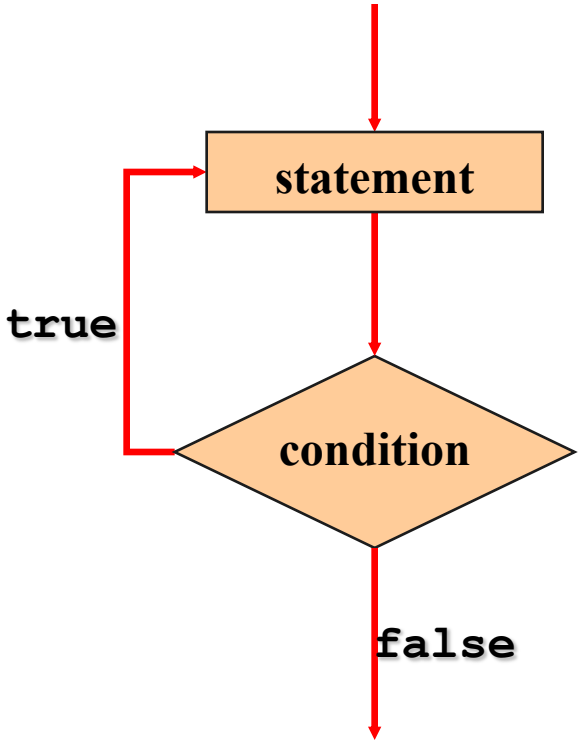
```
int count = 1;
  do {
      System.out.println("Count is: " + count);
      count++;
  } while (count <= 11);
```

Comparing the while and do – while

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

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





while loop statement example

- Greatest Common Divisor (GCD)

$$GCD(a,b) = \begin{cases} GCD(a-b,b), if\ a > b \\ GCD(a,b-a), if\ a < b \\ a, if\ a = b \end{cases}$$

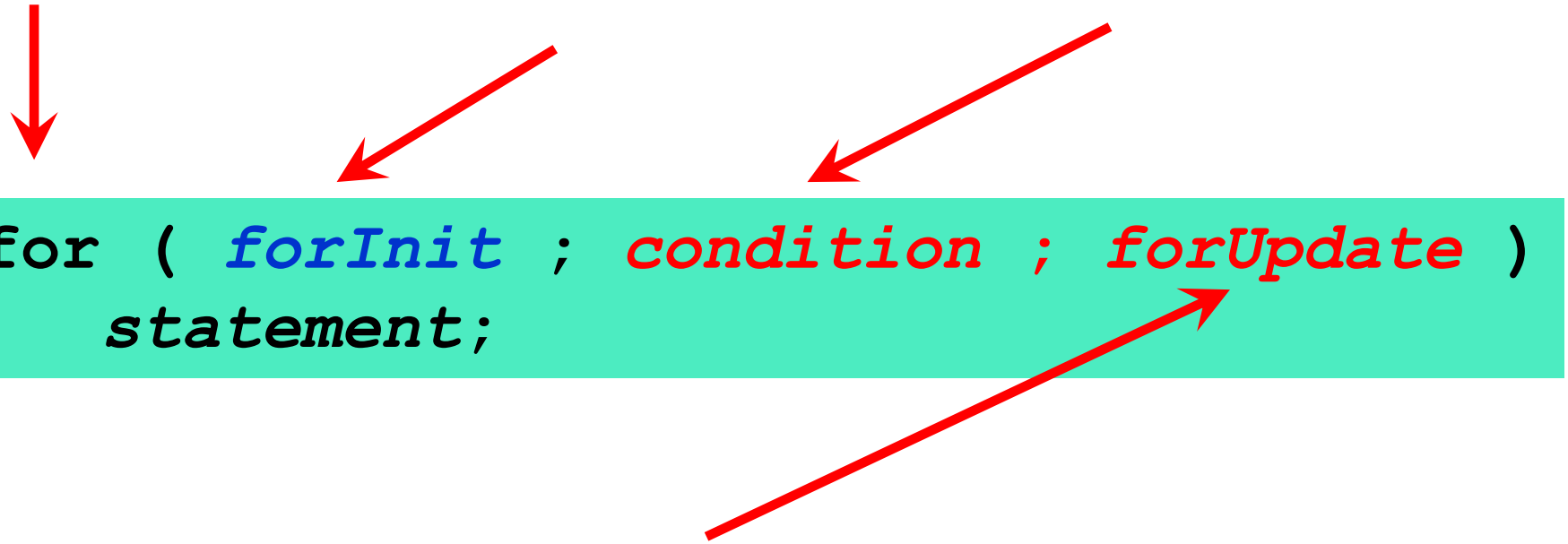
```
int a = 15;
int b = 6;
while (a != b){
   if (a > b) a = a - b;
   else if (a < b) b = b - a;
}
System.out.println("GCD is " + a);
// 3
```

