CS620
Structured Programming
Introduction to Java

Day 2 - Lecture 2

Conditionals - Switch, Loops

Conditional Control Structures

- If
- Then
- Else
- Switch

Conditional Control Structures

- Also known as ‘program flow-control’
  - Think of a pipe with flowing water and valves to direct the flow in one direction or another
- Provides a huge increase in the potential for complex programs
- Crucial for **decision-making** in software
- Alter the **sequence of execution**

Sequence of Execution

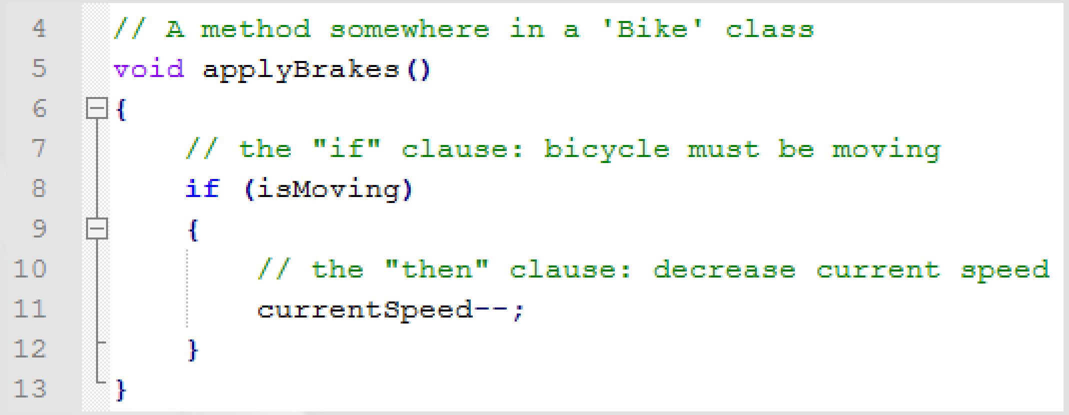
- When a program typically runs, it executes instructions (lines of code) in sequence
- Control structures can change the order of execution
- Control structures can enable/disable sections of code

If-Then

- The most basic control structure is **if-then**:
    - `if (someBoolean)`
    - `{`
        - `// Do stuff`
    - `}`
  - Tells your program to execute a certain section of code *only if* a particular test evaluates to true.
  - The opening brace is the equivalent of ‘then’ when saying ***if x, then do y***

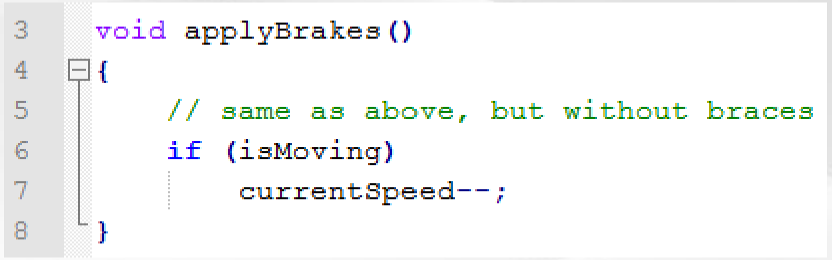
If-Then

- If *isMoving* is false, the execution jumps to the end of the 'if' block.



```
// A method somewhere in a 'Bike' class
void applyBrakes()
{
    // the "if" clause: bicycle must be moving
    if (isMoving)
    {
        // the "then" clause: decrease current speed
        currentSpeed--;
    }
}
```

- Braces can be left out **<u>if</u>** there's only one statement inside the block:



```
void applyBrakes()
{
    // same as above, but without braces
    if (isMoving)
        currentSpeed--;
}
```

