
ORACLE
ACADEMY

Java Fundamentals

5-2
Control Statements

ORACLE ACADEMY

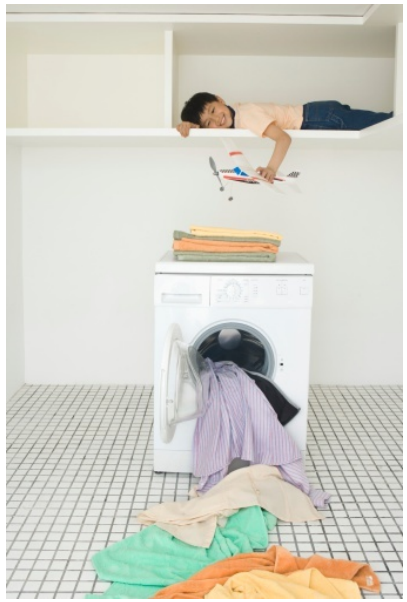
Objectives

This lesson covers the following objectives:

- Create a while loop
- Create a do-while loop
- Create a for loop

What is a Loop?

- Many everyday tasks involve doing an action, and then repeating the same procedure or action on a different object.
- When folding clean clothes, there are three basic steps:
  - Pick up the piece of clothing.
  - Fold it.
  - Put it away.
- For each piece of clothing, these actions are repeated. Each time you execute the action, only your input (the specific piece of clothing) is different.



Loops

- In programming, there are times when you want to work with multiple inputs, but you want to execute the same logic for each input item.
- A loop allows you to have a series of inputs with the same code.
- Loops will start at the beginning of a piece of code, execute the logic you wish, and then return to the beginning of the loop with new input, ready to execute the code once more.

Why are Loops Useful?

- Suppose you have a list of ten numbers and you want to find the sum of those numbers. You could create a statement like this:

```
sum = num1 + num2 + num3 + num4 + … + num10;
```

- While this is fairly simple code, using a loop will simplify the code further.

```
loop (loop condition){
       input currentNumber
       sum = sum + currentNumber;
}end loop
```

When the loop executes the first time,
the input will accept a number, and for
each time the loop runs, the sum will increase by that number.

Loop Control: Stopping the Loop

- For code to enter a loop and execute the code inside, the loop condition must be true.
- To end the loop, the loop condition must be false.
- When creating loops in Java, a condition must change from true to false in order for code to exit the loop.

Stopping Conditions

- A loop needs a stopping condition, which could be specified as:
  - A set number of times to run the code.
  - A boolean condition that is changed in the code to make the loop stop executing.

Types of Loops

- Java has three basic types of loops that work with these two types of stopping conditions:
  - while loops
  - for loops
  - do-while loop

Pre-test loops: The condition is tested prior to execution of the loop. If the condition is false, the loop will stop, or may never execute.

Post-test loop: The condition is tested after the each execution of the loop. If the condition is false, the loop will still execute at least once, but will stop at the end of the loop.

The while Loop

- The while loop is designed to loop while something remains true.
- Example condition:
  - While there are more numbers to enter.
- Think of the example as a true/false condition, or boolean condition:
  - If the condition "There are more numbers to enter" is true, continue to accept input.
  - When the condition "There are more numbers to enter" is false, stop accepting input.

Infinite Loops

- If you do not allow for a change in the condition, the loop will run forever as an infinite loop.

Java Syntax for while Loops

- With a while loop, Java uses the syntax:

```
while(condition is true){
        //logic
   }
```

- Similar to if statements, the while loop parameters can be boolean types, or can produce a boolean value.
- Conditional statements using (<, >, <=, >=, !=, ==) produce boolean values.
- Examples:

```
while (num1 < num2)

while (isTrue)

while (n !=0)
```

While Loop Example

- Implementation using a while loop is shown below.
- This example adds a sequence of ten integers that are input by the user.