The for Statement

Từ dành riêng

Phần khởi đầu *forInit* được thi hành 1 lần trước khi bắt đầu vòng lặp

Lệnh *statement* được thi hành cho đến khi điều kiện *condition* là `false`



```
for ( forInit ; condition ; forUpdate )
    statement;
```

Phần *forUpdate* được thi hành cuối mỗi vòng lặp.

The for Statement

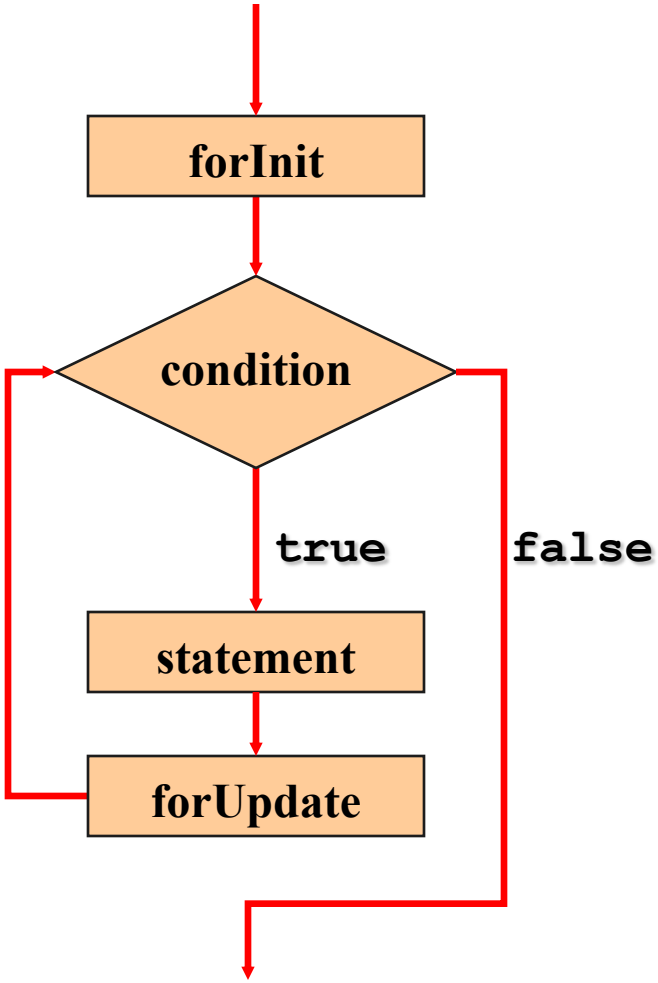
```
for (initialization; termination; increment) {
    statement(s)
}
```

- The *initialization* expression initializes the loop; it's executed once, as the loop begins.
- When the *termination* expression evaluates to false, the loop terminates.
- The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment *or* decrement a value.

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

-
```
for ( ; ; ) {    // infinite loop
    // your code goes here
}
```

The for Statement




```
forInit;
while ( condition )
{
    statement;
    forUpdate;
}
```