- It's good practice to use braces anyway

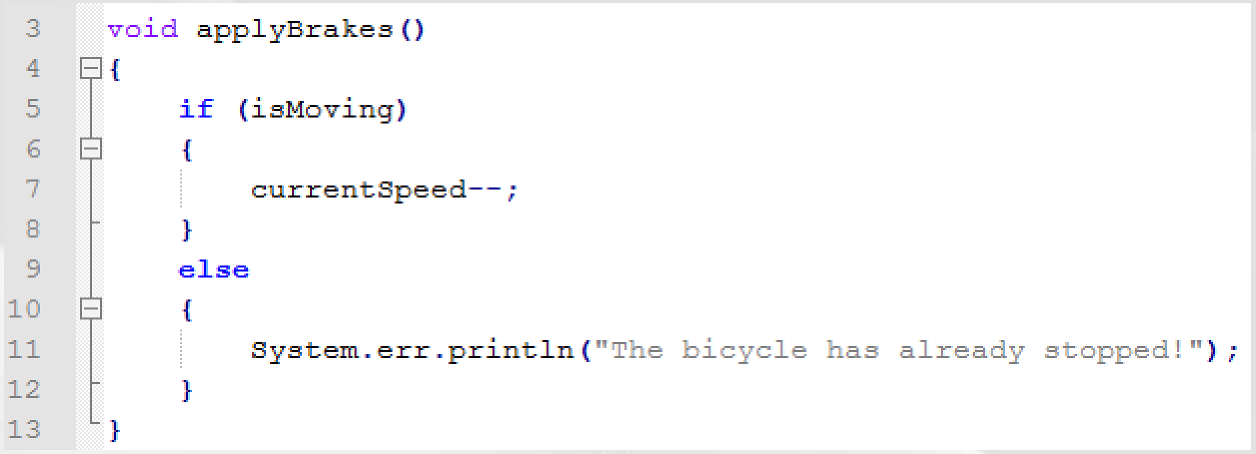
If-Then-Else

- The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false.

- You could use an if-then-else statement in the applyBrakes method to take some action if the brakes are applied when the bicycle is **not** in motion.

