Loops

Programming in Java

Contents

9. The **while** loop

10. The **do-while** loop

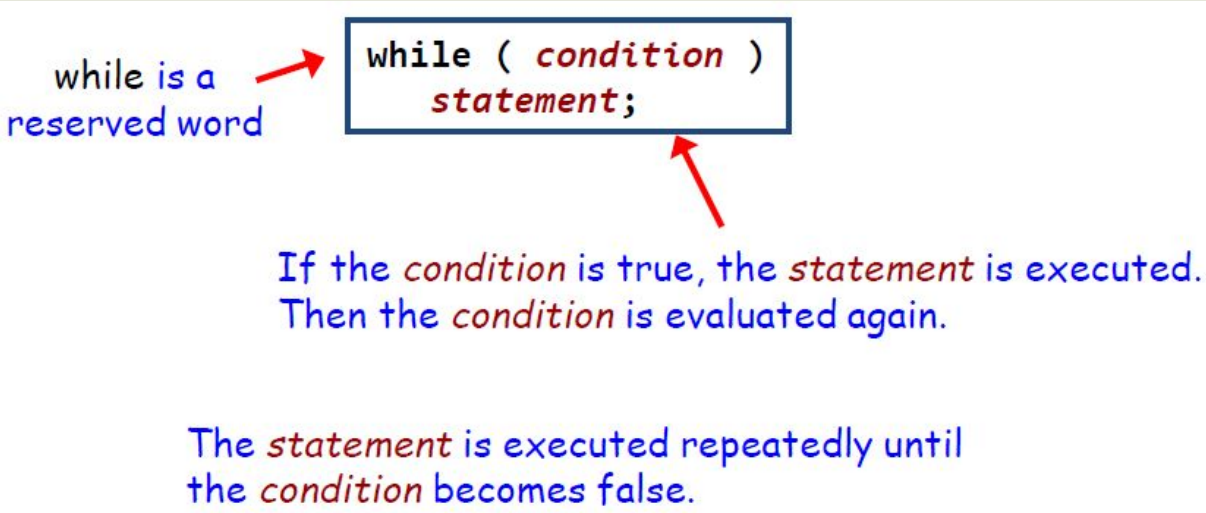
11. The **for** loop

12. Nested loops

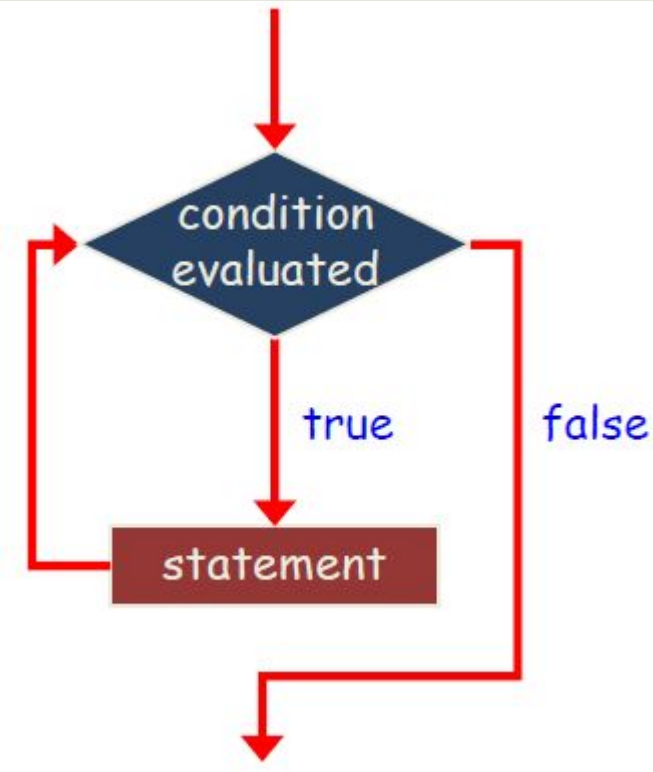
13. **break, continue & exit**

9- The while loop

- Syntax:



9- The while loop: Logic of a while loop

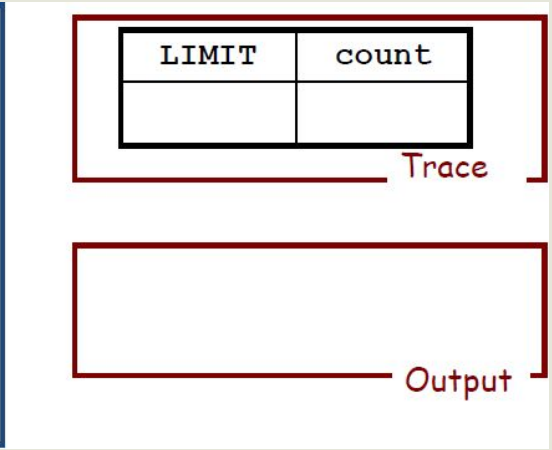


9- The while loop

- Note that if the condition of a **while** statement is false initially, the statement is never executed
- So, the body of a **while** loop will execute zero or more times

```
final int LIMIT = 5;
int count = 1;

while (count <= LIMIT)
{
   System.out.println(count);
   count = count + 1;
}
System.out.println("Done");
```



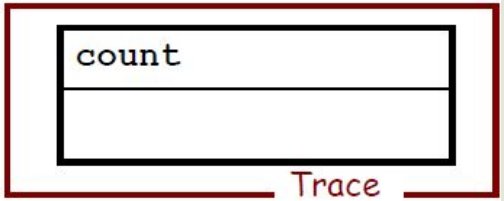
9- The while loop: Example

```
int remainingStars = 5;

while (remainingStars > 0)
{
    System.out.println("*");
    remainingStars--;
}
```

9- The while loop: Example

```
public class Forever
{
  public static void main (String[] args)
  {
       int count = 1;
       while (count <= 10)
       {
          System.out.println (count);
          count = count - 1;
       }
       System.out.println("Done");
    }
  }
```



9- The while loop: Exercise

```
  int index = 1;
while (index != 10)
{
   System.out.print("Hello");
   index = index + 2;
}
```

A. There will be no output, since index is not equal to 10

B. HelloHelloHelloHello

C. HelloHelloHelloHelloHello

D. HelloHelloHelloHelloHelloHello

E. None of the above are correct choices

9- The while loop: Exercise

```
boolean finished = false;
int firstInt = 3;
int secondInt = 20;
while (firstInt <= secondInt &&
  !finished)
  if (secondInt / firstInt <= 2)
     finished = true;
  else
     firstInt++;
  System.out.println(firstInt);
```

A. 3
B. 5
C. 7
D. 8
E. 9

9- The while loop: Exercise

```
Enter a series of marks (negative number to quit):
80.5 70  67  53.8 -1
The average is: 67.825
```

Data needed:

Algorithm:

9- The while loop: Exercise

- same thing… but now, determine the highest and lowest marks too.

Data needed:

Algorithm:

10- The do-while loop

- syntax:

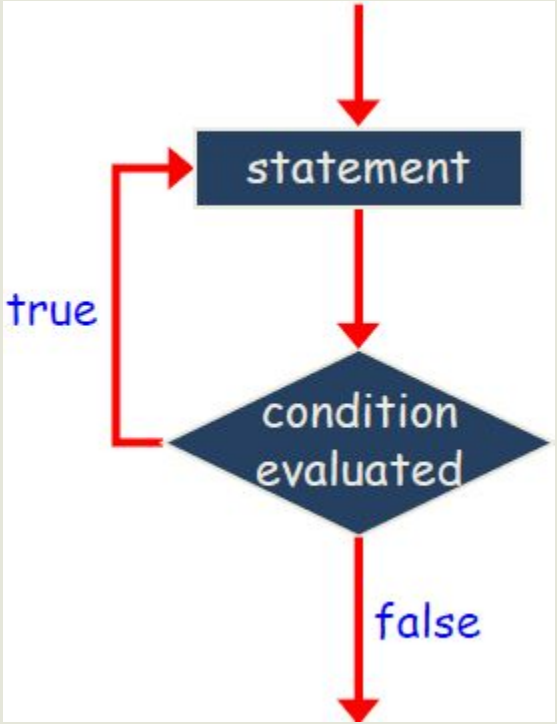
do and while are reserved words

```
do
{
    statement;
}
while ( condition );
```

The *statement* is executed once initially, and then the *condition* is evaluated.

The *statement* is executed repeatedly until the *condition* becomes false.

10- The do-while loop: Logic of a do-while loop



10- The do-while loop

- A do loop is similar to a while loop, except that the condition is evaluated after the body of the loop is executed.
- Therefore the body of a do loop will execute at least once.

