303.3.1.
Iteration Statements (Loops)

Loops in Java

Learning Objectives:

This presentation introduces the basic structure and concept of Java for loop, while loop, and do-loop with example codes and labs.

By the end of this lesson, learners will be able to:

- Describe and understand loops.
- Demonstrate and utilize the Java loops using for-loop, while loop, and do-while loop.
- Demonstrate and describe break and continue Loop Control Statements.

Outline

- Introduction to Loops.
  - **while** Loops.
  - **do-while** Loops.
  - **for** Loops.
  - Enhanced **for** Loop
- Use Which Loops?
- Loop Control Statements.
  - **break Keyword.**
  - **continue Keyword.**

Lesson 1

Introduction to Loops

In this lesson, we will discuss and review the loop concept and the different Java Loops.

Topic 1: Introduction to Loops

- In general, statements are executed sequentially – the first statement in a method is executed first, followed by the second, and so on.
- There are times when you need to execute a block of code multiple times; often with variations in each execution.
- Programming languages provide ***flow-control*** *structures* that allow more complicated execution paths.
- Loop statements allow for executing a single statement or group of statements multiple times.
- A loop is typically controlled by an index or counter variable.

Why Loops?

Repeated Actions:

- Suppose we need to print a string (e.g., "Welcome to Java World!") 100 times. It would be tedious to have to write the following statement a hundred times:

```
System.out.println("Welcome to Java World!");
```

- How do you solve this problem?

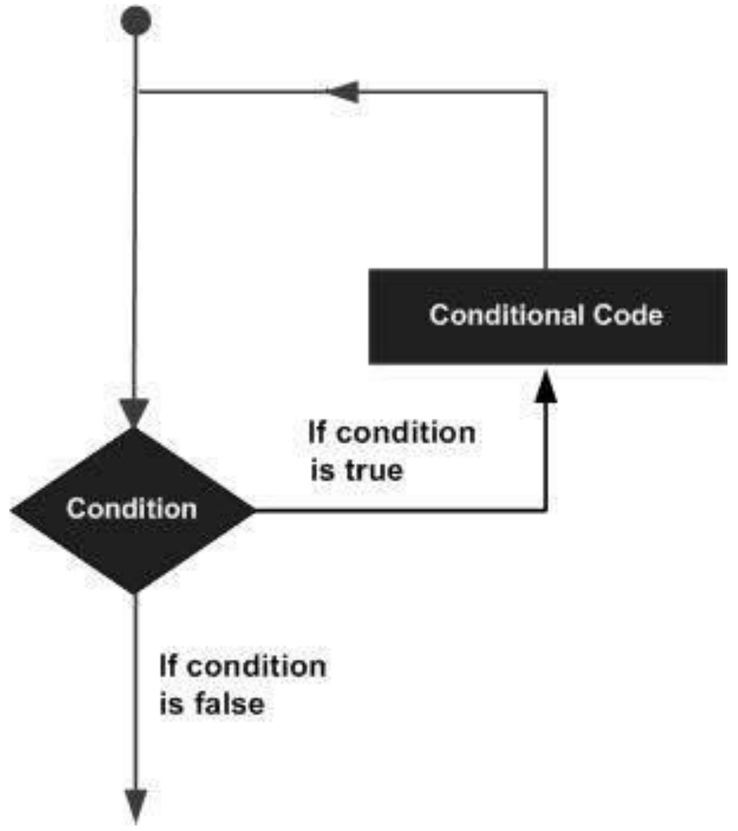
Opening Problem

100
times

```
System.out.println("Welcome to Java World!")
System.out.println("Welcome to Java World!")
System.out.println("Welcome to Java World!")
…
…
…
System.out.println("Welcome to Java World!")
System.out.println("Welcome to Java World!")
System.out.println("Welcome to Java World!")
```

Loop Statement Flowchart

The diagram illustrates a loop statement:

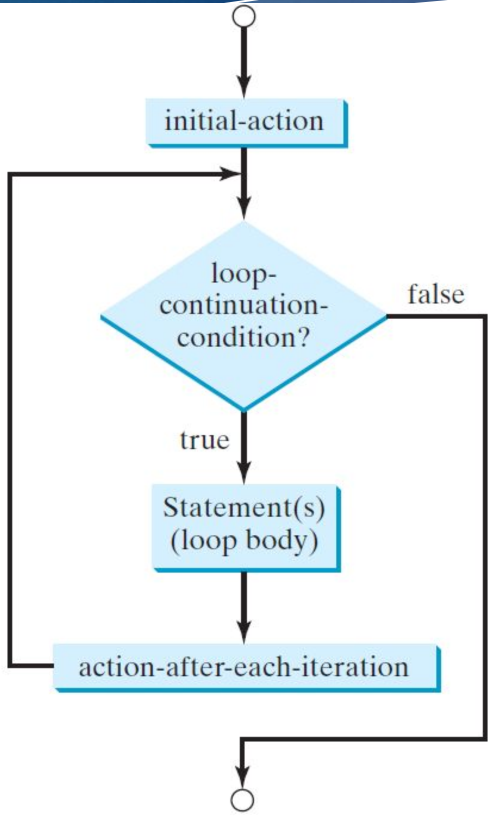


Types of Loops in Java

- Java supports three loop statements:
  - **for loop**
    - Executes a sequence of statements multiple times over a range of values, and abbreviates the code that manages the loop variable.
  - **while loop**
    - Repeats a statement or group of statements while a given condition is true.
    - Evaluates the condition before executing the loop body.
  - **do-while loop**
    - Similar to while loop, except that the condition is evaluated <u>after</u> the statements are executed.
    - A do-while loop guarantees that the loop executes at least once.

Topic 1a: Introduction to *for* Loops

- A **for** loop in Java iterates over the items in any sequence:
  - The statement(s) within the **for** loop will repeat until the entire sequence has been visited.
  - **For** loops are often used to iterate through all elements in a collection.
  - A **for** loop has three parts:
    - Initialization.
    - Continuation condition.
    - Action-after-each-iteration.
- General Syntax:



```
for (initialize action; continuation-condition; after-iteration-action) {
    // Loop body
  statement(s);
}
```

for Loop Flowchart Example

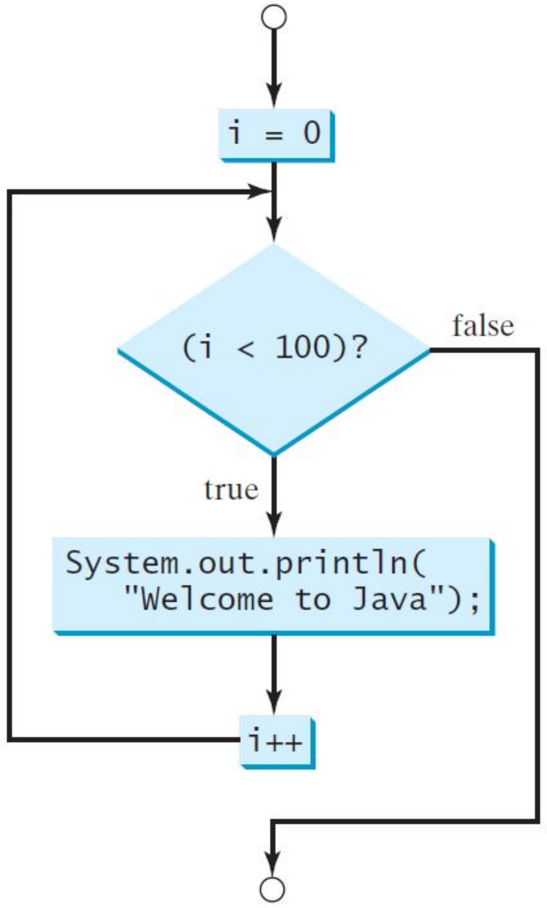
Three parts:

```
for (int i = 0; i < 100; i++) {
   System.out.println("Welcome to Java!");
}
```

