Lecture 5: Control Flow Statements

CSC 1214: Object-Oriented Programming

Control Flow Statements

- Conditionals/decision-making statements
  - The `if` statement
  - The `if..else` statement
  - The `switch` statement
- Iterations/looping statements
  - The `while` loop
  - The `do..while` loop
  - The `for` loop
- Branching statement
  - The `break` statement
  - The `continue` statement

Control Flow Statements

- So far, most of our methods include statements that execute **sequentially** from the first statement to the last one.

```
class CarTest {
  public static void main(String args[]) {
    <statement 1>
    <statement 2>
    <statement 3>
    <statement 4>
    <statement 5>
    <statement 6>
  }
}
```

Sequential execution

- Control flow statements modify that order, allowing us to **decide whether or not** to execute a particular statement, or execute a statement **over and over, repetitively**.

Control Flow Statements

- Conditional statements: a conditional statement lets us choose which statement will be executed next.

  Java's conditional statements are:
  - The `if` statement
  - The `if..else` statement
  - The `switch` statement

- Iteration/looping statements: a looping statement lets us execute a statement several times.

  Java's looping statements are:
  - The `while` loop
  - The `do..while` loop
  - The `for` loop

Control Flow Statements

- Conditional statements: a conditional statement lets us choose which statement will be executed next.

  Java's conditional statements are:
  - The **`if`** statement
  - The **`if..else`** statement
  - The **`switch`** statement

- Iteration/looping statements: a looping statement lets us execute a statement several times.

  Java's looping statements are:
  - The **`while`** loop
  - The **`do..while`** loop
  - The **`for`** loop

The if Statement

- The `if` statement in Java has the following syntax

```
if ( condition )
   statement;
```

The if Statement

- The `if` statement in Java has the following syntax

`if` is a Java reserved word

```
if ( condition )
    statement;
```

The if Statement

- The `if` statement in Java has the following syntax

`if` is a Java reserved word

The *`condition`* must be a boolean expression. i.e., It must evaluate to either `true` or `false`.

```
if ( condition )
   statement;
```

The if Statement

- The `if` statement in Java has the following syntax

The *condition* must be a boolean expression. i.e., It must evaluate to either `true` or `false`.

`if` is a Java reserved word

```
if ( condition )
    statement;
```

If the *condition* is true, the *statement* is executed. If it is false, the *statement* is skipped.

The if Statement

- An example of an `if` statement in Java

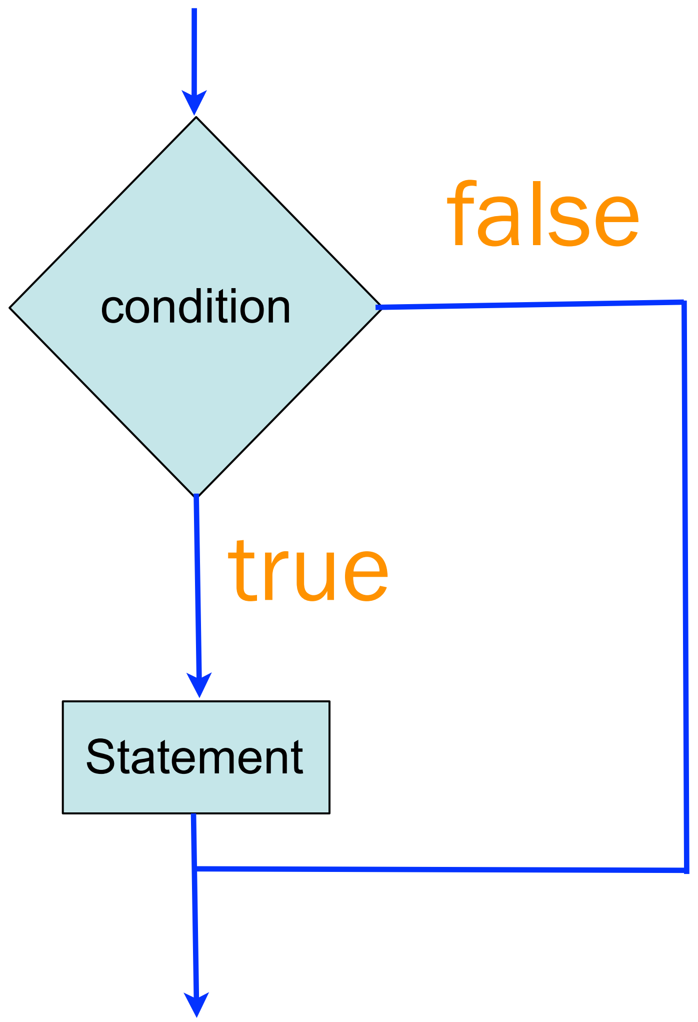
```
class Thermometer {
   public static void main(String args[]) {
      double currentTemp = 40.0;

      System.out.println("Current temperature is "+ currentTemp);
      if (currentTemp > 30.0)
         System.out.println(" It is too hot");
      }
}
```

First, the condition is evaluated. The value of **`currentTemp`** is either greater than the value of **`30.0`**, or it is not.

