

## Chapter 7 - Iteration

### Chapter Goals

- To be able to program loops with the `while`, `for`, and `do` statements
- To avoid infinite loops and off-by-one errors
- To understand nested loops
- To learn how to process input
- To implement simulations

### Terminology

iteration, `while` statement, block statement, loop, infinite loop, off-by-one error, test cases, `do` loop, `for` loop, boolean control variable, loop counter variable, initialize, test, update, scope, nested, inner loop, outer loop, sentinel value, loop and a half, control variable, symmetric and asymmetric bounds, upper and lower bounds, `break` and `continue` statements, random number, simulation, pseudorandom

### Syntax

The `while` statement

```
while (condition)
    statement
```

```
while (condition)
{
    statements
    . . .
}
```

The `do` loop

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

```
do
{
    statements
    . . .
}
while (condition);
```

The `for` statement

```
for (initialization; condition; update)
    statement;
```

```
for (initialization; condition; update)
{
    statement(s)
    . . .
}
```

### Common Errors

1. Infinite Loops
  - a. Forgetting to update the variable that controls the loop
  - b. Incrementing when you should be decrementing or vice versa.
2. Off-By-One Errors
3. Forgetting a Semicolon
4. A Semicolon Too Many
5. Using Commas instead of Semicolons in `for` Statements

## Implementing Loops

1. List the work that needs to be done in every step of the loop body.
2. Find out how often the loop is repeated.  
If a loop is executed for a definite number of times, a `for` loop is usually appropriate.
3. With a `while` loop, find out where you can determine that the loop is finished.  
There are three possibilities:  
Before entering the loop  
In the middle of the loop  
At the end of the loop
4. Implement the loop by putting the operations from Step 1 into the loop body
5. Double-check your variable initializations
6. Check for off-by-one errors

## Chapter Summary

1. A `while` statement executes a block of code repeatedly. A condition controls how often the loop is executed.
2. An off-by-one error is a common error when programming loops. Think through simple test cases to avoid this type of error.
3. You use a `for` loop when a variable runs from a starting to an ending value with a constant increment or decrement.
4. Loops can be nested. A typical example of nested loops is printing a table with rows and columns.
5. Sometimes, the termination condition of a loop can only be evaluated in the middle of a loop. You can introduce a Boolean variable to control such a loop.
6. Make a choice between symmetric and asymmetric bounds for each `for` loop.
7. Count the number of iterations of a `for` loop to check that your loop is correct.
8. In a simulation, you repeatedly generate random numbers and use them to simulate an activity.

## Classes, Objects and Methods

`java.util.Random`

`nextDouble()` - returns a random floating-point number between 0 (inclusive) and 1 (exclusive)

`nextInt(n)` - returns a random integer between 0 (inclusive) and n (exclusive)