Outline

- The `while` Statement
- Other Repetition Statements

Repetition Statements

- ***Repetition statements*** **allow us to execute a statement multiple times**
- **Often they are referred to as *loops***
- **Like conditional statements, they are controlled by boolean expressions**
- **Java has three kinds of repetition statements:**
  - **the *while loop***
  - **the *do loop***
  - **the *for loop***

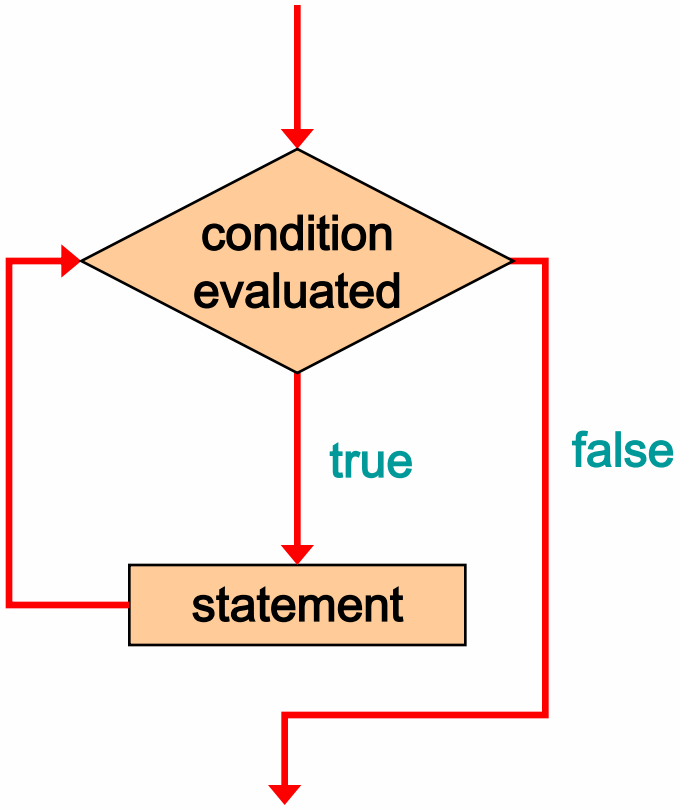
The while Statement

- A *while statement* has the following syntax:

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

- If the `condition` is true, the `statement` is executed
- Then the condition is evaluated again, and if it is still true, the statement is executed again
- The statement is executed repeatedly until the condition becomes false

Logic of a while Loop



The while Statement

- **An example of a while statement:**

```
int count = 1;
while (count <= 5){
    System.out.println (count);
    count++;
}
```

- **If the condition of a `while` loop is false initially, the statement is never executed**
- **Therefore, the body of a `while` loop will execute zero or more times**

The while Statement

- Let's look at some examples of loop processing
- A loop can be used to maintain a *running sum*
- A *sentinel value* is a special input value that represents the end of input

- A loop can also be used for *input validation*, making a program more *robust*

Infinite Loops

- **The body of a `while` loop eventually must make the condition false**
- **If not, it is called an *infinite loop*, which will execute until the user interrupts the program**
- **This is a common logical (semantic) error**
- **You should always double check the logic of a program to ensure that your loops will terminate normally**

Infinite Loops

- **An example of an infinite loop:**

```
int count = 1;
while (count <= 25){
    System.out.println (count);
    count = count - 1;
}
```

- **This loop will continue executing until interrupted (Control-C) or until an underflow error occurs**

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- For each iteration of the outer loop, the inner loop iterates completely
- Your second course project involves a `while` loop nested inside of a `for` loop

Nested Loops

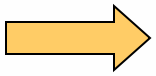
- **How many times will the string `"Here"` be printed?**

```
count1 = 1;
while (count1 <= 10){
    count2 = 1;
    while (count2 <= 20)    {
        System.out.println ("Here");
        count2++;
    }
    count1++;
}
```