If the condition is true, the **`System.out.println(" It is too hot")`** statement is executed. If it is not, the statement is skipped.

Logic of an if Statement



Boolean Expressions

- A condition often uses one of Java's equality operators or relational operators, which all return a boolean value:

`==`	equal to
`!=`	not equal to
`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to

- Note the difference between the equality operator (`==`) and the assignment operator (`=`)

The if..else Statement

- The `if..else` statement in Java has the following syntax

```
if ( condition )
    statement1;
    else
        statement2;
```

The if..else Statement

- The `if..else` statement in Java has the following syntax

If the *condition* evaluates to **true** statement 1 is executed.

```
if ( condition )
    statement1;
    else
        statement2;
```

The if..else Statement

- The `if..else` statement in Java has the following syntax

If the `condition` evaluates to `true` statement 1 is executed.

```
if ( condition )
    statement1;
    else
        statement2;
```

If the `condition` evaluates to `false`, then `statement2` is executed. One or the other is executed but not both!

The if..else Statement

- An example of an `if..else` statement in Java

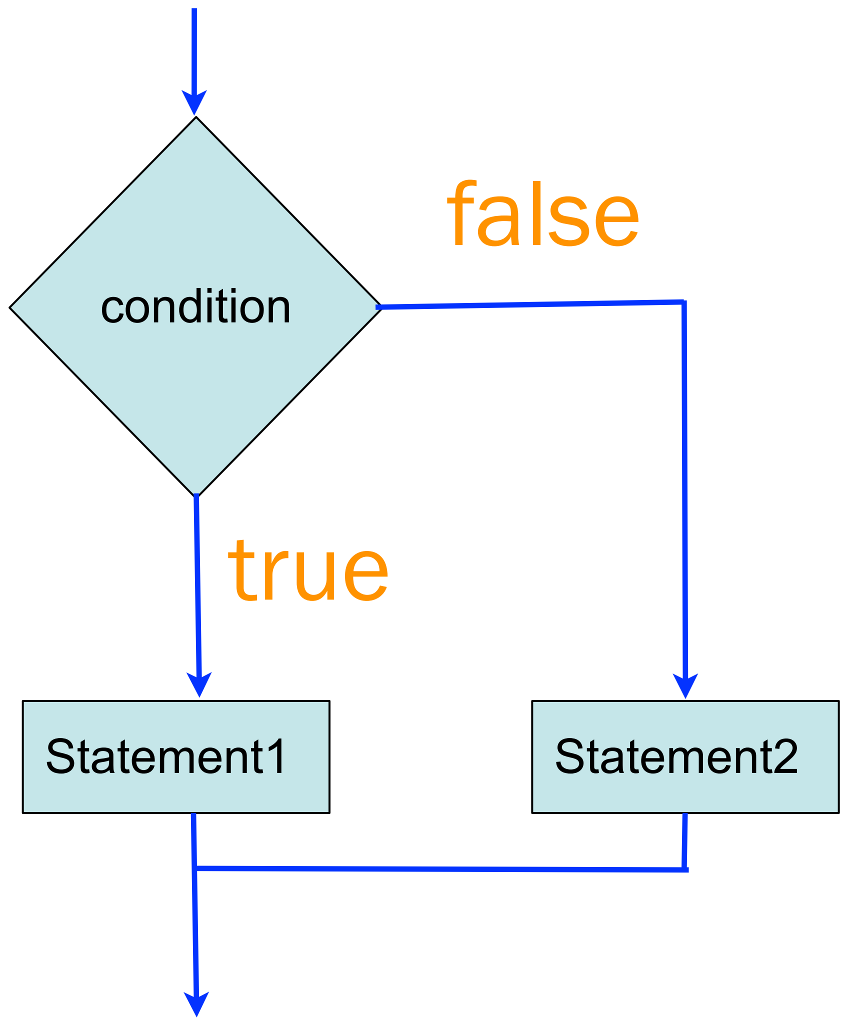
```
class Thermometer {
   public static void main(String args[]) {
      double currentTemp = 20.0;

      System.out.println("Current temperature is "+ currentTemp);
      if (currentTemp > 30.0)
         System.out.println(" It is too hot");
         else
           System.out.println(" It is warm or cold");

      }
}
```

- QUIZ: What is the output of the above program?

Logic of an if..else Statement



Block Statements

- Several statements can be grouped together into a block statement in either the true or false branch using braces {..}.