Initialization

Continuation Condition

action-after-each-iteration



Trace *for* Loop

Initialize #1:

```
for (int i=0; i<2; i++){
   System.out.println("Welcome to Java!");
}
System.out.println("Goodbye!");
```

Declare and initialize *i; i is now 0.*

It is also valid to declare *i* before the loop, giving *i* greater scope:

```
int i;

for ( i=0; i<2; i++){
   System.out.println("Welcome to Java!");
}
System.out.println("Goodbye!");
```

Declare *i variable*

Initialize 0 *value in variable i*

Trace *for* Loop

Test for Continuation #1

```
for (int i=0; i<2; i++){
    System.out.println("Welcome to Java!");
}
System.out.println(“Goodbye!");
```

i is 0, so i <2 is true.

Execute the Loop Body #1

```
for (int i=0; i<2; i++){
    System.out.println("Welcome to Java!");
}
System.out.println(“Goodbye!");
```

Print “Welcome to Java!”

Action-after-each-iteration #1

```
for (int i=0; i<2; i++) {
    System.out.println("Welcome to Java!");
}
System.out.println(“Goodbye!");
```

Execute the post-iteration action:
Increment i.
i is now *1*.

TEKsystems
Own change

Trace ***for*** Loop: Test for Continuation #2

Test for Continuation #2

```
for (int i=0; i<2; i++){
   System.out.println("Welcome to Java!");
}
System.out.println("Goodbye!");
```

i is 1; so i <2 is still true.

Trace *for* Loop: Execute the Loop Body #2

```
for (int i=0; i<2; i++){
    System.out.println("Welcome to Java!");
}
System.out.println("Goodbye!");
```

Print “Welcome to Java!”

Trace *for* Loop: After-Iteration Action #2

```
for (int i=0; i<2; i++){
   System.out.println("Welcome to Java!");
}
System.out.println(“Goodbye!");
```

Execute the post-iteration action:
Increment i.
i is now 2.

Trace ***for*** Loop: Test for Continuation #2

```
for (int i=0; i<2; i++){
   System.out.println("Welcome to Java!");
}
System.out.println(“Goodbye!");
```

i is 2, so i <2 is now false.

Trace *for* Loop Test for Continuation #2 (continued)

```
for (int i=0; i<2; i++){
   System.out.println("Welcome to Java!");
}
System.out.println("Goodbye!");
```

Exit the **for** loop.
Execute first statement after the loop.

A Misplaced Semicolon

- Adding a semicolon at the end of the **for** clause before the loop body is a common mistake — **a logic error**:

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

Logic Error

- With the misplaced semicolon, the loop body is empty.
- The code block below the loop will execute after the loop completes.

Nested Loops

- The placement of one loop inside the body of another loop is called **nesting**.
- When we **nest** two loops, the outer loop takes control of the number of complete repetitions of the inner loop.
- Any type of loop may be nested.
- Following is the basic Syntax for the "Nested for loop":

```
// ---outer loop---
for (int i = 1; i <= 5; ++i) {
  // codes in the body of the outer loop

  // ---inner loop--
  for(int j = 1; j <=2; ++j) {
    // code in the body of both outer and inner loops
     }
  }
```

Nested Loop Example

```
public class nestedloopexample{
  public static void main(String[] args) {

    int weeks = 3;
    int days = 7;

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

      // inner loop prints days
      for (int j = 1; j <= days; ++j) {
        System.out.println("  Day:  " + j);
      }
    }
  }
}
```

Output

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

Hands-On Lab - For loop

Complete **LAB- 3.3.1- For Loop.** You can find this lab on Canvas under the **Guided Lab section.**

Use your office hours to complete this Lab. If you have technical questions while performing the Lab activity, ask your instructors for assistance.