Notice that the loop condition is tested before each execution of the loop. This is a pre-test loop.

```
import java.util.Scanner;
public class LoopPractice{
     public static void main(String[] args){
           Scanner in = new Scanner(Syste[illegible]n);
           int numInputs = 0;
           int sum = 0;
           while(numInputs < 10){          //condition to be tested each time loop is executed
              input = in.nextInt();        //user inputs a number
              sum+=input;                  //add user input to sum
              numInputs++;                 //statement that will change the loop condition
           }
           System.out.println("The sum of those ten numbers is: " + sum);
           }
     }
}
```

Using a while Loop with String Methods

- A palindrome is a word spelled the same forward or backward. Examples: deed, racecar, and level.
- Write Java code that asks for a word and returns true if it is a palindrome, and false if it is not.
- This code should:
  - Calculate the length of the word.
  - Compare the first and last letters if they match.
  - Compare letters until the middle of the word is reached.

Using a while Loop with String Methods Example

- Palindrome method code:

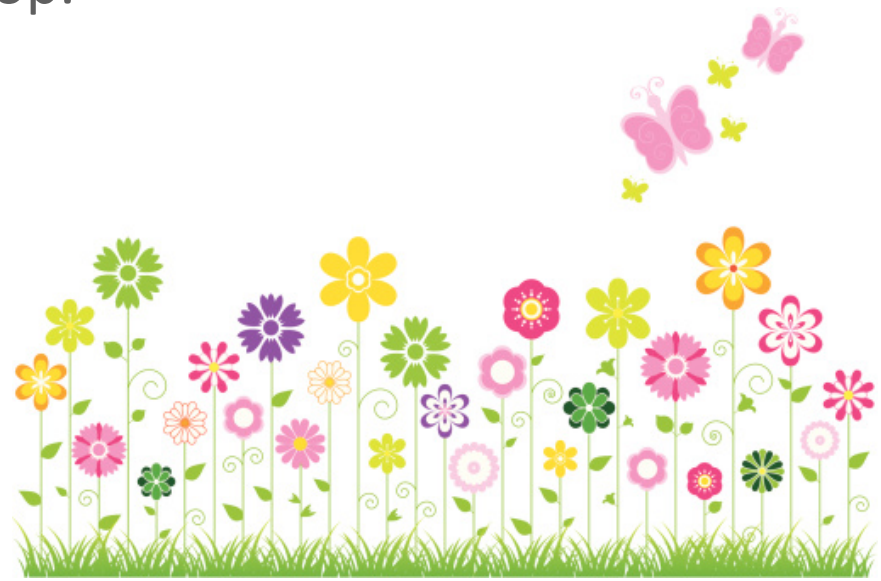
```
public class PalindromeTester{
  public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    System.out.println("Enter a word:");
    String word = in.next();
    int firstPosition = 0;
    int lastPosition = word.length() - 1;
    boolean isAPalindrome = true;
    while(firstPosition < lastPosition){
        if(word.charAt(firstPosition)!=word.charAt(lastPosition))
            isAPalindrome = false;
        firstPosition++;
        lastPosition--;
    }
    if(isAPalindrome)
      System.out.println("The word is a Palindrome");
    else
       System.out.println("The word is not a Palindrome");
  }
}
```

The do-while Loop

- The do-while loop:
  - Is a post-test loop.
  - Is a modified while loop that allows the program to run through the loop once before testing the boolean condition.
  - Continues until the condition becomes false.
- If you do not allow for a change in the condition, the loop will run forever as an infinite loop.

The do-while Loop Insect Example

- Consider the movement of a flying insect that lands on each flower it sees.
- To recreate this insect's movement in code, you could use a do-while loop.



Insect Action Pseudocode

- Pseudocode for the movement of a flying insect that lands on each flower it sees:
  - The insect is initially flying, before any condition is tested.
  - The insect continues to fly until it spots a flower.
  - The condition that will be tested is noFlowerSpotted.
  - When noFlowerSpotted is true, the insect will continue flying. When it is false, the insect will stop going through the loop and land on the flower.

Java Syntax for do-while Loops

- The do-while loop uses the same boolean logic that is used for a regular while loop.
- The do code block is run once, and the while condition is tested at the end of each run of the code.
  - First, the do code block is executed.
  - Next, the condition is tested.
  - This repeats until the condition becomes false.
- do-while loop syntax:

```
do{
        //statements to repeat go here
} while(condition);
```

Do not forget the semicolon or the code will not compile.