For example:

```
class Thermometer {
   public static void main(String args[]) {
      double currentTemp = 20.0;

      System.out.println("Current temperature is "+ currentTemp);
      if (currentTemp > 30.0){
         System.out.println(" It is too hot");
         System.out.println(" End of weather report");
       }
         else {
           System.out.println(" It is warm or cold");
           System.out.println(" End of weather report");
         }

      }
}
```

Nested if Statements

- The true or false branch of an `if` statement can be another `if` . For example:

```
class Thermometer {
   public static void main(String args[]) {
      double currentTemp = 20.0;

      System.out.println("Current temperature is "+ currentTemp);
      if (currentTemp > 30.0)
         System.out.println(" It is too hot");
         else if (currentTemp > 18.0)
           System.out.println(" It is warm");
           else
             System.out.println(" It is too cold");
      }
}
```

The switch Statement

- The **switch** statement provides another means to decide which statement to execute.
- The **switch** statement in Java has the following syntax.

```
switch ( expression )
{
    case value1:
        statement1;
    case value2:
        statement2;
    case value3:
        statement3;
    case  ...
}
```

- The **switch** statement evaluates the expression and then attempts to match the result to one of several possible cases.

The switch Statement

- The `switch` statement provides another means to decide which statement to execute.
- The `switch` statement in Java has the following syntax.

`switch` and `case` are reserved words

```
switch ( expression )
{
   case value1:
      statement1;
   case value2:
      statement2;
   case value3:
      statement3;
   case  ...
}
```

- The `switch` statement evaluates the expression and then attempts to match the result to one of several possible cases.

The switch Statement

- The `switch` statement provides another means to decide which statement to execute.
- The `switch` statement in Java has the following syntax.

```
switch ( expression )
{
   case value1:
       statement1;
   case value2:
       statement2;
   case value3:
       statement3;
   case  ...
}
```

`switch` and `case` are reserved words

If ***expression*** matches ***value2***, ***statement2*** is executed

- The `switch` statement evaluates the expression and then attempts to match the result to one of several possible cases.

The switch Statement

- The **switch** statement provides another means to decide which statement to execute.
- The **switch** statement in Java has the following syntax.

switch and **case** are reserved words

```
switch ( expression )
{
    case value1:
        statement1;
    case value2:
        statement2;
    case value3:
        statement3;
    case  ...
}
```

expression must evaluate to an `int` or `char`

If ***expression*** matches ***value2***, ***statement2*** is executed

- The **switch** statement evaluates the expression and then attempts to match the result to one of several possible cases.

The switch Statement

- An example of `switch` statement in Java:

```
class WeekDays {
   public static void main(String args[]) {
      int day = 4;
      String weekdayString;
      switch (day){
          case 1: weekdayString = "Monday";
                  break;
          case 2: weekdayString = "Tuesday";
                  break;
          case 3: weekdayString = "Wednesday";
                  break;
          case 4: weekdayString = "Thursday";
                  break;
          case 5: weekdayString = "Friday";
                  break;
          case 6: weekdayString = "Saturday";
                  break;
          case 7: weekdayString = "Sunday";
                  break;
       }
     System.out.println(" The day  of the week is " + weekdayString);
     }
   }
```

The switch Statement

- An example of `switch` statement in Java:

```
class WeekDays {
   public static void main(String args[]) {
      int day = 4;
      String weekdayString;
      switch (day){
          case 1: weekdayString = "Monday";
                  break;
          case 2: weekdayString = "Tuesday";
                  break;
          case 3: weekdayString = "Wednesday";
                  break;
          case 4: weekdayString = "Thursday";
                  break;
          case 5: weekdayString = "Friday";
                  break;
          case 6: weekdayString = "Saturday";
                  break;
          case 7: weekdayString = "Sunday";
                  break;
       }
     System.out.println(" The day  of the week is " + weekdayString);
     }
   }
```

A break statement causes control to transfer to the end of the switch statement.

The switch Statement

- An example of `switch` statement in Java:

```
class WeekDays {
   public static void main(String args[]) {
      int day = 2;
      String weekdayString;
      switch (day){
         case 1: weekdayString = "Monday";
                 break;
         case 2: weekdayString = "Tuesday";
         case 3: weekdayString = "Wednesday";
                 break;
         case 4: weekdayString = "Thursday";
                 break;
         case 5: weekdayString = "Friday";
                 break;
         case 6: weekdayString = "Saturday";
                 break;
         case 7: weekdayString = "Sunday";
                 break;
       }
     System.out.println(" The day  of the week is " + weekdayString);
     }
   }
```

If there is no break statement, the execution continues to the next case!

- QUIZ: What is the output of the above program?