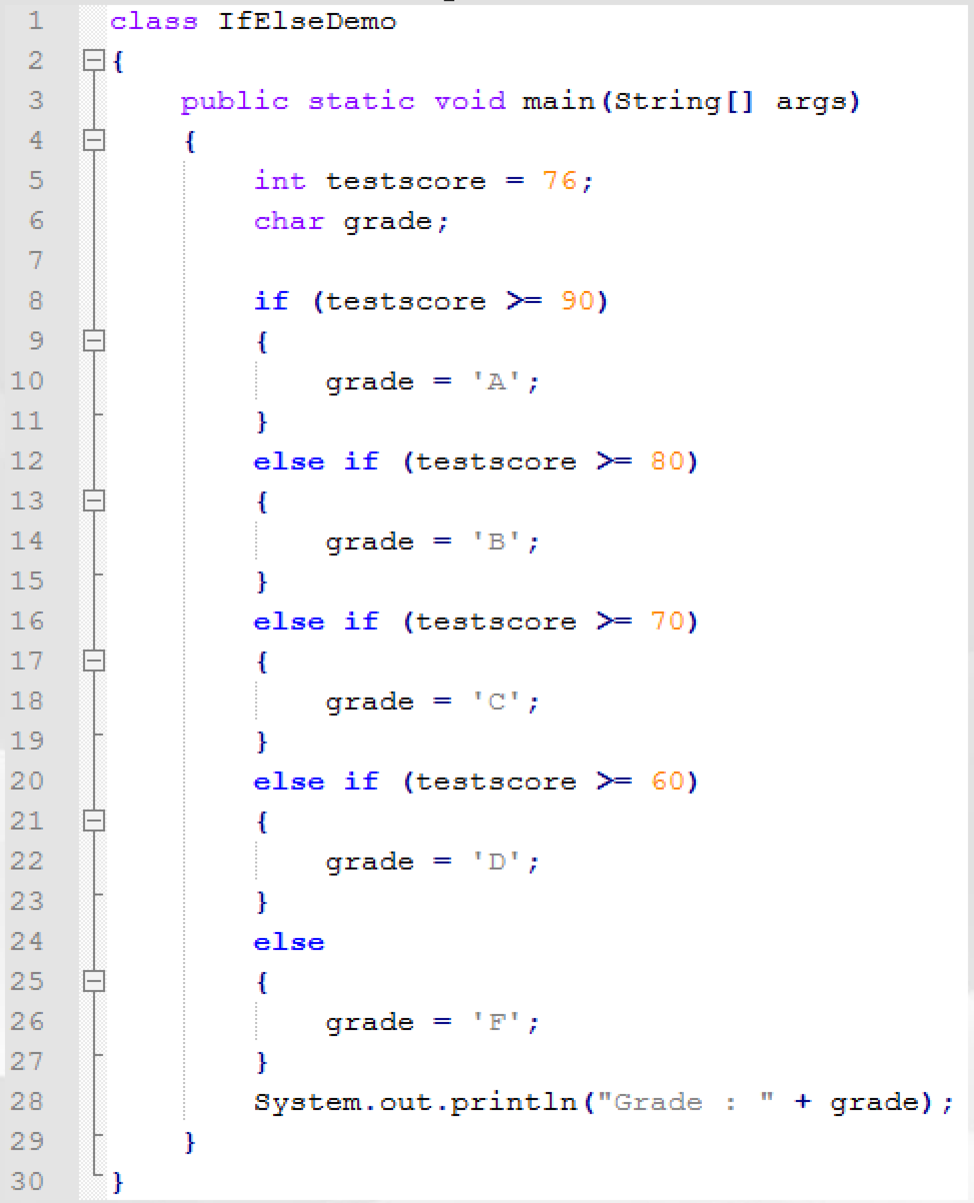
If-Then-Else

- In this case, the action is to simply print an error message stating that the bicycle has already stopped.



```java
void applyBrakes()
{
    if (isMoving)
    {
        currentSpeed--;
    }
    else
    {
        System.err.println("The bicycle has already stopped!");
    }
}
```

A more complex example:



```
class IfElseDemo
{
    public static void main(String[] args)
    {
        int testscore = 76;
        char grade;

        if (testscore >= 90)
        {
            grade = 'A';
        }
        else if (testscore >= 80)
        {
            grade = 'B';
        }
        else if (testscore >= 70)
        {
            grade = 'C';
        }
        else if (testscore >= 60)
        {
            grade = 'D';
        }
        else
        {
            grade = 'F';
        }
        System.out.println("Grade : " + grade);
    }
}
```

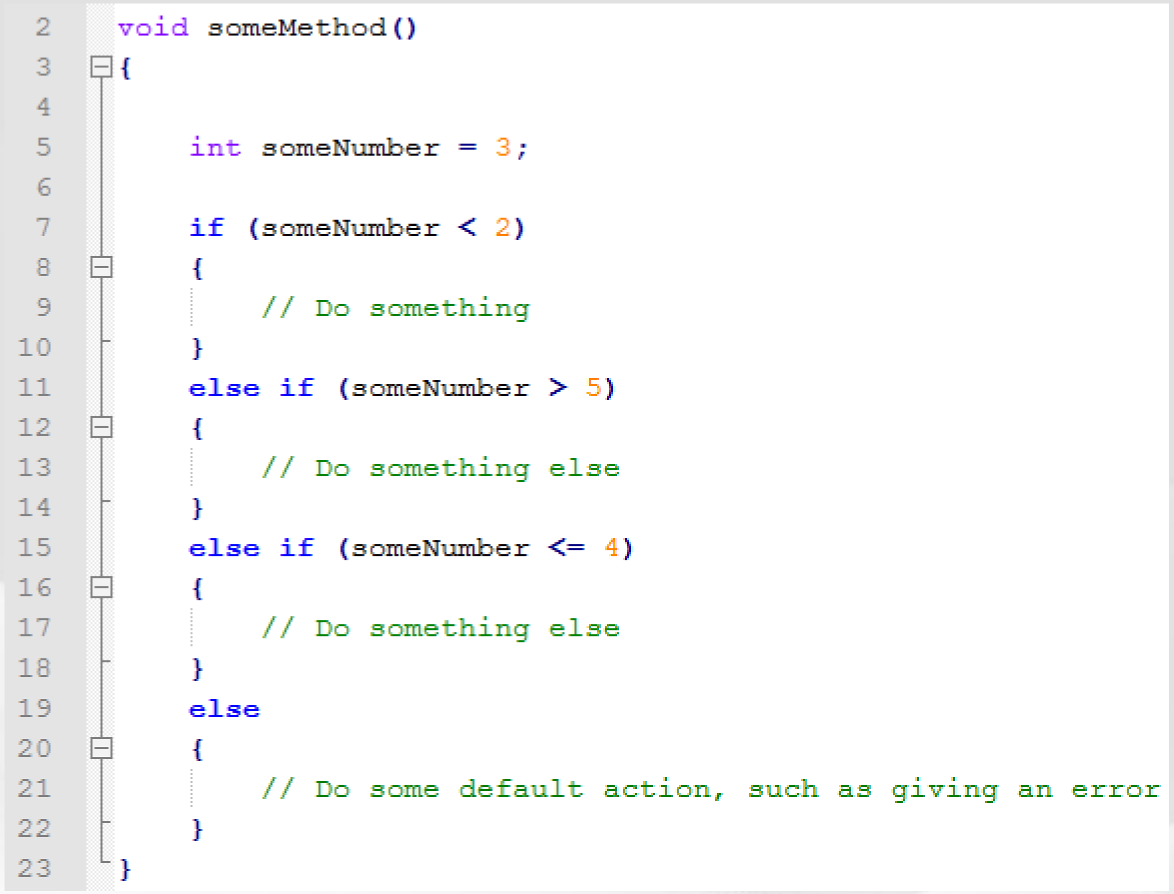
A more complex example:

- The output from the program is:
  - `Grade = C`

- You may have noticed that the value of testscore can satisfy more than one expression in the compound statement: 76 >= 70 and 76 >= 60.