Topic 1b: Enhanced ***for*** Loops

- The **for** statement has another form designed to iterate through arrays and collections.
- The syntax of the Java enhanced loop is that it can be applied to arrays, various collection classes, and any class implementing the **<u>Iterable</u>** interface.

```
for(dataType item : array) {
    statement ....
 }
```

In the example above:

- array - an array or a collection.
- item - a variable array/collection assigned.
- dataType - the data type of the array/collection.

Enhanced ***for*** Loop (continued)

- Syntax:

```
for (T element:Collection obj/array)
{
    statement(s)
}
```

- Example:

```
public class enhanceforloop {
   // Java program to illustrate enhanced for Loop
   public static void main(String args[])
   {
     String movies [] = {"Braveheart","Troy","Life is beautiful"};
       //enhanced for L0op
       for (String value: movies) {
           System.out.println(value);
       }
       // for Loop for same function
       //for (int i = 0; i < movies.length; i++) {
       //   System.out.println(movies[i]);
       //}
   }
```

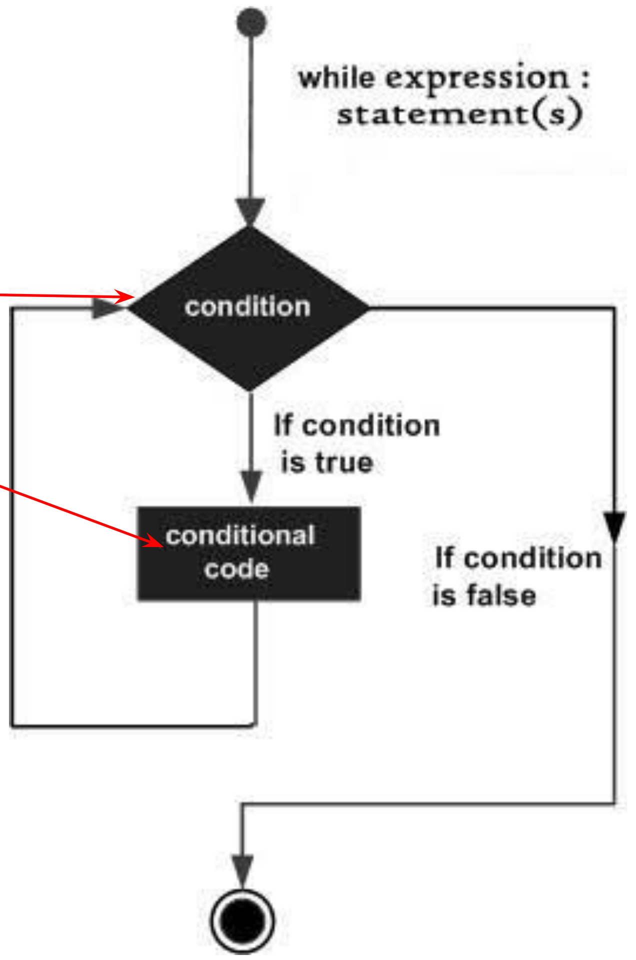
Topic 1c: Introduction to *while* Loops

- As the name suggests, ***while loop*** will continually process a statement or given code block while its evaluation ***condition*** is true.
- Syntax:

```
while (condition){
   //statement(s)
   // statement1...n
   condition = false;
}
```