The switch Statement

- A **switch** statement can have an optional default case. If the default case is present, control jumps to the default if no case matches.

```
class WeekDays {
   public static void main(String args[]) {
      int day = 9;
      String weekdayString;
      switch (day){
         case 1: weekdayString = "Monday";
                 break;
         case 2: weekdayString = "Tuesday";
                 break;
         case 3: weekdayString = "Wednesday";
                 break;
         case 4: weekdayString = "Thursday";
                 break;
         case 5: weekdayString = "Friday";
                 break;
         case 6: weekdayString = "Saturday";
                 break;
         case 7: weekdayString = "Sunday";
                 break;
         default: weekdayString = "Unknown weekday";
         }
      System.out.println(" The day  of the week is " + weekdayString);
      }
   }
```

- QUIZ: What is the output of the above program?

Exercise

- Re-write the following `switch` statement using an `if..else` statement

```
class WeekDays {
   public static void main(String args[]) {
      int day = 4;
      String weekdayString;
      switch (day){
         case 1: weekdayString = "Monday";
                 break;
         case 2: weekdayString = "Tuesday";
                 break;
         case 3: weekdayString = "Wednesday";
                 break;
         case 4: weekdayString = "Thursday";
                 break;
         case 5: weekdayString = "Friday";
                 break;
         case 6: weekdayString = "Saturday";
                 break;
         case 7: weekdayString = "Sunday";
                 break;
        }
      System.out.println(" The day  of the week is " + weekdayString);
    }
  }
```

Control Flow Statements

- Conditional statements: a conditional statement lets us choose which statement will be executed next.

  Java’s conditional statements are:
  - The `if` statement
  - The `if..else` statement
  - The `switch` statement

- Iteration/looping statements: a looping statement lets us execute a statement several times.

  Java’s looping statements are:
  - The `while` loop
  - The `do..while` loop
  - The `for` loop

Control Flow Statements

- Conditional statements: a conditional statement lets us choose which statement will be executed next.

  Java's conditional statements are:
  - The `if` statement
  - The `if..else` statement
  - The `switch` statement

- Iteration/looping statements: a looping statement lets us execute a statement several times.

  Java's looping statements are:
  - The `while` loop
  - The `do..while` loop
  - The `for` loop

The while Loop

- The `while` loop in Java has the following syntax

`while` is a Java reserved word

The *`condition`* must be a boolean expression. i.e., It must evaluate to either `true` or `false`.

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

If the *`condition`* is `true`, the *`statement`* is executed. Then the *`condition`* is evaluated again.

The *`statement`* is executed repeatedly until the *`condition`* evaluates to `false`.

The while Loop

- An example of a `while` loop in Java

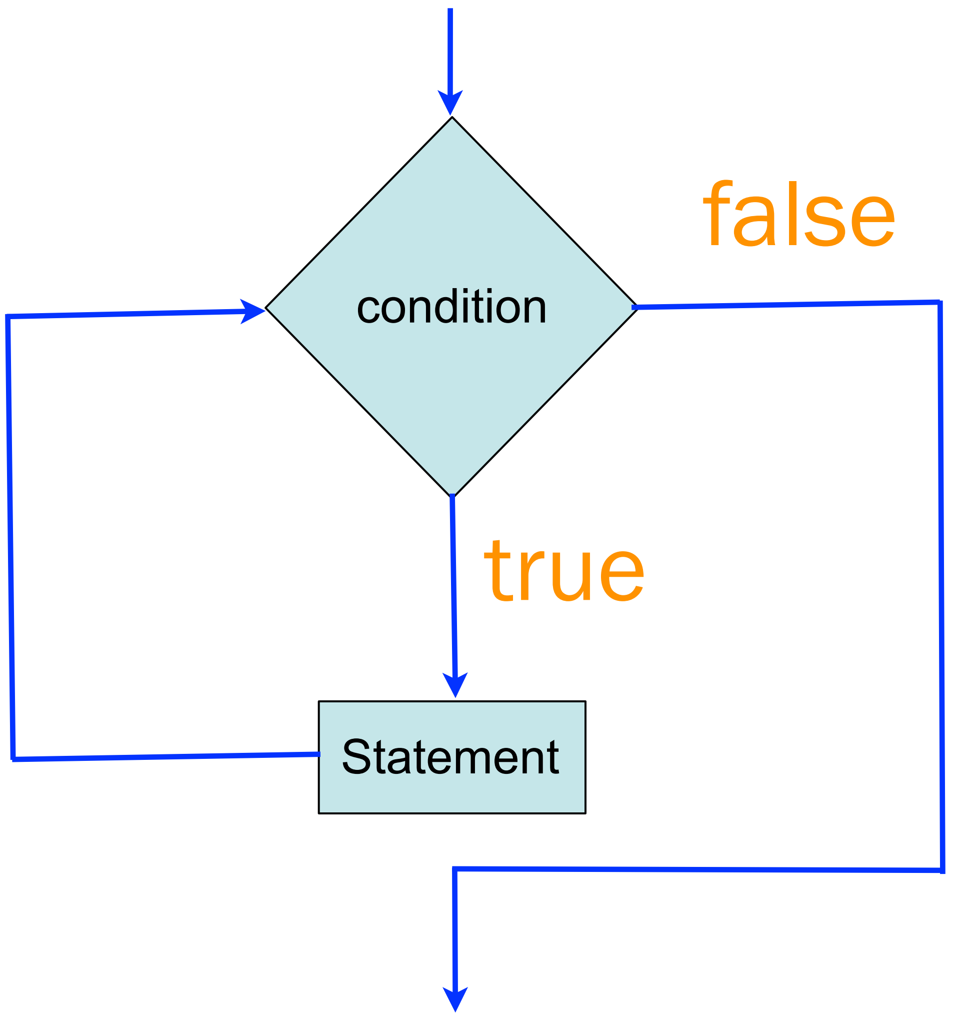
```
class Counter {
   public static void main(String args[]) {
      int count = 0;
      int max = 10;

      while (count <= max){
         System.out.println(" The count is "+count);
         count++;
      }
   }
}
```