- However, once a condition is satisfied, the appropriate statements are executed (grade = 'C';) and the remaining conditions are not evaluated.

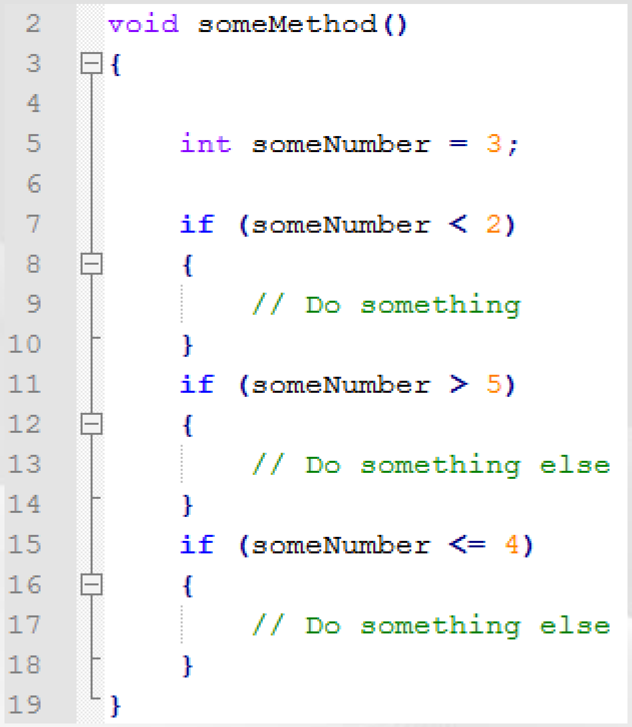
Another Example



```
void someMethod()
{

    int someNumber = 3;

    if (someNumber < 2)
    {
        // Do something
    }
    else if (someNumber > 5)
    {
        // Do something else
    }
    else if (someNumber <= 4)
    {
        // Do something else
    }
    else
    {
        // Do some default action, such as giving an error
    }
}
```

Another Example

- What happens if we take out the ‘else’ keywords?



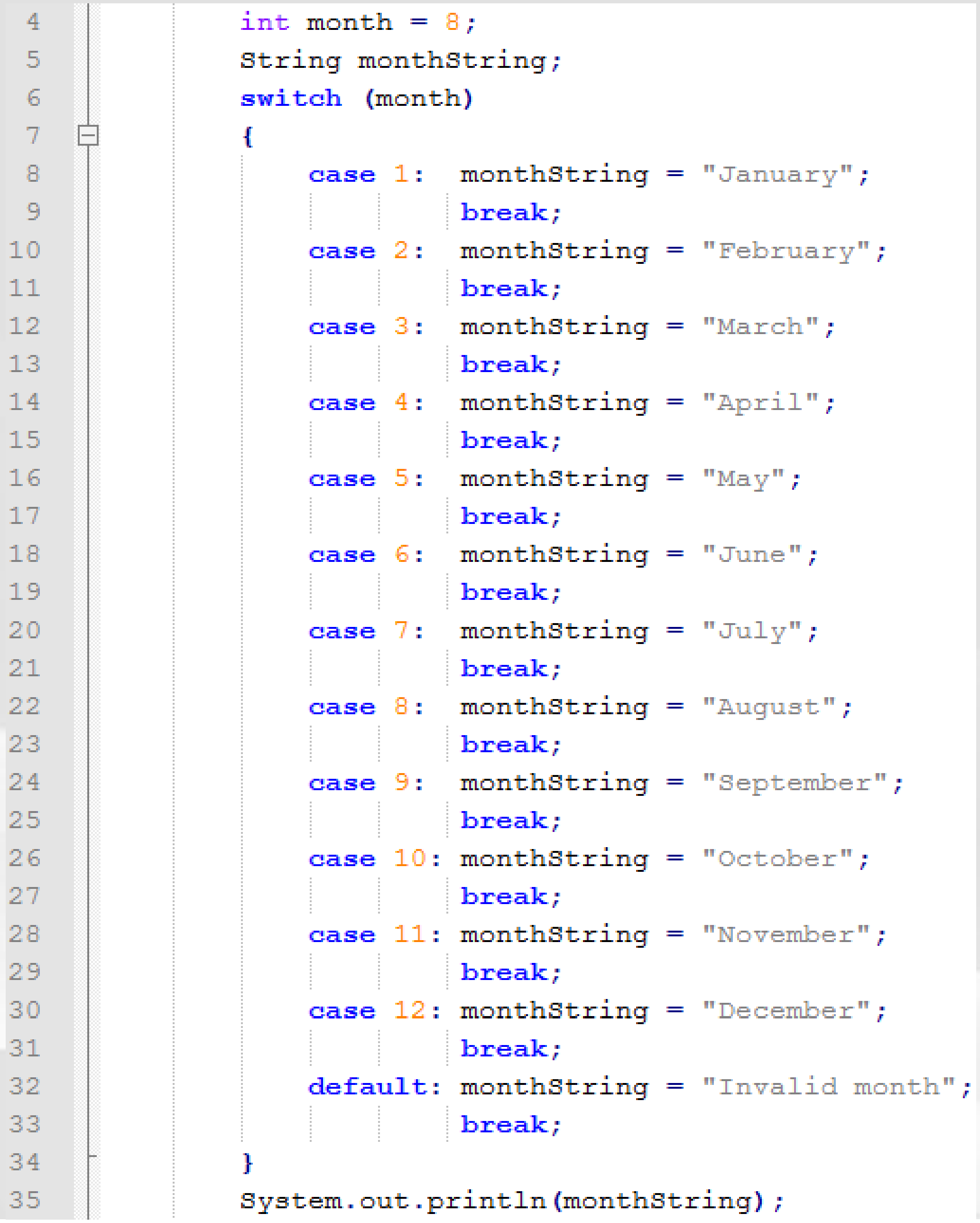
```
void someMethod()
{

    int someNumber = 3;

    if (someNumber < 2)
    {
        // Do something
    }
    if (someNumber > 5)
    {
        // Do something else
    }
    if (someNumber <= 4)
    {
        // Do something else
    }
}
```