"for" ekvivalent

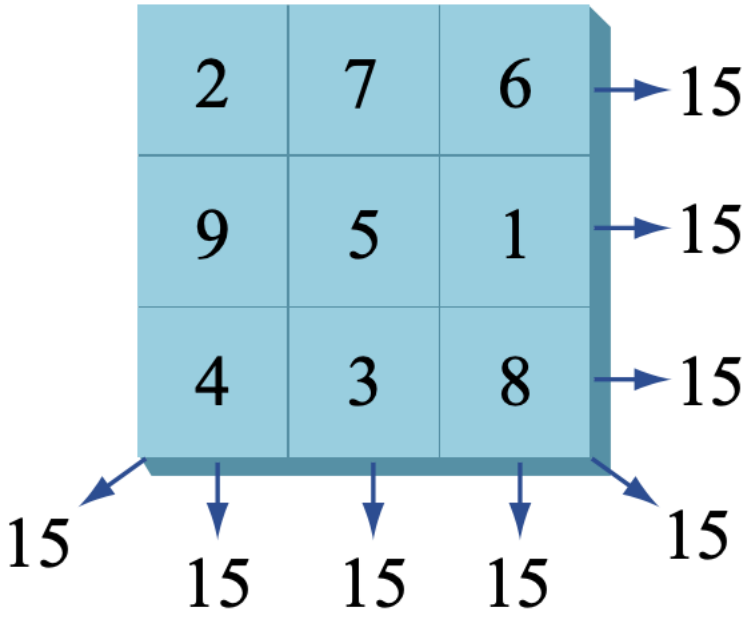
For loop statement example

- Compute n! = 1 x 2 x … x n

```
int n = 10;
long s = 1;
for (int i = 1; i <= n; i++){
    s *= i;
}
System.out.println("Factorial is " + s);
```

Assignment 1

- A magic square of order n is an arrangement of n × n numbers, usually distinct integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant (see Wikipedia: Magic Square).



Assignment 2

- Write a Java program to find the k largest elements in a given array. Elements in the array can be in any order.
- *Expected Output:*
  Original Array:
  [1, 4, 17, 7, 25, 3, 100]
  3 largest elements of the said array are:
  100 25 17

Assignment 3

- Write a Java program to create a two-dimensional array (m x m) A[][] such that A[i][j] is false if I and j are prime otherwise A[i][j] becomes true.
- *Sample Output:*

  true true true
  true true true
  true true false

Assignment 4

- Write a Java program to display the pattern like a right angle triangle with a number.
- *Test Data*

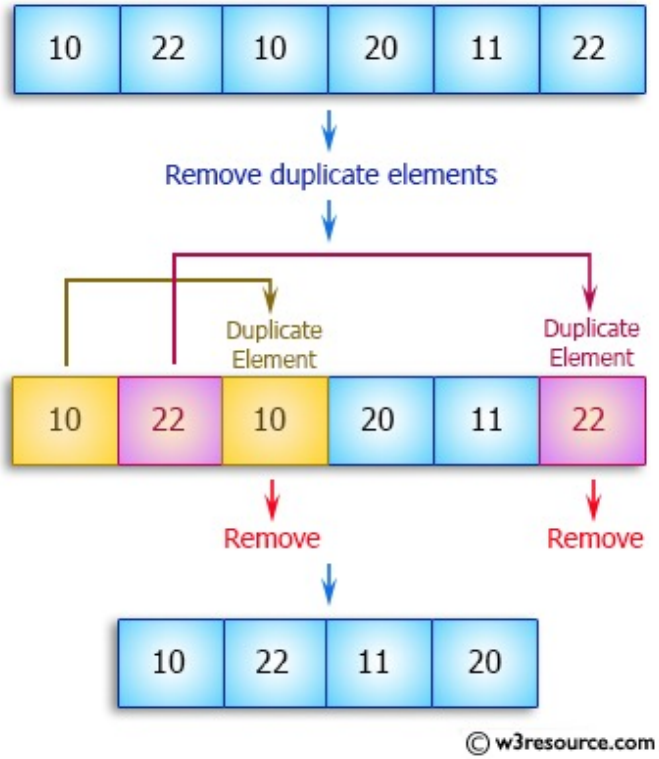
  Input number of rows : 10

  Expected Output :

```
1
12
123
1234
12345
123456
1234567
12345678
123456789
12345678910
```

Assignment 5

Write a Java program to remove duplicate elements from an array.



for vs. while

- These statements provide equivalent functionality
  - Each can be implemented in terms of the other
- Used in different situations
  - `while` tends to be used for open-ended looping
  - `for` tends to be used for looping over a fixed number of iterations

```
int sum = 0;
for (int index = 1; index <= 10; index++) {
   sum += index;
}
```