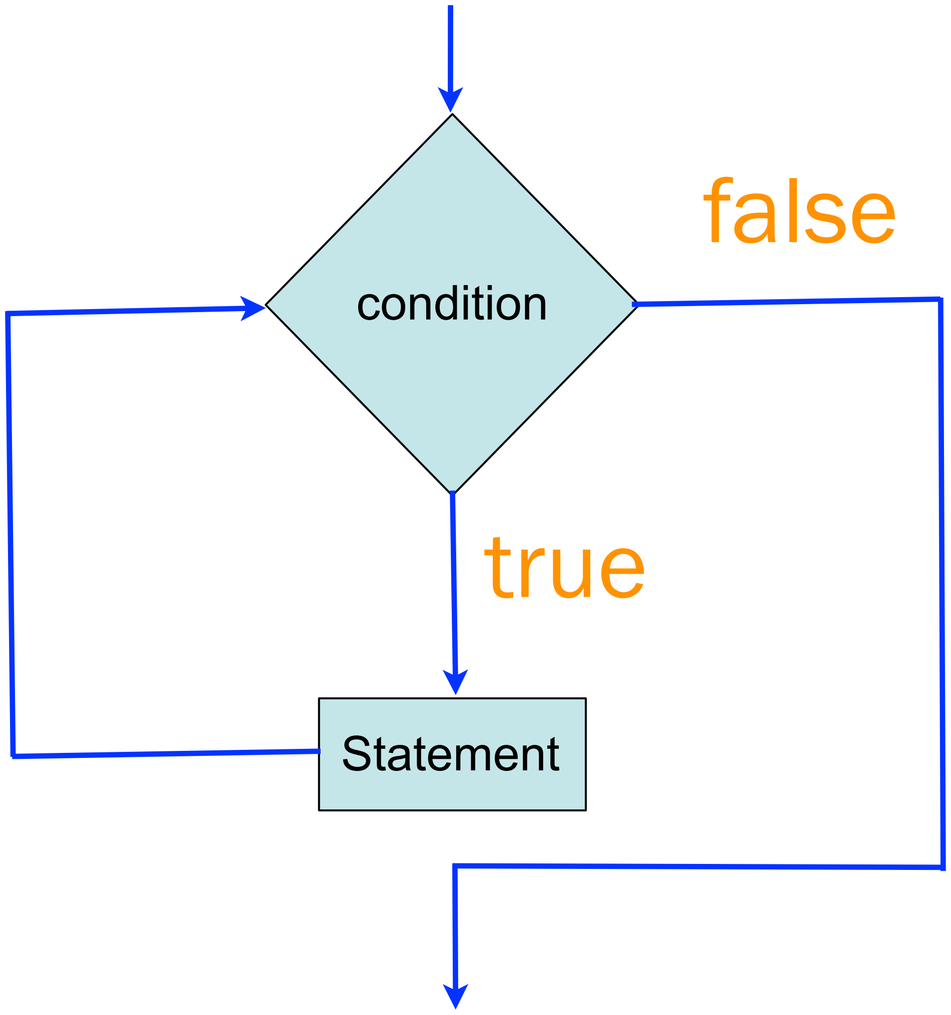
First, the condition is evaluated. The value of **`count`** is either less than or equal to 10, or it is not.

If the condition is true, the **`System.out.println(" The count is "+count)`** statement is executed. If it is not, the statement is skipped.