10 * 20 = 200

Outline

The `while` Statement



Other Repetition Statements

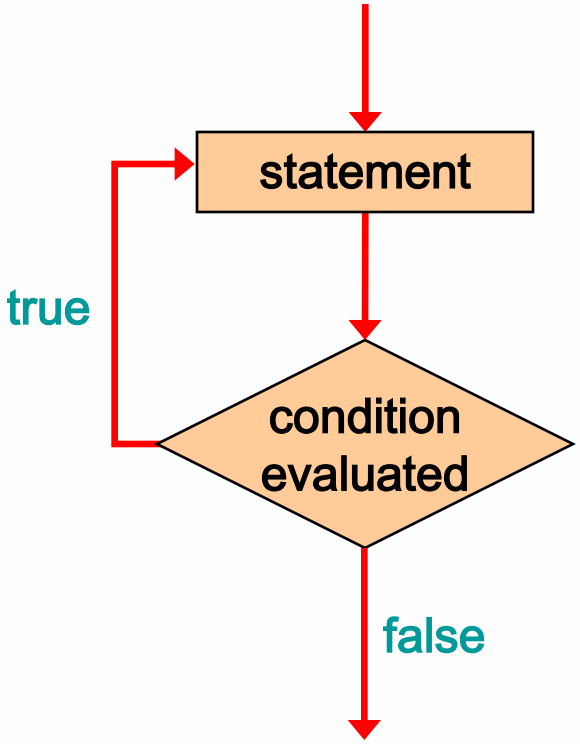
The do-while Statement

- **A *do-while statement* (also called a `do` loop) has the following syntax:**

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

- **The `statement` is executed once initially, and then the `condition` is evaluated**
- **The statement is executed repeatedly until the condition becomes false**

Logic of a do-while Loop



The do Statement

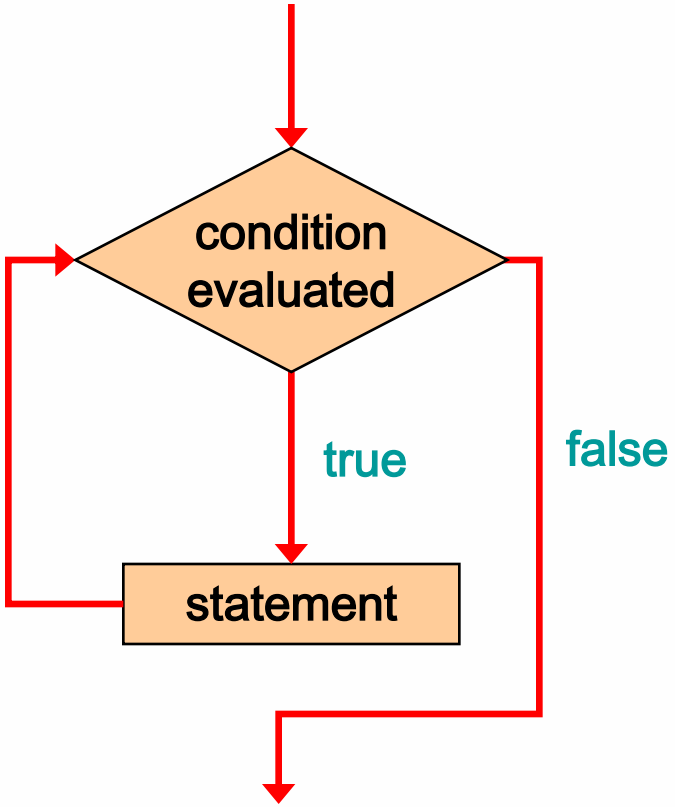
- **An example of a `do` loop:**

```
int count = 0;
do{
   count++;
   System.out.println (count);
} while (count < 5);
```