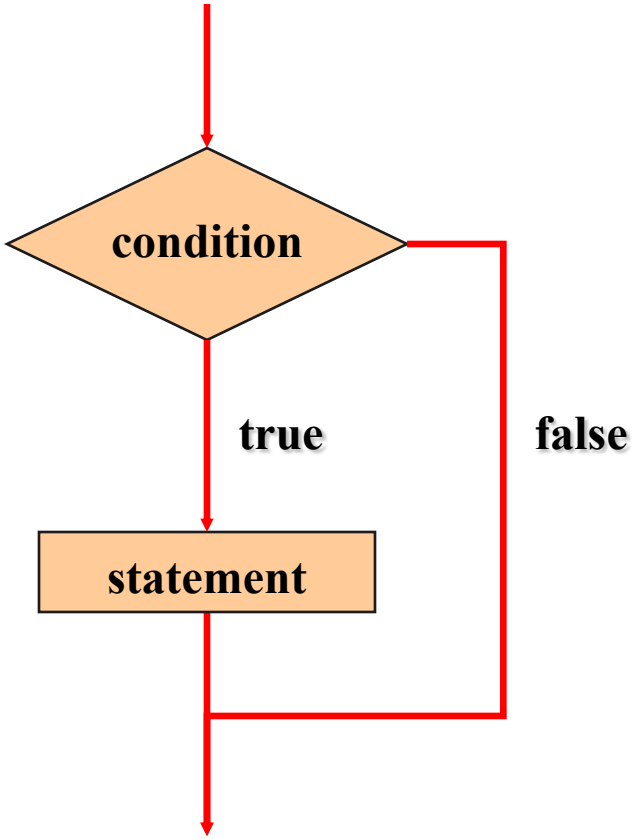
```
int sum = 0;
int index = 1;
while (index <= 10) {
   sum += index;
   index++;
}
```

Iterating with Enhanced for

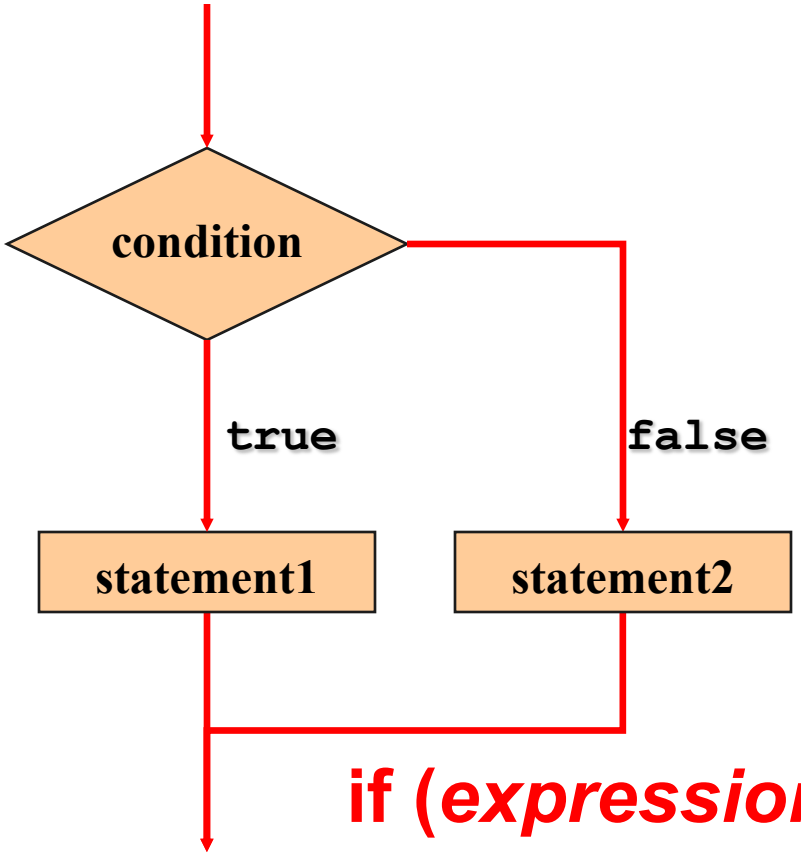
```
int[] arrayOfInts = { 32, 87, 3, 589, 12,
                      1076, 2000, 8, 622, 127 };
for (int element : arrayOfInts) {
        System.out.print(element + " ");
}
System.out.println();
void cancelAll(Collection<TimerTask> c) {
   for (TimerTask t : c)
      t.cancel();
}

for (Suit suit : suits) {
   for (Rank rank : ranks)
      sortedDeck.add(new Card(suit, rank));
}
```