Logic of a while Loop



Logic of a while Loop



- If the condition is false initially then the statement (s) will never be executed!
- Hence the body of a while loop may execute zero or more times

The while Loop

QUIZ (a): What is the output of the following program?

```
class Counter {
   public static void main(String args[]) {
      int count = 0;
      int max = 10;

      while (count <= max){
         count++;
         System.out.println(" The count is "+count);
      }
   }
}
```

The while Loop

QUIZ (b): What is the output of the following program?

```
class Counter {
   public static void main(String args[]) {
      int count = 0;
      int max = 10;

      while (count < max){
         count++;
         System.out.println(" The count is "+count);
      }
   }
}
```

The do..while Loop

- The `do..while` loop in Java has the following syntax

`do` and `while` `are` Java reserved words

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

- The ***statement*** is executed once initially, and then the ***condition*** is evaluated.

- Then the ***statement*** is executed repeatedly until the ***condition*** evaluates to `false`.

The do .. while Loop

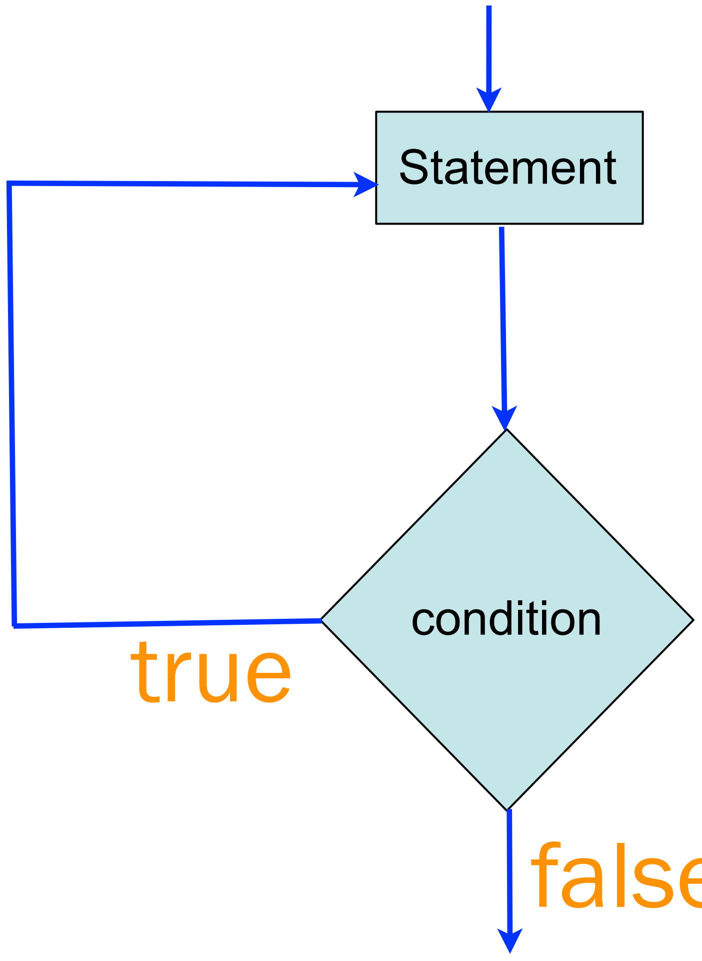
- An example of a do .. `while` loop in Java

```
class Counter {
   public static void main(String args[]) {
      int count = 10;
      do {
         System.out.println(" The count is "+count);
         count--;
      } while (count > 0);
   }
}
```

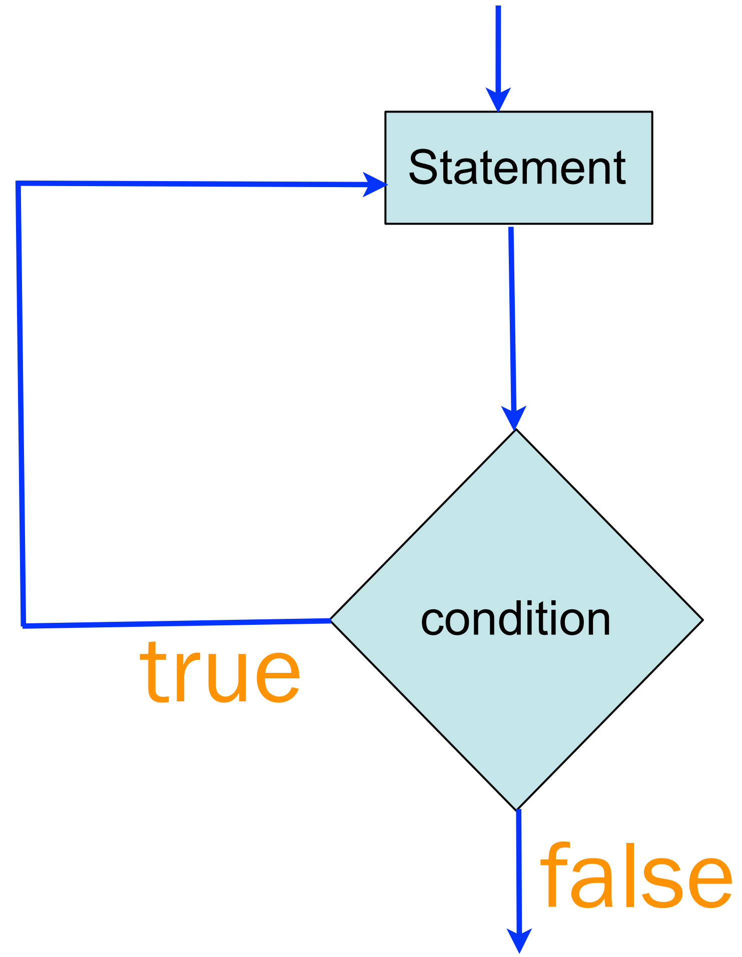
First, the statements are executed. Then the condition is evaluated.