Switch

- Switch allows multiple execution paths depending on a single variable.

- With switch, you set up a variable to determine which path to take; then define a ‘case’ for each path you want.

Switch - Example



```
int month = 8;
String monthString;
switch (month)
{
    case 1:  monthString = "January";
             break;
    case 2:  monthString = "February";
             break;
    case 3:  monthString = "March";
             break;
    case 4:  monthString = "April";
             break;
    case 5:  monthString = "May";
             break;
    case 6:  monthString = "June";
             break;
    case 7:  monthString = "July";
             break;
    case 8:  monthString = "August";
             break;
    case 9:  monthString = "September";
             break;
    case 10: monthString = "October";
             break;
    case 11: monthString = "November";
             break;
    case 12: monthString = "December";
             break;
    default: monthString = "Invalid month";
             break;
}
System.out.println(monthString);
```

Switch

- In this case, August is printed to standard output.
- The body of a switch statement is known as a *switch block*.
- A statement in the switch block can be labelled with one or more case or *default* labels.
- The switch statement evaluates its expression, then executes all statements that follow the matching case label.

Switch

- You could also display the name of the month with if-then-else statements:

```
int month = 8;
if (month == 1)
{
    System.out.println("January");
}
else if (month == 2)
{
    System.out.println("February");
} ... // and so on
```

- Deciding whether to use if-then-else statements or a switch statement is based on readability and the expression that the statement is testing.

- An if-then-else statement can test expressions based on ranges of values or conditions, whereas a switch statement tests expressions based only on a single integer, enumerated value, or String object.

Switch - Break Statement

- Another point of interest is the break statement.
- Each break statement terminates the enclosing switch statement.

- Control flow continues with the first statement following the switch block.

- The break statements are necessary because without them, statements in switch blocks *fall through*:
  - All statements after the matching case label are executed in sequence, regardless of the expression of subsequent case labels, until a break statement is encountered.

Loops

- A very useful conditional control structure, Loops are used to do something over and over in a program.
- You will often find a use for a piece of code that runs repetitively until a certain condition is met, or to count through a list, etc.

Loops

- Computers in general run in a constant loop, cycling over and over.
- When your computer is idle on the desktop it appears to be doing nothing, but is in fact running many continuous loops and constantly updating millions of variables and running functions accordingly.

While

- The while statement continually executes a block of statements while a particular condition is true.

- Its syntax can be expressed as:
  - `while (expression) { statement(s) }`

- The **while** statement evaluates *expression*, which must return a boolean value.

- If the expression evaluates to true, the **while** statement executes the *statement*(s) in the while block.

- The **while** statement continues testing the expression and executing its block until the expression evaluates to false.