The do-while Loop Insect Example Code

- The insect flies until it finds a flower, then lands.
- Step 1: The insect begins to fly before testing if it can see a flower.
- Step 2: Test if the insect sees a flower.
  - If the insect does not see a flower, repeat steps 1 and 2.
  - If the insect sees a flower, go to the final step.

```
do {
        insect.fly();                   //Step 1
} while(noFlowerFound)                  //Step 2
insect.land();                          //Final Step
```

do-while Loop Example

- Implementation using a do-while loop is shown below. This example also a sequence of ten integers that are input by the user. Can you see and explain the differences in this example and the pre-test loop?

```
import java.util.Scanner;
public class LoopPractice{
     public static void main(String[] args){
          Scanner in = new Scanner(System.in);
          int numInputs = 0;
          int sum = 0, input = 0;
          do{
             System.out.println("Enter a nu      );
             input = in.nextInt();          //user inputs a number
             sum+=input;                    //add user input to sum
             numInputs++;       //statement that will change the loop condition
          } while(numInputs <  10);
          System.out.println("The sum of those ten numbers is: " + sum);
     }
}
```

Notice that the loop condition is tested after each execution of the loop. This is a post-test loop.

do-while Loop Example

```
import java.util.Scanner;
public class LoopPractice2{
     public static void main(String[] args){
          Scanner in = new Scanner(System.in);
          int repeat = 0;
          do{
             System.out.println("Enter a number");
             input = in.nextInt();    //user inputs a number
             sum+=input;              //add user input to sum
             System.out.println("Do you want to enter another number?");
             System.out.println("Enter 1 for yes or 2 for no:");
             repeat = in.nextInt();
          } while(repeat==1);
          System.out.println("The sum of the numbers is: " + sum);
     }
}
```

The for Loop

- Recall the folding clothes example.
- If we know how many pieces of clothing we have, we know exactly when the condition "There are more clothes" will be false.
- for loops tell the loop when to stop by explicitly saying "Stop when the loop has run once for every piece of clothing."
- For example, if we have ten pieces of clothing, we can tell the loop, "Run 10 times" since we know that after the tenth execution, there will not be any more clothes.

for Loop Java Syntax

- for loop syntax contains three parts:
  - Initializing the lcv (loop control variable).
  - Conditional statement, or stopping condition.
  - Updating the counter (going to the next value).
    - Think of i as a counter, starting at 0 and incrementing until i=timesToRun.
- for loop syntax:

for loop naming conventions typically will use i as the lcv variable name.

```
for(int i=0; i < timesToRun; i++){
    //logic
}
```

for Loop Explained

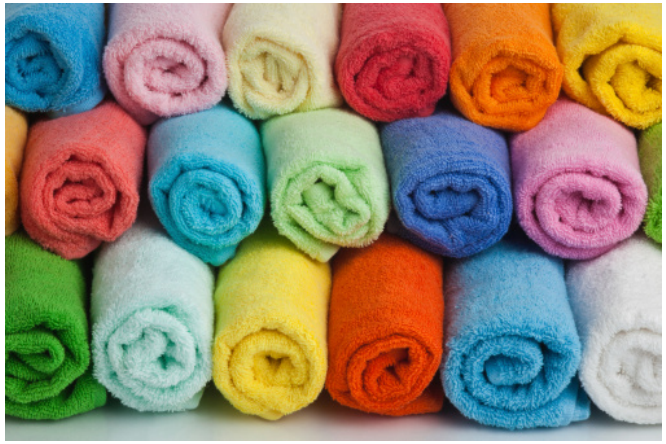
- The for loop has the following components:
- Initializing the counter variable (i).
  - Often, i is set to 0 (zero), thereby starting the count at 0.
- Conditional statement, or stopping condition.
  - The loop will continue to execute while i < timesToRun, which means it will run timesToRun times.
- Updating the counter (going to the next value).
  - Every time the loop runs, i is incremented by one (the ++ operator adds one to i).

```
for(int i=0; i < timesToRun; i++){
    //statements to repeat
}
```

for Loop Example 1

- Review the example code below to fold ten articles of clothing.
- Can you identify the three parts of the for loop in this example?

```
for(int i = 0; i < numFolded; i++)
{
    fold();
}
System.out.println("All Done!");
```