The if and else Statements




```
if (expression) {
    statement(s)
}
```




```
if (expression) {
    statement1(s)
} else {
    statement2(s)
}
```

Example for if-else statement

```
public class IfElseDemo {
    public static void main(String[] args) {
        int testscore = 76;
        char grade;
        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

The Ternary Operator

- Shortcut for if-else statement:

  `(<boolean-expr> ? <true-choice> : <false-choice>)`

- Can result in shorter code
  - Make sure code is still readable

```
int max;
if (x > y) {
   max = x;
} else {
   max = y;
}
```

vs.

```
max = (x > y) ? x : y;
```

Switch statement

```
switch ( value )
{
    case value_1 :
        statement_list_1
    case value_2 :
        statement_list_2
    case value_3 :
        statement_list_3
    default:   ...
}
```

Use the `switch` statement to conditionally perform statements based on an `integer expression` or `enumerated type` (`byte`, `short`, `char`, and `int` primitive data types)

Example for switch statement

```
int month = 8;
 switch (month) {
     case 1:  System.out.println("January"); break;
     case 2:  System.out.println("February"); break;
     case 3:  System.out.println("March"); break;
     case 4:  System.out.println("April"); break;
     case 5:  System.out.println("May"); break;
     case 6:  System.out.println("June"); break;
     case 7:  System.out.println("July"); break;
     case 8:  System.out.println("August"); break;
     case 9:  System.out.println("September"); break;
     case 10: System.out.println("October"); break;
     case 11: System.out.println("November"); break;
     case 12: System.out.println("December"); break;
     default: System.out.println("Not a month!");break;
 }
```

Ex.: Calculte a number of days in a month

```
int month = 2;
int year = 2000;
int numDays = 0;
switch (month) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        numDays = 31;
        break;
```

Ex.: Calculte a number of days in a month

```
 case 4:
   case 6:
   case 9:
   case 11:
        numDays = 30;
        break;
   case 2:
        if ( ((year % 4 == 0) && !(year % 100 == 0))
             || (year % 400 == 0) )
        numDays = 29;  else  numDays = 28;
        break;
    default:
        numDays = 0;
        break;
}
System.out.println("Number of Days = " +
                                        numDays);
```

Enumerated Types in switch Statements

```
public class SwitchEnumDemo {
    public enum Month { JANUARY, FEBRUARY, MARCH, APRIL,
                        MAY, JUNE, JULY, AUGUST, SEPTEMBER,
                        OCTOBER, NOVEMBER, DECEMBER }