While - Infinite Loop

- The following code will cause an *infinite loop*; otherwise known as a **crash.**

  - `while(true) { doSomething(); }`

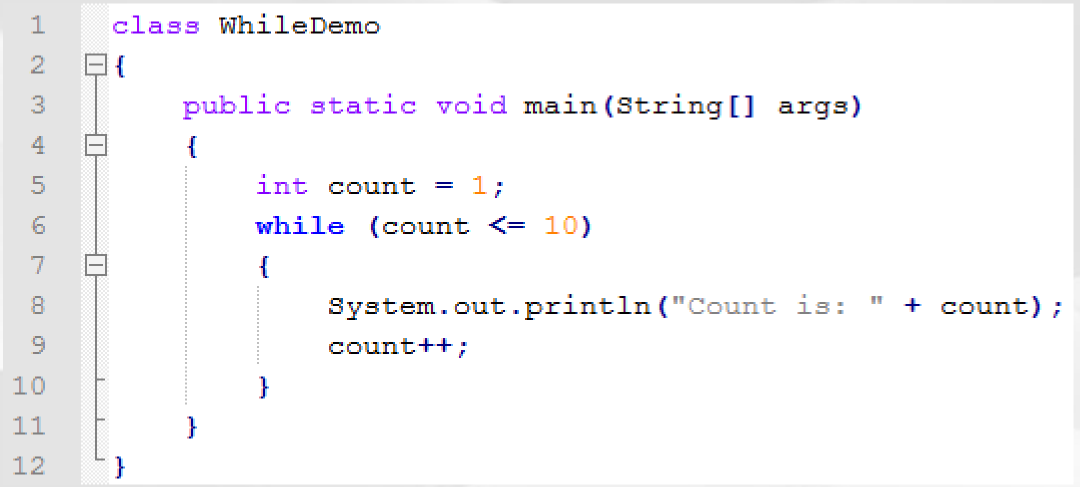
- The expression ‘true’ will never evaluate as ‘false’ under any circumstances, so the loop never stops.

- Because a program executes its instructions in sequence and only ever does one thing at a time, it gets *‘stuck’* inside the infinite loop and appears to freeze up!

While - Example

- The following program will print 10 messages to the screen by executing the same statements 10 times over

- The repeated statements are the ones inside the **while** loop's block.



```
class WhileDemo
{
    public static void main(String[] args)
    {
        int count = 1;
        while (count <= 10)
        {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
```

Do-While

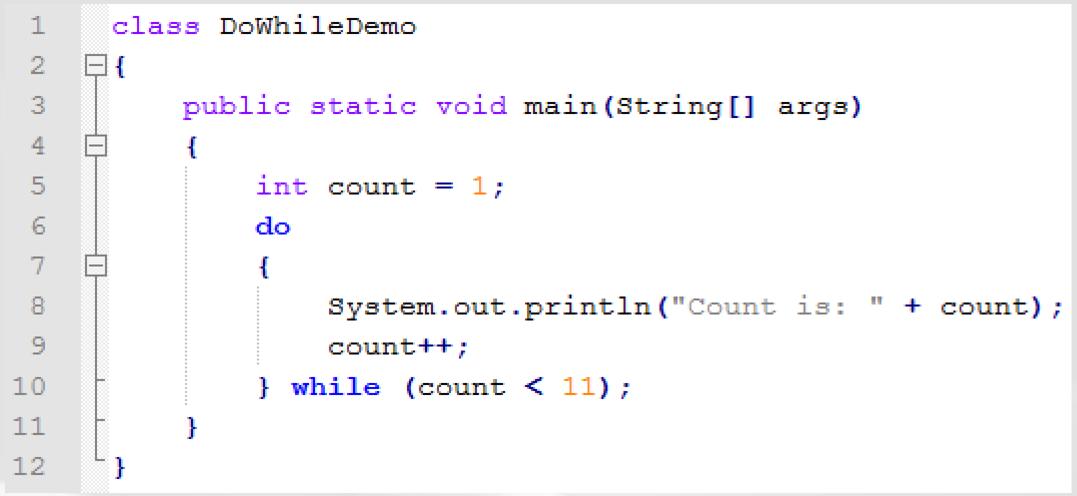
- Java programming also provides a **do-while** statement, which can be expressed as follows:

  - `do { statement(s) } while (expression);`

- The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top.

- Therefore, the statements within the do block are always executed at least once.