If the condition is true, the statements are executed again. If it is not, the statements are skipped.

Logic of a do .. while Loop

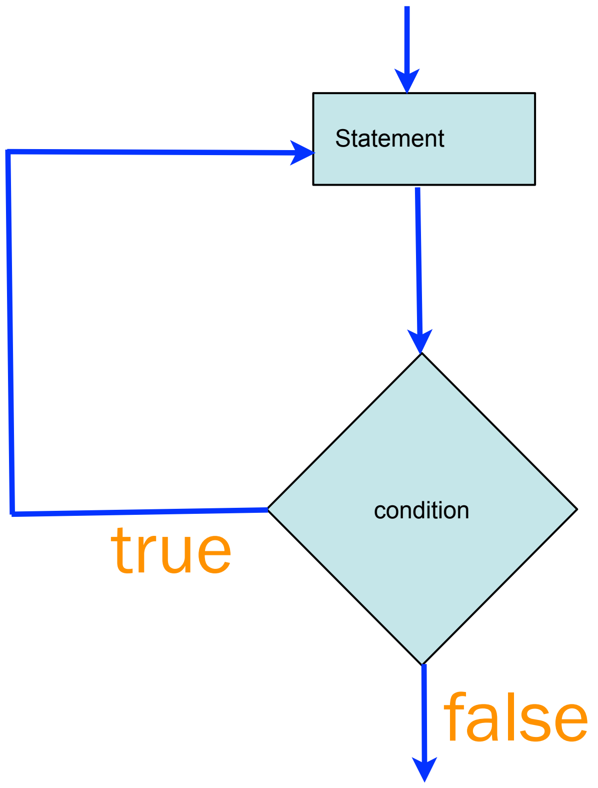


Logic of a do .. while Loop

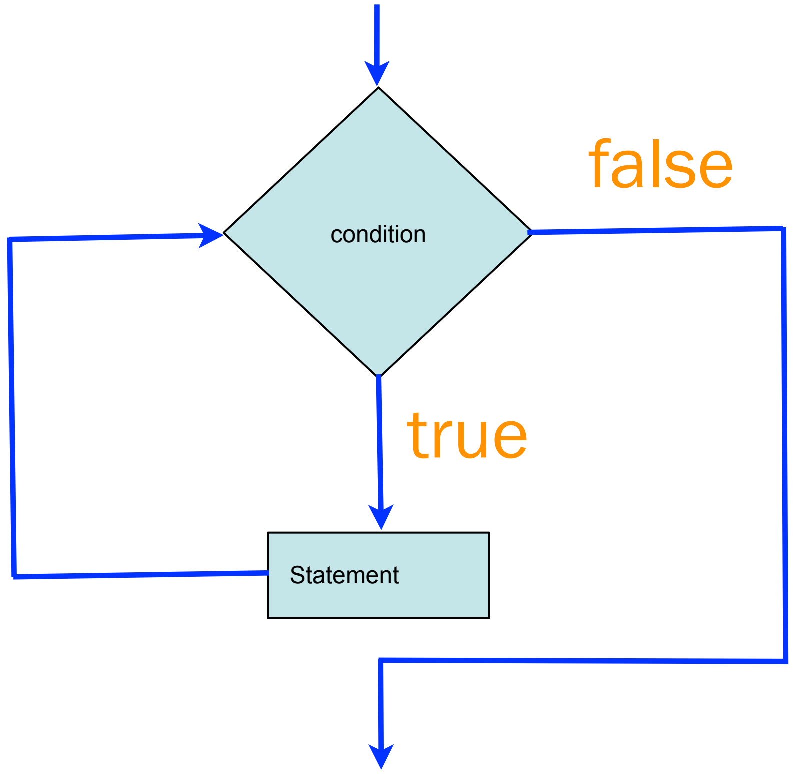


- Similar to a while loop **except that the condition is evaluated after the body has been executed.**
- Hence the body of a do .. while loop is guaranteed to execute at least once.