for Loop Example 2

- This example is a sequence of ten integers that are input by the user. Can you see and explain the differences in this example and the post-test loop?

```
import java.util.Scanner;
public class LoopPractice{
     public static void main(String[] args){
          Scanner in = new Scanner(System.in);
          int numInputs = 10;
          int sum = 0;
          for(int i = 0; i < numInputs; i++){
             input = in.nextInt();    //user inputs a number
             sum+=input;  //add user input to sum
          }
          System.out.println("The sum of those ten numbers is: " + sum);
     }
}
```

The loop condition is tested before each execution of the loop. This is a pre-test loop.

Which Loop Do I Use?

Loop Type	Definition	When to Use
while	Pre-test loop that repeats until a specified condition is false.	Use when you are not certain the number of times the loop should be executed or even if it should at all.
do-while	Post-test loop that executes the loop before testing the condition, then repeats until the condition is false.	Use when you know that the code must be executed at least once and possibly more times depending on the condition.
for	Loop that contains an initialized counter, and increments the counter with each run through the loop. Repeats until the condition is false.	Use when you need to execute a loop a specific number of times, or when you need to increment through a set of data. The counter can also be used as an index for accessing data one item at a time.

Using break and continue

- break and continue are both keywords in Java that aid in controlling the flow of your program.
- The keyword break is useful for instances where you wish to exit a loop at a specified point that is different than the condition statement.
- Using break in a loop will cause code to exit the loop.

Using break and continue

- The keyword continue is useful for special cases where you want to exclude code for a particular element in a list.
- Using continue will cause the code to skip the rest of the code in the loop and evaluate the condition statement (in a for loop, the lcv will also increment).

break Example

- This example will ask user to input a number ten times.
- If the user enters the value 999, the loop will terminate regardless of the value of i (the lcv).

```
import java.util.Scanner;
public class BreakExample{
     public static void main(String[] args){
           Scanner in = new Scanner(System.in);
           int numInputs = 10, input = 0, sum = 0, stopLoop = 999;
           for(int i = 0; i < numInputs; i++){
              input = in.nextInt();    //user inputs a number
              if(input==stopLoop)   //if the number is 999, exit the loop without adding to the sum
                  break;
              else
                  sum+=input;          //if the number is not 999, add it to the sum
           }
           System.out.println("The sum of the numbers entered is: " + sum);
     }
}
```

continue Example

- Given a list of integers (you want to output a message for all the odd numbers, and you want to skip over the even numbers), this can be done using the following code.

```
import java.util.Scanner;
public class ContinueExample{
     public static void main(String[] args){
          Scanner in = new Scanner(System.in);
          int numInputs = 10, input = 0;
          for(int i = 0; i < numInputs; i++){
             input = in.nextInt();                          //user inputs a number
             if(input % 2 == 0)                             //if it's even
             continue;   //skip the remaining code in the loop, increment i, re-check the condition
             System.out.println("That number was odd");   //output only if odd
          }
     }
}
```

if, do-while, and switch Example

```
import java.util.Scanner;
public class Section5Example{
  public static void main(String[] args){
    boolean quit = false;
    int num1 = 10, num2 = 6, answer = 0;
    char operand = '';
    Scanner in = new Scanner(System.in);
    do {
        System.out.println("Please enter a mathematical operand");
        String input = in.next();
        char operand = input.charAt(0);
              switch(operand) {
              case '*':
                  answer = num1 * num2;
                  break;
              case '+':
                  answer = num1 + num2;
                  break;
              case '-':
                  answer = num1 - num2;
              case '/':
                  answer = num1 / num2;
              default:
                  System.out.println("Invalid input.");
            }
         System.out.println("Quit? Y/N");
         if(in.next().equalsIgnorecase("Y"))
               quit=true;
    } while(!quit);
   }
}
```

Terminology

Key terms used in this lesson included:

- break
- continue
- do-while loop
- for loop
- while loop

Summary

In this lesson, you should have learned how to:

- Create a while loop
- Create a do-while loop
- Create a for loop


ORACLE
ACADEMY