Do-While



```
class DoWhileDemo
{
    public static void main(String[] args)
    {
        int count = 1;
        do
        {
            System.out.println("Count is: " + count);
            count++;
        } while (count < 11);
    }
}
```

For

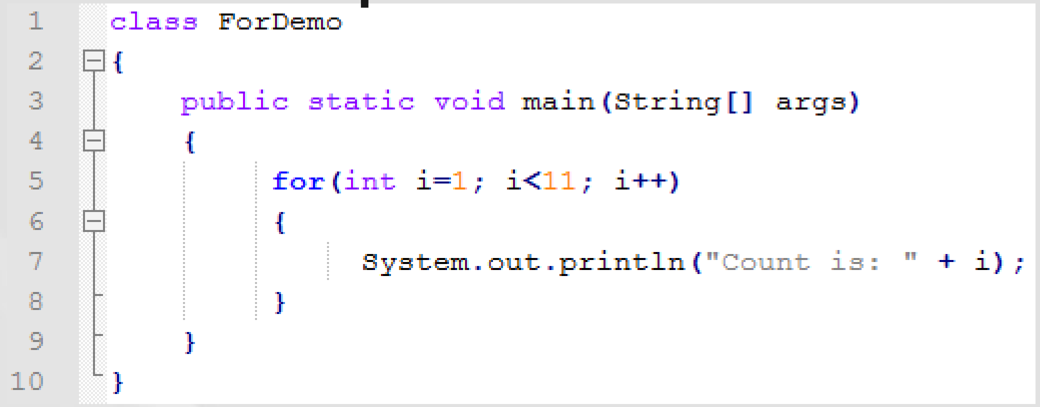
- **For** provides a compact way to iterate over a range of values.

- Programmers often refer to it as the *"for loop"* because of the way in which it repeatedly loops until a particular condition is satisfied.

- It can do the same things as a while loop, but in a more compact manner.

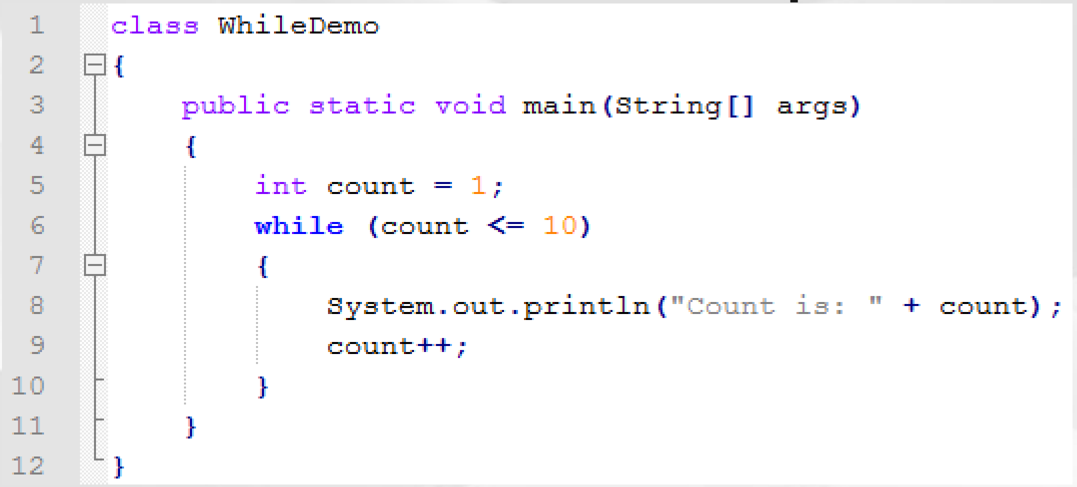
For

- A typical **for** loop:



```
class ForDemo
{
    public static void main(String[] args)
    {
        for(int i=1; i<11; i++)
        {
            System.out.println("Count is: " + i);
        }
    }
}
```

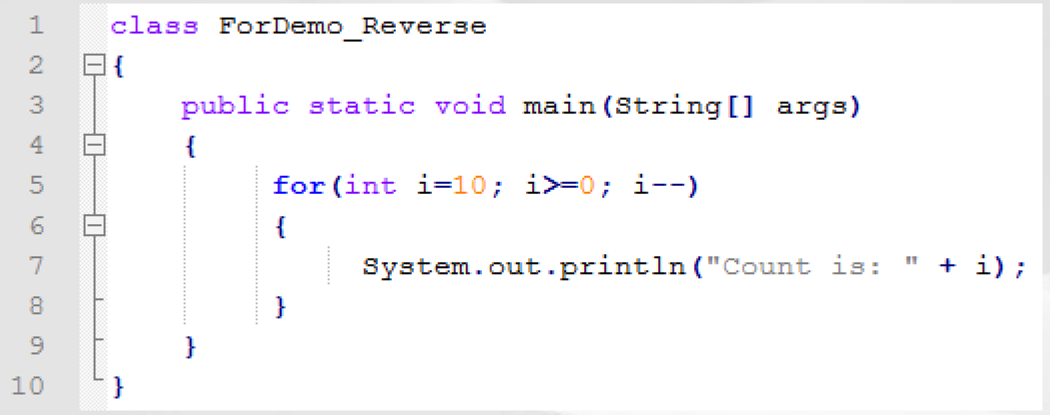
- The above example does exactly the same as our first **while** loop:



```
class WhileDemo
{
    public static void main(String[] args)
    {
        int count = 1;
        while (count <= 10)
        {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
```