A do .. while Loop *vs* a while Loop



do .. while loop



while loop

The for Loop

- The `for` loop in Java has the following syntax

`for` **`is a Java`** reserved word

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

The ***`condition-statement-increment`*** cycle is executed repeatedly

The for Loop

- Like a `while` loop, the condition of a `for` statement is tested prior to executing the loop body
- Therefore, the body of a `for` loop will execute zero or more times
- It is well suited for executing a loop a specific number of times that can be determined in advance

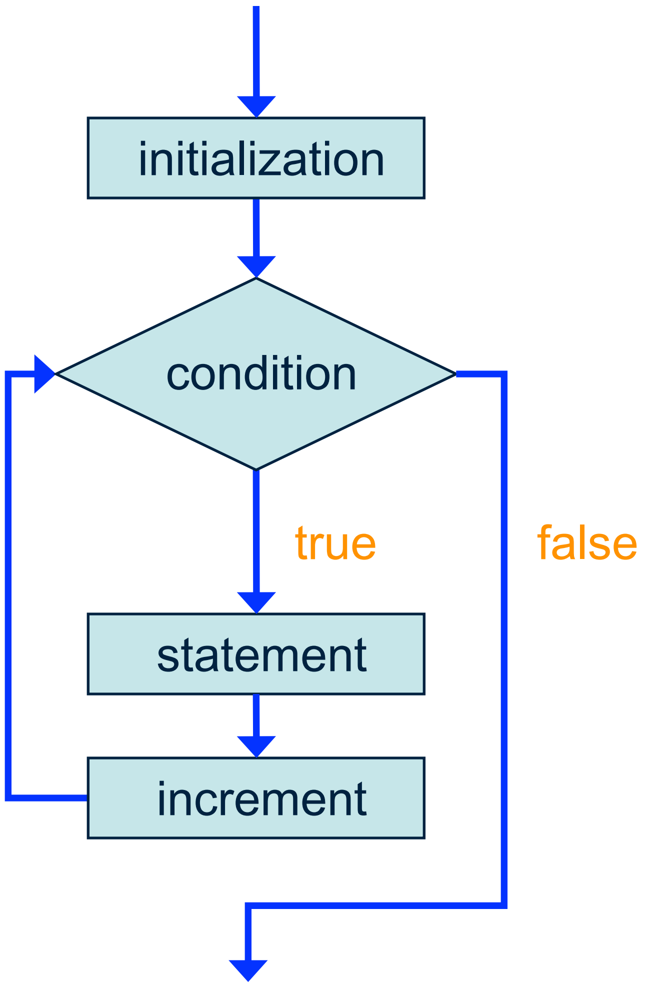
The for Loop

- An example of a `for` loop in Java

```
class CounterFor {
   public static void main(String args[]) {
      int max = 10;

      for (int count =1; count <= max; count++){
         System.out.println(" The count is "+count);
      }
   }
}
```

Logic of a for Loop



The for Loop

- 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
  - If the ***`increment`*** is left out, no increment operation is performed
- Both semi-colons are always required in the **`for`** loop header

The for Loop

- More examples of a **for** loop in Java

```
 class CounterFor {
    public static void main(String args[]) {
       int max = 10;
       int count = 1;
       for (; count <= max; count++){
          System.out.println(" The count is "+count);
       }
    }
 }
```

Each expression in the header is optional!

```
class CounterFor {
   public static void main(String args[]) {
      int max = 10;
      int count = 1;
      for (;; count++){
         System.out.println(" The count is "+count);
      }
   }
}
```

Infinite!

Choosing a Loop Statement to Use

- When you can’t determine how many times you want to execute the loop body, use a `while` statement or a `do..while` statement
  - If it might be zero or more times, use a `while` statement
  - If it will be at least once, use a `do..while` statement
- If you can determine how many times you want to execute the loop body, use a `for` statement

Infinite Loops

- The body of a loop should eventually make the condition to evaluate to false. Otherwise, it is an infinite loop, which will execute until the user interrupts the program execution.

```
class Thermostat {
   public static void main(String args[]) {
      int outsideTemp = 17;
      int currentTemp = 18;

      while (currentTemp > outsideTemp){
         currentTemp++;
         System.out.println("Increasing temperature to "+currentTemp);
      }
   }
}
```

Infinite Loops

- The body of a loop should eventually make the condition to evaluate to false. Otherwise, it is an infinite loop, which will execute until the user interrupts the program execution.

```
class Thermostat {
   public static void main(String args[]) {
      int outsideTemp = 17;
      int currentTemp = 18;

      while (currentTemp > outsideTemp){
         currentTemp++;
         System.out.println("Increasing temperature to "+currentTemp);
      }
   }
}
```

Exercise

- Read about **logical operators** and how to use them to construct complex Boolean expressions.

- Read about the **break** and **continue** statements in Java and how to use them in loop structures.

- What is another way of executing statements repeatedly other than looping statements?
  Hint: Read about *recursion* and *iteration* in Java