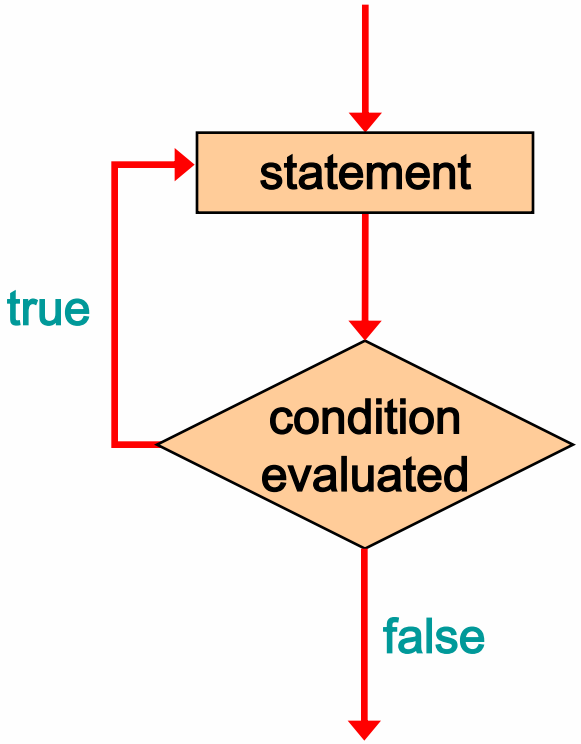
- **The body of a `do` loop executes at least once**

Comparing while and do

The while Loop



The do Loop



The for Statement

- **A *for statement* has the following syntax:**

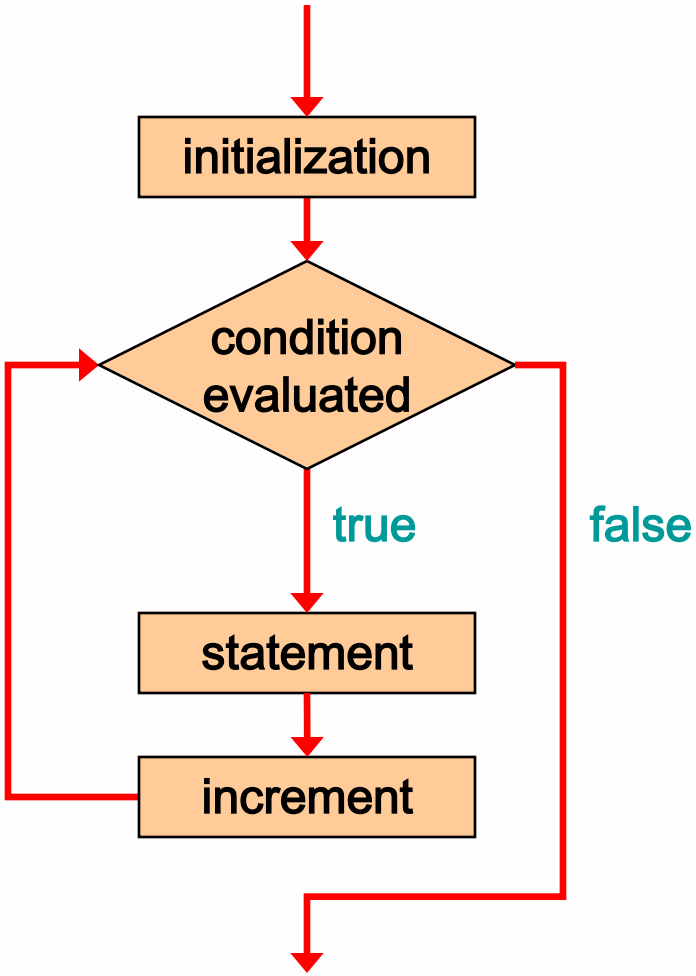
The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment ){
   statement;
}
```

The *increment* portion is executed at the end of each iteration

Logic of a for loop



The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while ( condition ){
    statement;
    increment;
}
```

The for Statement

- **An example of a `for` loop:**

```
for (int count=1; count <= 5; count++){
   System.out.println (count);
}
```

- **The initialization section can be used to declare a variable**
- **Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop body**
- **Therefore, the body of a `for` loop will execute zero or more times**

The for Statement

- **The increment section can perform <u>any calculation</u>**

```
for (int num=100; num > 0; num -= 5){
   System.out.println (num);
}
```

- **A `for` loop is well suited for executing statements a *specific number of times* that can be calculated or determined *in advance***

The for Statement

- Each expression in the header of a `for` loop is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
  - We usually call this a “forever loop”
- If the increment is left out, no increment operation is performed

`break` revisited

- Remember the `break` keyword that we used to stop a `switch` statement from executing more than one statement?
- `break` can also be used to exit an infinite loop
- But it is almost always best to use a well-written while loop