while Loop Flowchart

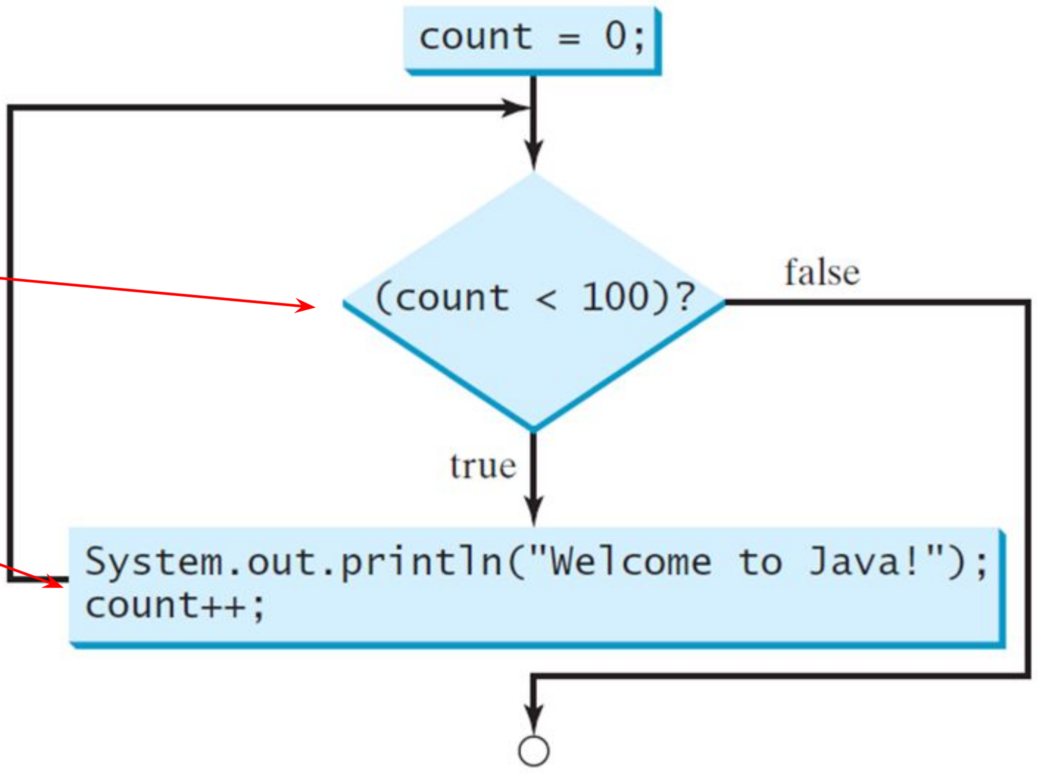
```
while (condition){
      //statement(s);
}
```



The *condition* must evaluate to ***true*** for the body of the **while loop** to execute (just like the *if* statement and *for* loop).

while Loop Flowchart Example

```
int count = 0;
while (count < 100){
   System.out.println("Welcome to Java!");
   count++;
}
```



Trace *while* Loop

```
int count = 0;
while (count < 2) {
        System.out.println("Welcome to Java!");
        count++;
}
System.out.println("Goodbye!");
```

Initialize *count*.

```
int count = 0;
while (count < 2) {
        System.out.println("Welcome to Java!");
        count++;
}
System.out.println("Goodbye!");
```

(*count* < 2) is true.

Trace *while* Loop (continued)

```
int count = 0;
while (count < 2) {
        System.out.println("Welcome to Java!");
        count++;
}
System.out.println("Goodbye!");
```

Print "Welcome to Java!"

```
int count = 0;
while (count < 2) {
        System.out.println("Welcome to Java!");
        count++;
}
System.out.println("Goodbye!");
```

Increment *count* by 1;
count is now 1.

Trace *while* Loop (continued)

```
int count = 0;
while (count < 2) {
        System.out.println("Welcome to Java!");
        count++;
}
System.out.println("Goodbye!");
```

(*count* < 2) is still true since *count* is 1.

```
int count = 0;
while (count < 2) {
        System.out.println("Welcome to Java!");
        count++;
}
System.out.println("Goodbye!");
```

Print "Welcome to Java!"

