What we will cover:
The do-while Loop
The for Loop
Choosing a Loop

The do-while Loop
The do-while loop is a variation of the while loop. This type of loop executes one or more times since the loop test is at the bottom of the loop. The loop body is always executed at least once. The general format of the do-while loop is as follows:

initialize control objects;
do {
    statements;
    update control objects;
} while (loop-continuation-condition);

Take notice of braces and semi-colons. You must close the loop-test line with a semi-colon or you will receive an error. Using our “count to 100” example from the while loop lecture, we could rewrite it as a do-while loop as shown:

```java
int counter;
counter = 0;

do {
    System.out.println(counter);
counter++;
} while (counter <= 100);
```

Compare the flow charts of the while and do-while loop below.
Use the *do-while* loop when it reduces code complexity (recall the examples where we had to get input twice—once outside the loop and once inside it) and you have code that has to execute at least once.

```java
public class TextInputAverageDoWhile {

    public static void main(String[] args) {
        int counter = 0, data;
        int evenNumbers = 0;
        double sum = 0.0, average = 0.0;
        Scanner scanner = new Scanner(System.in);

        do {
            System.out.print("Enter a number (or 0 to quit): ");
            data = scanner.nextInt();
            if (data != 0) {
                sum += data;
                counter++;

                if (data % 2 == 0) {
                    evenNumbers++;
                }
            }
        } while (data != 0);

        average = sum/counter;
        System.out.println("Average: "+ average);
        System.out.println("# of Even Numbers: "+ evenNumbers);
    }
}
```

*Class Exercise 15 – CountingCompleted.java*

**The for Loop**

The *for* loop works similarly to the other loops we have studied. The general format of a *for* loop combines different aspects of the loop control and tests into a simplified general format:

```
for (pre-iteration-action; loop-continuation-condition; post-iteration action) {
    statements;
}
```

Take special notice of the use of semi-colons and braces. You must have a semi-colon after the initial-action and the loop-continuation-condition, but not one after the post-iteration-action. An extra semi-colon or blank clause in the *for* loop is considered *true*. This can cause problems with infinite loops, loop-tests, or the post-iteration-action.

Using our “count to 100” example, we could write it as a *for* loop as shown:

```java
int counter;
counter = 0;

for (counter = 0; counter <= 100; counter++) {
    System.out.println(counter);
}
```
The initial-action is performed first, the loop-continuation-condition is tested, and then the body is executed. The post-iteration-action is executed and the loop is started over, minus the initial-action. for loops use a control variable to control the loop’s iterations. We’ve been doing this with other loops, but notice how this particular loop condenses our actions. The initial-action typically initializes the control variable, the post-iteration-action typically increments or decrements the control variable and the loop-continuation-condition tests for whether or not to stop the loop. The convention for control variables in for loops is to use a variable named \( i \). Let’s rewrite the previous loop:

```java
int i;
i = 0;

for (i = 0; i <= 100; i++) {
    System.out.println(i);
}
```

In fact, it is further convention to both declare and initialize the control variable in the initial-action clause of the for statement. Let’s rewrite the previous loop once again:

```java
for (int i = 0; i <= 100; i++) {
    System.out.println(i);
}
```

Now, we have a nice compact, concise loop structure.

Compare the flow chart of the for loop to the previous flow charts and follow the code through the previous example:

---

Class Exercise 16 – CountingCompleted.java (again)
Choosing a Loop
Both the for and while loops are considered pretest loops. The do-while loop is a posttest loop based on where the loop test occurs. As noted in the book, these loops are expressly equivalent. You should be able to convert any loop from one form to the other. In general, for loops are used when the number of iterations is finite or known; in counting-type situations. while loops tend to be used when iterations need to be executed until a certain condition is met, and the do-while can replace a while loop where the body has to be executed at least once.

Learn the position of semi-colons, parentheses and braces for each type of loop. A misplaced semi-colon or missing braces can completely change the behavior of a loop. Similar to conditional statements, braces are optional with single-statement loop bodies. Because of this, you must be very observant when writing and forming loops to avoid empty body loops or infinite loops.

Class Exercise/Homework 17 – Converting Loops