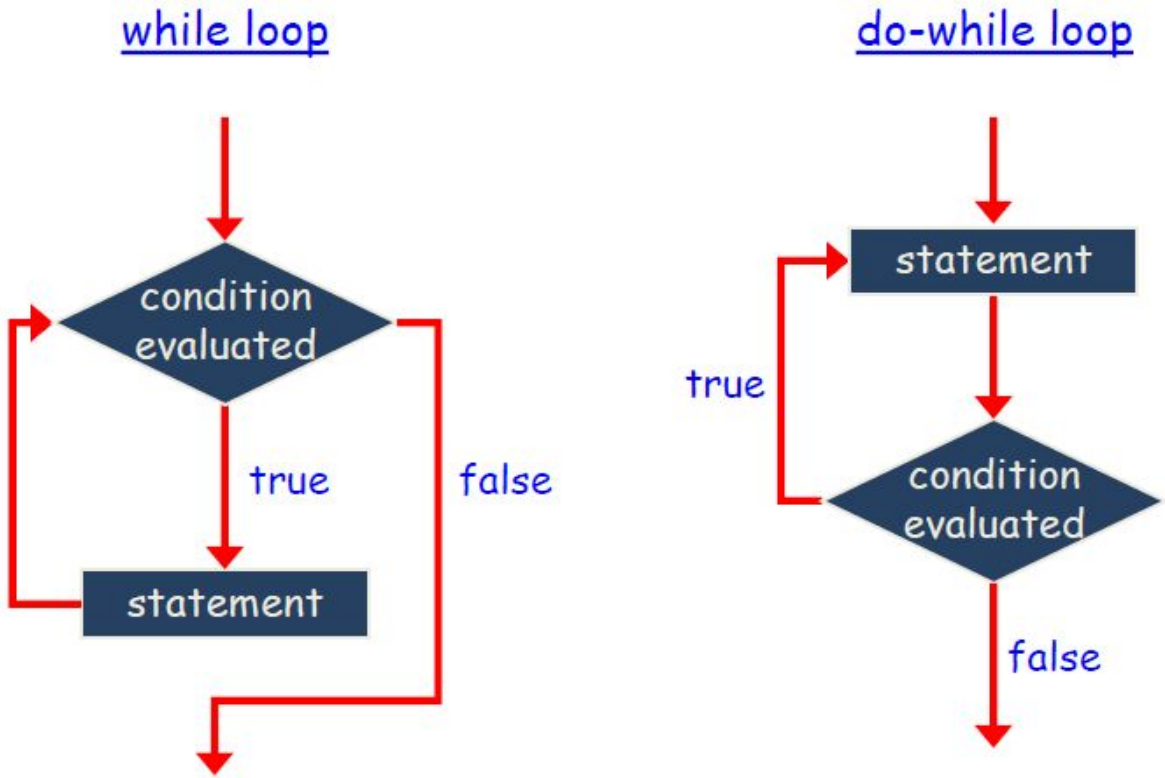
```
int n = 0;
while (n > 0)
{
    System.out.println("*");
    n--;
}
System.out.println(n);
```

Output

```
int n = 0;
do
{
    System.out.println("*");
    n--;
}
while (n > 0);
System.out.println(n);
```

Output

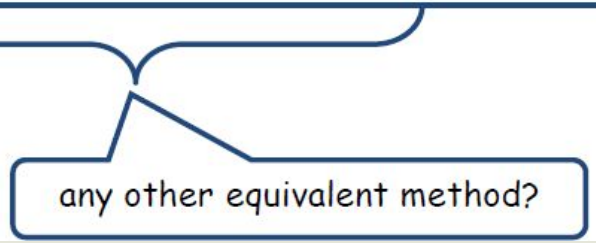
10- The do-while loop: Comparing while and do while



10- The do-while loop: Typical applications

- User-controlled loop

```
String answer;
do
{
    // do the computation
    // …
    System.out.println("Do you wish to continue(yes/no)?");
    answer = myKeyboard.next();
}
while ((answer.toUpperCase()).compareTo("YES") == 0);
```



10- The do-while loop: Typical applications

- To verify user input

```
int age;
boolean valid;
do
{
   System.out.println("How old are you?");
   age = myKeyboard.nextInt();
   valid = (age > 0) && (age < 125);
   if (!valid)
      System.out.println("error! try again!");
}
while (!valid);
```