Trace *while* Loop (continued)

```
int count = 0;
while (count < 2) {
        System.out.println("Welcome to Java!");
        count++;
}
System.out.println("Goodbye!");
```

Increase *count* by 1; *count* is 2 now.

```
int count  = 0;
while (count < 2) {
        System.out.println("Welcome to Java!");
        count ++;
}
System.out.println("Goodbye!");
```

(*count* < 2) is false since *count* is 2 now.

Trace *while* Loop (continued)

```
count = 0;
while (count < 2) {
        System.out.println("Welcome to Java!");
        count ++;
}
System.out.println(“Goodbye!");
```

Loop exits.
Execute the next statement after the loop.

- Thus far, all of our programs have been *one-offs*.
  - You run the program once, and then it is done.
  - If you want to play again, you have to run the program again.
- Using loops, you can offer your users the option of “Do you want to play again?”

Caution: Floating-Point Equality is Inexact

Do not use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in inaccurate counter values and results.

Consider the following code for computing:

1 - 0.1 - 0.1 - 0.1 – 0.1 …

```
double value = 1.0;
while (value != 0
   value -= 0.1;
   System.out.println(value);
}
```

There is no guarantee that this will ever evaluate true; This may become an **infinite loop**.

Omitting the Loop-Continuation-Condition

- If the loop-continuation-condition in a **for** loop is omitted, it is implicitly true.
- The statement (a) is valid Java.
  - It is an infinite loop.
- The statement (b) is preferred just to avoid confusion.

```
for ( ; ; ) {
  // Do something
}
```

(a)

Equivalent

```
while (true) {
  // Do something
}
```

(b)

Ending a Loop With a Sentinel Value

- Often, the number of times a loop is executed is not known ahead of time.
- You may use a special input value to signify the end of the loop. Such a value is called a *sentinel value*.

Example: **While** loop

Make a new class and call it ***whileloopexample***. Inside, write the code shown below:

```java
public class Whileloopexample {

    public static void main(String[] args) {

        int i = 0;

        while (i < 3) {
            System.out.println("Value of i = " + i);
            i++;
        }
    }
}
```

Output

```
Value of i = 0
Value of i = 1
Value of i = 2
```

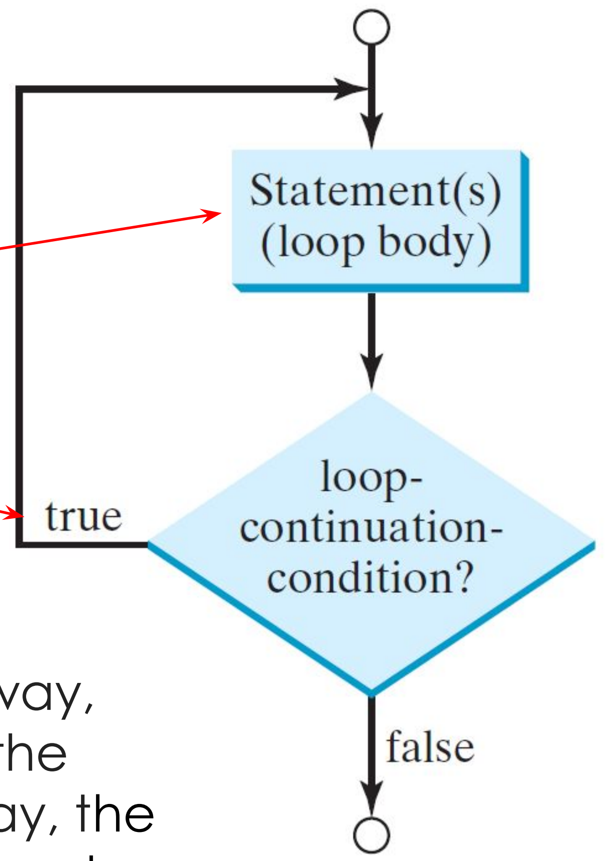
Hands-On Lab - While loop

Complete **LAB- 3.3.2 -While Loop.** You can find this lab on Canvas under the **Guided Lab section.**

Use your office hours to complete this Lab. If you have technical questions while performing the lab activity, ask your instructors for assistance.

