# The While Loop

**while** (loop\_continuation-condition) {

//loop body

Statement(s);

}

Note: The **loop-continuation-condition** must always appear inside the paremtnesis. The braces enclosing the loop body can be omitted **only** if the loop body contains one or no statement.

# Controlling a Loop with a Sentinel Value

A common technique for controlling a loop is to designate a special value when reading and processing a set of values signifies the end of the loop. (ie often a zero)

**while** (data != 0) {

Caution: Don’t use floating-point values for equality checking in a loop control. Floating-point values are approximations for some values, and therefore imprecise.

# The do-while Loop

**do** {

// loop body

Statement(s);

} **while** (loop\_continuation\_condition);

Note: the **do-while** loop is most effective when you have statements inside the loop that must be executed at lesast once.

# The for Loop

This should be familiar;

i = initialValue; //initialize loop control variable

**while** (I < endValue) {

//Loop body

…

I++; //Adjust loop control variable

}

Better version;

**for** (I = initialValue; I < endValue; i++) {

// loop body

…

}

Therefore the syntax generally looks like;

**for** (initial-action; loop-continuation-condition;

action-after-each-iteration) {

//loop body;

statement(s);

}

Note: the control variable must be declared inside the control structure of the loop or before the loop.

If can also have a list;

**for** (int I = 0, j = 0; (I + j < 10); i++, j++) {

//do something

}

# Which loop?

The while and for loop are called pre-test loops because the continuation conditions is checked before the loop body is executed. The do-while loop is therefore called a post-test loop as the condition is checked after the loop body is executed.

These three, for do-while and while are equivalent, just suit different purposes.

Caution: Adding a semicolon at the end of a for clause before the loop body is a common mistake – it ends the loop prematurely – don’t do it.

# Keywords **break** and **continue**

**Break** immediately ends the innermost loop that contains it. Break, breaks out of the loop.

**Continue** ends only the current iteration. Program control goes to the end of the loop body. In other words, **continue** breaks out of an iteration.

You may also use the keyword **break** in a **switch** statement

See TestBreak.java and TestContinue.java

# (GUI) Controlling a Loop with a Confirmation Dialogue Box.

A sentinel-controlled loop can be implemented using a confirmation dialog. The answer *yes* or *no* continue or terminate the loop.

See SEntinelValueUsingConfirmationDialog.java

# Time to Practice…

1. Rewrite the programs TestBreak and TestContinue without using break and continue.
2. Identify the errors in this code

**public class** test {

**public static void**  main (String[ ] args) {

**int sum = 0;**

**int j = 1**

**for** (int i = 0; i < 10; i++){

sum +=1;

**if** (i < j);

System.out.println (i);

**else**

System.out.println(j);

}

**while** (j < 10);

**{**

**j++;**

**};**

do **{**

**j++;**

} **while** (j< 10)

}

}

1. Write a program that prompts the user to enter the number of students and each student’s name and score, and finally displays the student with the highest score.

1. Write a program that displays all the leap years, ten per line, in the twenty-first century (from 2001 to 2100).