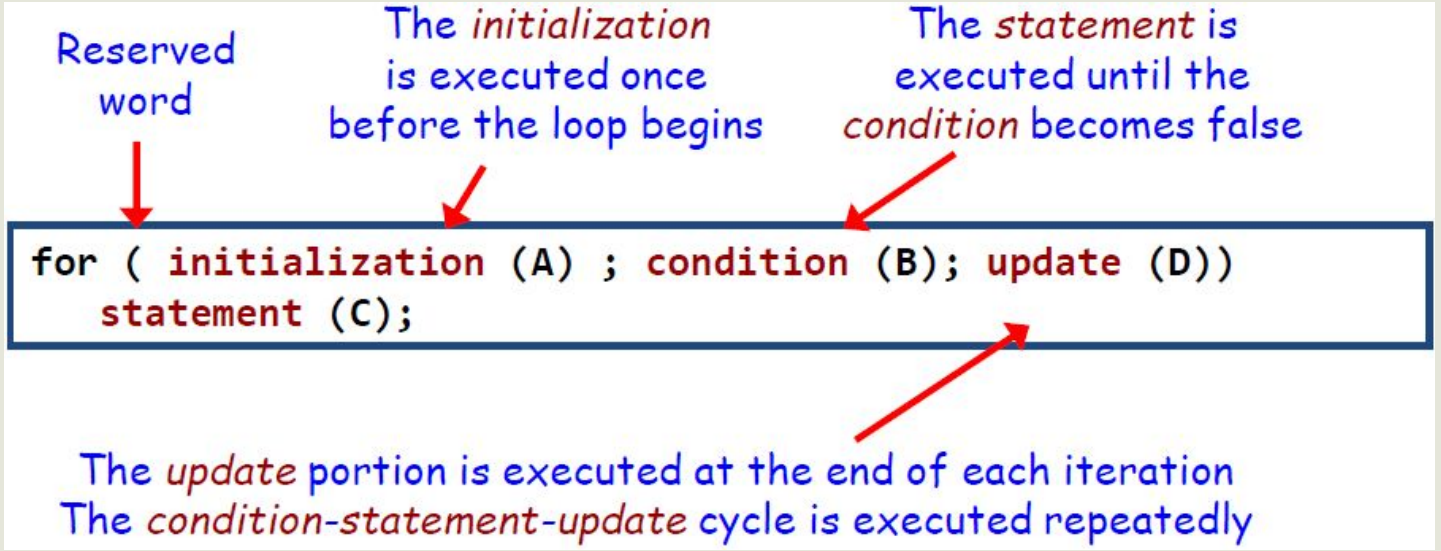
10- The do-while loop: Exercise

```
int beta = 5;
do
{
  switch (beta)
  {
      case 1 :
System.out.print('R');
                  break;
      case 2 :
      case 4 :
System.out.print('O');
                  break;
      case 5 :
System.out.print('L');
    }
    beta--;
 } while (beta > 1);
System.out.print('X');
```

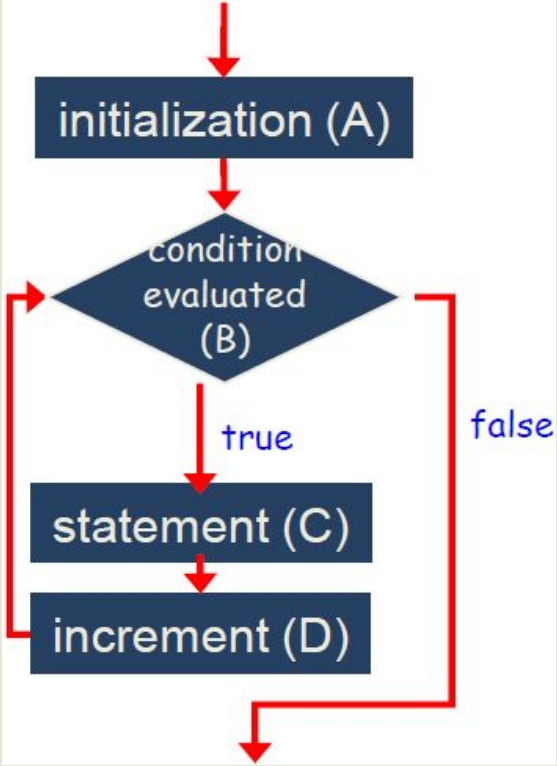
A. X
B. ROOLX
C. LOORX
D. LOOX
E. ROOX

11- The for loop

- Syntax:

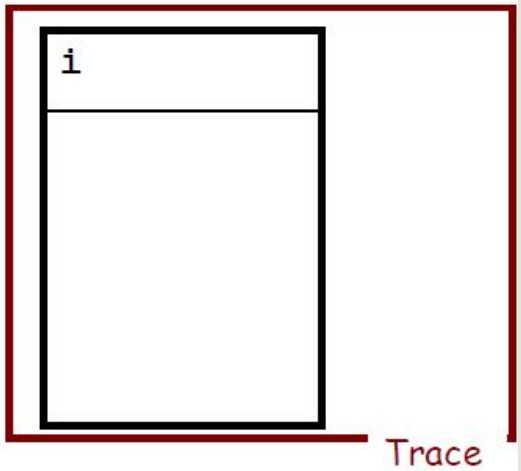


11- The for loop: Logic of a for loop



11- The for loop: Example

```
int i;
for (i=1; i <= 10; i++)
    System.out.print(i);
System.out.print(i);
```



11- The for loop: for versus while

- A **for** loop is equivalent to the following **while**:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

```
for ( initialization (A) ; condition (B); update (D))
    statement (C);
```

11- The for loop: Example

```
for (int i=0; i < 0; i--)
    System.out.print("hello");
```

Output

```
for (int i=0; i <= 0; i--)
  System.out.print("hello");
```

Output

11- The for loop: Example

```
Enter a positive value:  10
Enter an upper limit:  95
Multiples of 10 between 10 and 95:
10 20 30 40 50
60 70 80 90
```

User input in red

Data needed:

Algorithm:

11- The for loop: Example

```
final int PER_LINE = 5;
int value, limit, mult, count = 0;
Scanner myKeyboard = new Scanner(System.in);
System.out.print("Enter a positive value: ");
value = myKeyboard.nextInt();
System.out.print("Enter an upper limit: ");
limit = myKeyboard.nextInt();
System.out.println("Multiples of "+value+" between "+ value + " & " +
limit);
for (_________________; _________________; _____________________) {
   System.out.print(mult + "\t");
   // Print a specific number of values per line of output
   count++;
   if (count % PER_LINE == 0)
      ________________________;
}
```

11- The for loop: More on for loops

- Each expression in the header of a **for** loop is optional
  - If the initialization is left out for (; j > 0; j++) ….
    - no initialization is performed
  - If the condition is left out for (int j = 0 ; ; j++) ….
    - it is always considered to be true
  - If the increment is left out for (int j = 0 ; j > 0;) ….
    - no increment operation is performed
- Both semi colons are always required

11- The for loop: Exercise

Which of the loops below produces the same number of loop iterations as the following loop? (count is of type int.)

```
for (count = 1; count <= 10; count++)
        System.out.println (count);
```

A.
```
for (count = 10; count >= 1; count--)
        System.out.println (count);
```

B.
```
for (count = 0; count < 10; count++)
        DoSomething();
```

C.
```
for (count = 10; count >= 0; count--)
        DoSomething();
```

D. a and b above

E. a, b, and c above

11- The for loop: Even more on for loops

- Can have multiple expressions in:
  - the initialization part
  - the increment part

- Separate each expression by a comma

```
for (int i=0, j = 1; i <= 10; i++, j=2*j)
    System.out.println(i + " " + j);
```

11- The for loop: Example

- Assume:

```
int sum, i;
```

```
sum = 0;
for (i=0; i < 10; i++)
    sum+=i;
System.out.print(sum);
```

Output

```
for (i=0, sum=0; i<10; sum+=i,i++);
System.out.print(sum);
```

Output

11- The for loop: Example

- Assume:

```
int sum, i;

for (i=0,sum=0; i<10; i++)
    sum+=i;
System.out.print(sum);
```

Output

```
for (i=0,sum=0; i<10; i++,sum+=i);
System.out.print(sum);
```

Output

Which loop to use?