Topic 1d: Introduction to ***do-while*** Loops

```
do {
   //Loop Body
   statement(s);
} while (loop-continuation-condition);
```



The ***do-while*** and ***while loop*** are similar, and work in a similar way, except that in a ***do-while loop***, an expression is evaluated at the bottom as compared to the ***while loop***. To put it in another way, the **do-while** loop guarantees that the loop body will execute at least once.

do-while Loop Examples

- Finite **do-while** loop:

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

- Looping while true:

```
do {
      System.out.println(“Infinite loop”);
} while(true);
```

Topic 1e: Use Which Loop?

- Use the loop that is most intuitive and most comfortable for you.

- In general, a **for** loop may be used if the number of repetitions is known. For example, you may need to print a message 100 times, or it can be supplied by the user.

- A **while** loop may be used if the number of repetitions is not known. For example, it may read numbers until the input is 0 or it may read data from a database.
- A **do-while** loop is a good choice when you are asking a question that will answer whether or not the loop is repeated.

Lesson 2

Introduction to Loop Control Statements

In this lesson, we will discuss and review the break and continue keywords.

Topic 2: Loop Control Statements

- Loop control statements — sometimes referred to as ***jump statements*** — change loop execution from its normal sequence. Java supports the following loop control statements:

  - **break** – terminates the loop statement and transfers execution to the statement immediately following the loop.
  - **continue** – causes the loop to skip the remainder of its body and immediately jump to the top of the loop.
    - In while loops and for loops, the loop-continuation-condition will be evaluated.

Please view our **Wiki** document for more information about break and continue keywords.

Example: Using ***break*** keyword in a Loop

```java
public class breakExample {
   public static void main(String[] args) {
       int sum = 0;
       int number = 0;
       while (number < 20) {
           number++;
           sum += number;
           if (sum >= 100) {
               break;
           }
       }
       System.out.println("The number is " + number);
       System.out.println("The sum is " + sum);
   }
}
```

This program adds the integers from 1 to 20, in this order, to sum until the sum is greater than or equal to 100.

Output:

```
The number is 14.
The sum is 105.
```

Without the *if* statement, the program calculates the sum of the numbers from 1 to 20.

Output:

```
The number is 20.
The sum is 210.
```

Example: Using ***continue*** keyword in a Loop

```
public class continueexample {
   public static void main(String[] args) {
       int sum = 0;
       int number = 0;
       while (number < 20) {
           number++;
           if (number == 10 || number == 11)
           {
               continue;
           }
           sum += number;
       }
       System.out.println("The sum is " + sum);
    }
}
```

This program adds all the integers from 1 to 20, except **10** and **11** to sum.

Output:

```
The sum is 189.
```

Without the *if* statement in the program, all of the numbers are added to sum, even when the number is 10 or 11.

Output:

```
The sum is 210.
```

Practice Assignment

After you learn about loops and related concepts, you can complete the Practice Assignment: **303.3.1 Practice Assignment - Loops.** You can find this assignment on Canvas under the **Assignment** section.

Use your office hours to complete this assignment. If you have any technical questions while completing the practice assignment, you can ask the instructors for assistance.

Summary

In this lesson, we discussed Iteration Statements (Loops).

- In general, a *for* loop may be used if the number of repetitions is known (e.g., when you need to print a message 100 times).
- A *while* loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.
- A *do-while* loop can be used to replace a *while* loop if the loop body has to be executed before testing the continuation condition. Use the *do-while* loop if you have statements inside the loop that must be executed at least once.

Reference

https://www.teamten.com/lawrence/programming/intro/intro8.html

https://math.hws.edu/eck/cs124/downloads/javanotes6-linked.pdf

Questions

TEKsystems
Own change

End of Module