Loops in Reverse

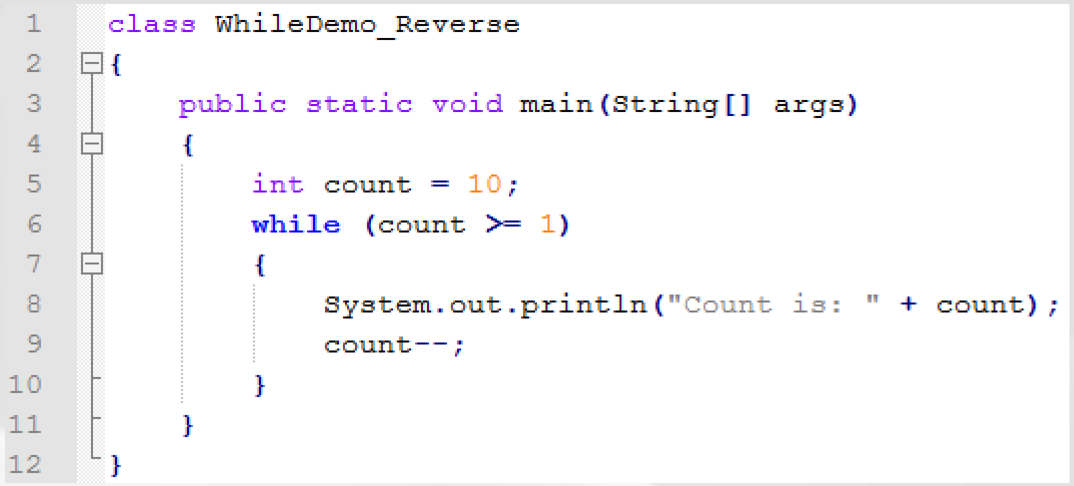
- Loops can work ‘backwards’ too
- In fact, you simply need to set them up so that they check a condition that will eventually reach ‘false’ for the loop to not be infinite.
- ‘Reverse’ **for** loop:



```java
class ForDemo_Reverse
{
    public static void main(String[] args)
    {
        for(int i=10; i>=0; i--)
        {
            System.out.println("Count is: " + i);
        }
    }
}
```

Loops in Reverse

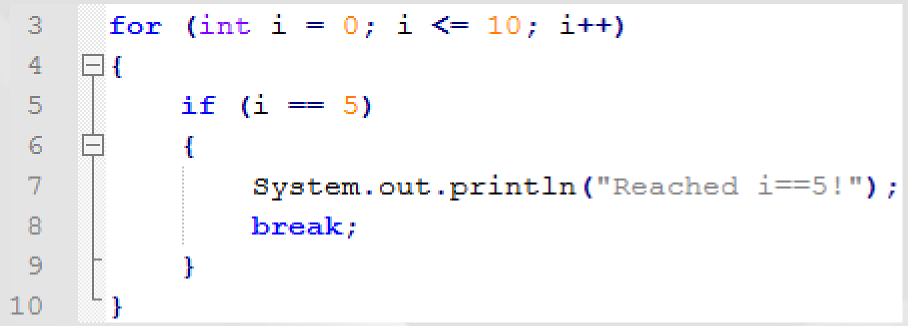
- ‘Reverse’ **while** loop:



```
class WhileDemo_Reverse
{
    public static void main(String[] args)
    {
        int count = 10;
        while (count >= 1)
        {
            System.out.println("Count is: " + count);
            count--;
        }
    }
}
```

Break

- We saw **break** used before with switch
- It can also be used in loops to exit the loop ‘early’



```
for (int i = 0; i <= 10; i++)
{
    if (i == 5)
    {
        System.out.println("Reached i==5!");
        break;
    }
}
```

- This loop is set up to run 11 times, but will exit after 5 iterations because of **break;**