Introduction to Repetition Structures (`while` and `do` Loops)
Norton CS139

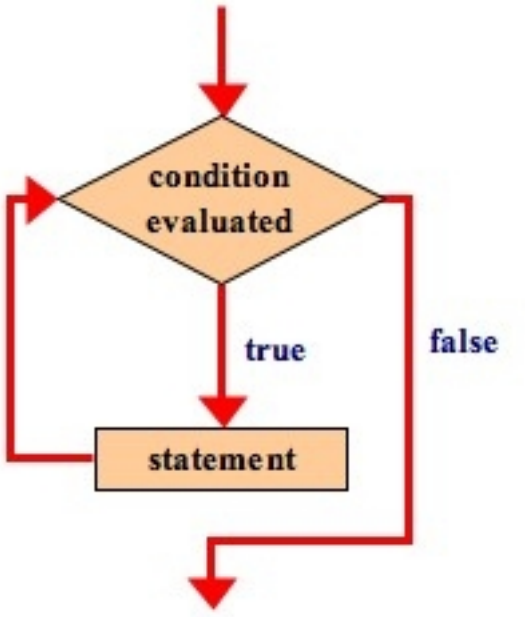
- Repetition Structures
  - Repetition statements allow us to execute a statement multiple times
  - Often they are referred to as loops
  - Like conditional statements, they are controlled by `boolean` expressions
  - Java has three kinds of repetition statements:
    - the `while` loop
    - the `do` loop
    - the `for` loop
  - The programmer should choose the right kind of loop for the situation

- 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



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

```
int count = 0;
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

- 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
    - See **Average.java**
  - A loop can also be used for input validation, making a program more robust
    - See **WinPercentage.java**

- 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 error
  - You should always double check the logic of a program to ensure that your loops will terminate normally
  - An example of an infinite loop:

```
int count = 0;
while ( count < 25 )
{
     System.out.println( count );
     --count;
}
```

  - 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
  - See **PalindromeTester1.java**

  - In the following example, how many times will the string `"Here!!"` be printed?

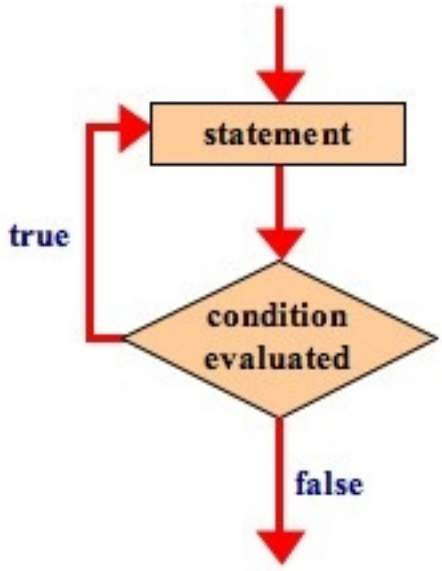
```
count1 = 0;
while ( count1 < 10 )
{
      count2 = 0;
      while ( count2 < 20 )
      {
            System.out.println( "Here!!" );
            ++count2;
      }
      ++count1;
}
```

- The `do` Statement
  - A do statement 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

- The Logic of a `do` Loop



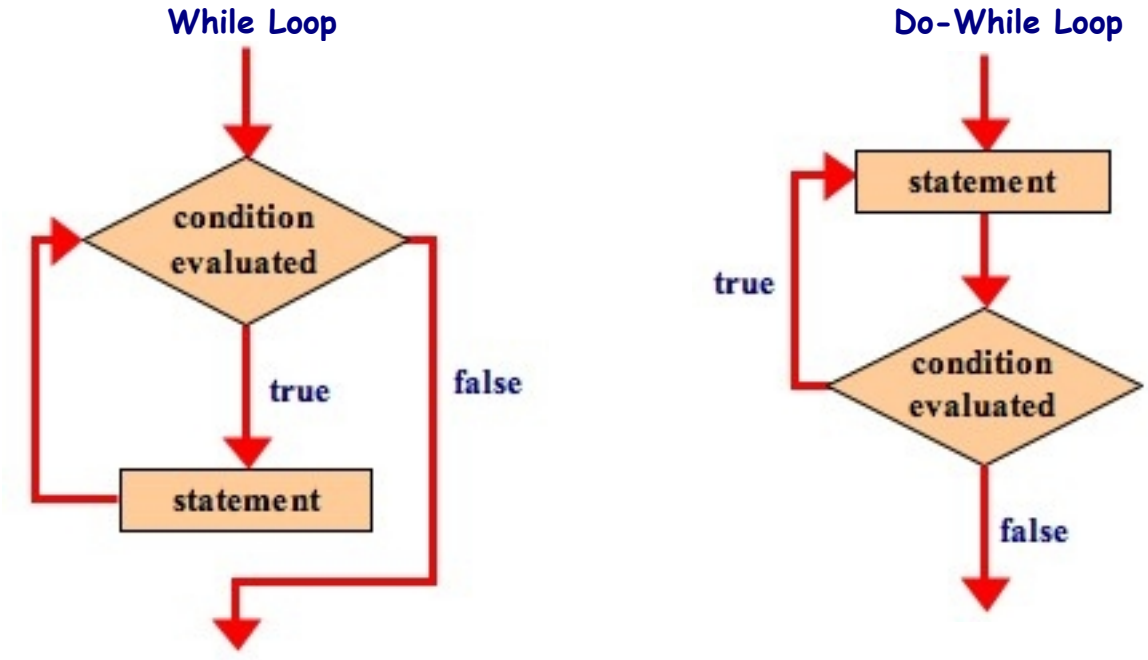
- An example of a `do` loop:

```
int count = 0;
do
{
     ++count;
     System.out.println( count );

} while ( count < 5 );
```

  - The body of a `do` loop executes at least once
  - See **ReverseNumber.java**
  - See also **PalindromeTester2.java**

- Comparing **`while`** and **`do`**



- This brings up the need to initialize variables
  - If a variable is given its value only within a block structure, such as a decision or repetition stucture, then it is possible that the variable will never be assigned a value.
  - Therefore, it is important to initialize all variables before the conditional or loop structure begins.
  - For example, the last line of the following will fail (actually this will not compile), since it is possible that myInt will never have been given a value:

```
int myInt;
int yourInt;

while ( someConditionIsTrue )
{
      myInt += 6;
      someConditionIsTrue = checkIfSomeConditionRemainsTrue();

}

yourInt = myInt * 10; // this will cause the compiler to fail!!
```

  - To prevent the error, you need to make sure to initialize **`myInt`** before the loop begins (set **`myInt = 0`** or some other appropriate value). This holds true for if/else structures also.

- Types of Loops
  - Physical Structures
    - **`while`**
      - 0 or more iterations
    - **`do/while`**
      - 1 or more iterations
  - Logical Structures
    - Boolean Loop (generic loop)
      - Loop runs **`x`** number of iterations
      - Will repeat until the condition becomes false
      - Can be **`while`** or **`do/while`** loop, depending on whether or not the first iteration is required
    - Sentinel Loop
      - A Boolean loop whose exit is determined by a sentinel value (a flag)
      - Used with reading files (end of file marker is the sentinel)
        - Read until you get to the end of file.
      - Could be **`while`** or **`do/while`**, depending on the number of iterations
    - Menu Loop
      - A sentinel loop whose flag is supplied by the user.
        - Keep going until the user decides to quit
      - A minimum of 1 iteration is required, thus you should use a **`do/while`** loop here.
    - Validation Loop
      - A Boolean loop whose exit s caused by the entry of correct data (seems strange, doesn't it)
        - Repeat until the user gets it right!!!
        - Used to enforce data integrity and "correctness".
      - Depending on circumstances, could be a **`while`** or a **`do/while`** loop (what circumstances, you might ask?)
    - We will revisit these looping "types" next week.