- Any loop can be re written with another loop
- In general, use a:
  - **while** or a **do:**
    - when you don't know in advance how many times you want to execute the loop body
    - if it will be at least once, use a do loop
  - **for:**
    - when you know how many times you want to execute the loop body

12- Nested Loops

- A **for** inside a **for**, a **while** inside a **for**, a **do** inside a **while**, …
- i.e. the body of a loop can contain another loop
- Consists of:
  - an outer loop
  - an inner loop
- For one iteration of the outer loop, the inner loop goes through its full set of iterations

12- Nested Loops: Example

```
for (i = 2; i <= 4; i++)
{
   for (j = 6; j <= 7; j++)
      System.out.println(i + " " + j);

   System.out.println("j is now " + j);
}
System.out.println("i is now " + i);
```

i	j

Trace

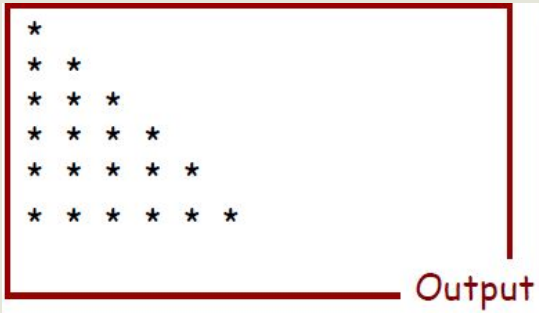
Output

12- Nested Loops: Example

algorithm??

Java code ??

12- Nested Loops: Example



algorithm??

Java code ??

12- Nested Loops: Example

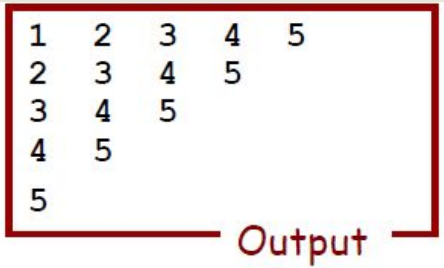
```
1
2  2
3  3  3
4  4  4  4
5  5  5  5  5
```

Output

algorithm??

Java code ??

12- Nested Loops: Example



algorithm??

Java code ??

12- Nested Loops: Example

```
String str, another = "y";
int left, right;
do {
   System.out.println ("Enter a string:");
   str = Keyboard.next();
   left = 0;
   right = str.length() - 1;
   while (str.charAt(left) == str.charAt(right) && left < right) {
      left++;
      right--;
   }
   System.out.println ((left < right)? "NO": "YES");
   System.out.print ("Test another string (y/n)? ");
   another = keyboard.next();
}
while ((another.equalsIgnoreCase("y"));
```

Output

12- Nested Loops

```
int n = 2;
for (int loopCount = 1; loopCount <= 3; loopCount++)
        while (n <= 4)
            n = 2 * n;
System.out.println(n);
```

A. 4

B. 8

C. 16

D. 32

E. 64

13- break and continue

- Bypasses the normal flow of control of loops
- Very practical sometimes… but use in moderation…
- **break**
  - Will exit the inner most loop without evaluating the condition
- **continue**
  - Will interrupt the current iteration (of the inner most loop)
  - And will force a new evaluation of the condition for a possible new iteration
  - Note: in a for loop, the incrementation is done before the condition is tested…

13- break and continue: Example

```
int n;
while (true) {
  System.out.print("Enter a positive number");
  n = keyboard.nextInt();
  if (n < 0)
    break;
  System.out.println("squareroot of " + n
                + " = " + Math.sqrt(n));
}
```

13- break and continue: Example

```
int n;
while (true)
 {
        System.out.print("Enter a positive integer, 0 to end");
        n = keyboard.nextInt();
        if (n == 0)
                break;
        if (n < 0)
                continue;
        System.out.println("squareroot of " + n + " = " +
 Math.sqrt(n));
 }
```

13- break and continue: Prime numbers from 10 to 50 example

```
11 13 17 19 23 ...
```

Output

```
10 --> verify 2 3 4 5 6 7 8 9
11 --> verify 2 3 4 5 6 7 8 9 10
12 --> verify 2 3 4 ... 11
...
15 --> verify 2 3 4 5 ... 14
...
33 --> verify 2 3 4 5 ... 32
...
50 --> verify 2 3 4 ... 49
```

Method

```
boolean divisible;
final int UP = 50;
final int LOW = 10;

for (int number = LOW; number <= UP; number++)
{
   for (int candidate = 2; candidate < number;
     candidate++)
   {
      divisible = (number % candidate) == 0;
      if (divisible)
         break; ??  continue; ??
   }
   if (!divisible) // ok ?
      System.out.print(number + " ");
}
```

Code

The exit statement

- A **break** statement will end a loop or switch statement, but will not end the program

- The **exit** statement will immediately end the program as soon as it is invoked:

  System.exit (0);

- The **exit** statement takes one integer argument
  - By tradition, a zero argument is used to indicate a normal ending of the program