    public static void main(String[] args) {
        Month month = Month.FEBRUARY;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case JANUARY:
            case MARCH:
            case MAY:
            case JULY:
            case AUGUST:
            case OCTOBER:
            case DECEMBER:
                numDays = 31;
                break;
```

Enumerated Types in switch Statements

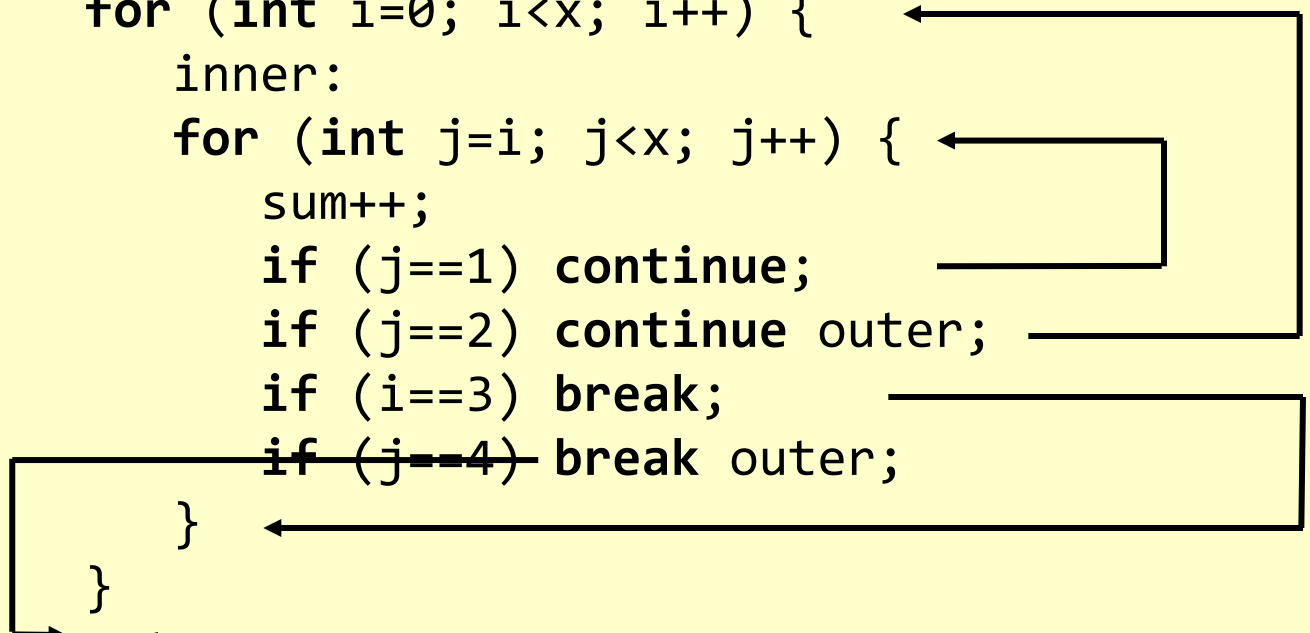
```
            case APRIL:
            case JUNE:
            case SEPTEMBER:
            case NOVEMBER:
                numDays = 30;
                break;
            case FEBRUARY:
                if ( ((year % 4 == 0) && !(year % 100 == 0))
                     || (year % 400 == 0) )
                    numDays = 29;
                else
                    numDays = 28;
                break;
            default:
                numDays=0;
                break;
        }
        System.out.println("Number of Days = " + numDays);
    }
}
```

Branching Statements

- The Java programming language supports the following branching statements:
  - The **break** statement
  - The **continue** statement
  - The **return** statement
- The **break** and **continue** statements, which are covered next, can be used with or without a **label**. A **label** is an identifier placed before a statement; it is followed by a colon (**:**).
  statementName : someStatement;

Sample Branching Statements

```
public int myMethod(int x) {
   int sum = 0;
   outer:
   for (int i=0; i<x; i++) {
      inner:
      for (int j=i; j<x; j++) {
         sum++;
         if (j==1) continue;
         if (j==2) continue outer;
         if (i==3) break;
         if (j==4) break outer;
      }
   }
   return sum;
}
```



The break Statements

The **break** statement has two forms:

- **unlabeled form**: The unlabeled form of the **break** statement was used with **switch** earlier. As noted there, an unlabeled **break** terminates the enclosing **switch** statement, and flow of control transfers to the statement immediately following the **switch**. That is mean unlabeled **break** terminates the enclosing loop. The unlabeled form of the **break** statement is used to terminate the innermost **switch**, **for**, **while**, or **do-while** statement;
- **labeled form**: the labeled form terminates an outer statement, which is identified by the label specified in the break statement.

Unlabled break statement

```
public class BreakContinue extends TestCase{
   public void test(){
      int out,in=0;
      for (out = 0 ;out<10; out++){
         for (in =0; in<20; in++){
                  if (in>10) break;
         }
         System.out.println("inside the outer loop: out = " +
                                                   out + ", in = " + in);
      }
     System.out.println("end of the outer loop: out = " +
                                                   out + ", in = " + in)
   }
}
```

What do you see?

Labeled break statement

```
public class BreakContinue extends TestCase{
   public void test(){
      int out,in=0;
       outer:
      for (out = 0 ;out<10; out++){
         for (in =0; in<20; in++){
                  if (in>10) break outer;
         }
         System.out.println("inside the outer loop: out = " +
                                                  out + ", in = " + in);
      }
     System.out.println("end of the outer loop: out = " +
                                                  out + ", in = " + in)
   }
}
```

What do you see?

Assignment 6

- Write a Java program to find the first duplicate value and return that index in an array of integer values.
- Input: int[] arrays = {1,4,5,7,8,5,9}

  int numberFind = 5;
- Output: 2

The continue Statement

- The **continue** statement is used to skip the current iteration of a `for`, `while` , or `do-while` loop. The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop, basically skipping the remainder of this iteration of the loop.
- The labeled form of the `continue` statement skips the current iteration of an outer loop marked with the given label.

Unlabled continue statement

```
StringBuffer searchMe = new StringBuffer(
        "peter piper picked a peck of pickled peppers");
        int max = searchMe.length();
        int numPs = 0;
        System.out.println(searchMe);
        for (int i = 0; i < max; i++) {
                //interested only in p's
                if (searchMe.charAt(i) != 'p') continue;

                //process p's
                numPs++;
                searchMe.setCharAt(i, 'P');
        }
        System.out.println("Found " + numPs
                        + " p's in the string.");
        System.out.println(searchMe);
}
```

The result of mentioned example

peter piper picked a peck of pickled peppers

Found 9 p's in the string.

Peter PiPer Picked a Peck of Pickled PePPers

The Labeled continue statement

```
//finds a substring(substring) in given string(serchMe)
   String searchMe = "Look for a substring in me";
    String substring = "sub";
    boolean foundIt = false;
    int max = searchMe.length() - substring.length();
test:
    for (int i = 0; i <= max; i++) {
       int n=substring.length(), j=i, k= 0;
       while (n-- != 0) {
            if(searchMe.charAt(j++)!= substring.charAt(k++)){
                continue test;
             }
        }
        foundIt = true;
        break test;
    }
    System.out.println(foundIt ? "Found it" :"Didn't find it");
}
```

The return Statement

- The **`return`** statement has two forms: (1) one that returns a value and (2) one that doesn't. To return a value, simply put the value (or an expression that calculates the value) after the **`return`** keyword.
- **`return ++count;`**
- The data type of the value returned by **`return`** must match the type of the method's declared return value. When a method is declared **`void`**, use the form of **`return`** that doesn't return a value
- **`return;`**

The first form of "return" statement

```
public boolean seachFirst(){
        int[] array = {10,5,9,3,8,5,8,5};
        int matchValue = 8;
        for (int i=0; i<array.length; i++){
                if (matchValue == array[i]) return true;
        }
        return false;
}
```

2. form of "return" Statement (version 1)

```
public void displayDayOfWeek(int day){
        if (day == 1){
                System.out.println("Sunday");
                return;
        }
        if (day == 2){
                System.out.println("Monday");
                return;
        }
        if (day == 3){
                System.out.println("Tueday");
                return;
        }
          if .....................
}
```

2. form of "return" Statement (version 2)

```
public void displayDayOfWeek(int day){
        if (day == 1){
                System.out.println("Sunday");
        } else if (day == 2){
                System.out.println("Monday");
        } else if (day == 3){
                System.out.println("Tueday");
        } else .....
   }
```

Which version do you like?

Summary: Branching Statements

- *statementName: someJavaStatement*;
- break;
- break *label*;
- continue;
- continue *label*;
- return;
- return *value*;

Assignment 7

- Write a Java program to create a two-dimensional array (m x m) A[][] such that A[i][j] is false if I and j are prime otherwise A[i][j] becomes true.
- *Input: int m=3; int[][] array= new int[m][m];*
- *Sample Output:*

  true true true
  true true true
  true true false

Assignment 8

- Write a Java program to find the k^{th} smallest and largest element in a given array. Elements in the array can be in any order.
- *Expected Output:*

  Original Array:
  [1, 4, 17, 7, 25, 3, 100]
  K'th smallest element of the said array:
  3
  K'th largest element of the said array:
  25

Bài tập MyString

```
Class MyString{
    private char[] content;
    public MyString(char[] con);
    public MyString(String s);
    public int indexOf(char c);
    public int lastIndexOf(char c);
    public int indexOf(String sub);
    public int lastIndexOf(String sub);
    public void insert(int index, String sub);
    public void replace(String old_s, String new_s);
    public void replaceAll(String old_s, String new_s);
}
```