Loops

Lecture 4a

Topics (1 of 2)

- The Increment and Decrement Operators
- The `while` Loop
- Using the `while` Loop for Input Validation
- Mixing Calls to nextLine with Calls to Other Scanner Methods
- The `do-while` Loop
- The `for` Loop
- Running Totals and Sentinel Values

Topics (2 of 2)

- Nested Loops
- The `break` and `continue` Statements
- Deciding Which Loop to Use
- Generating Random Numbers with the `Random` class

The Increment and Decrement Operators (1 of 3)

- There are numerous times where a variable must simply be incremented or decremented.
- Java provide shortened ways to increment and decrement a variable's value.

```
number = number + 1
number = number - 1
```

- Using the **++** or **--** unary operators, this task can be completed quickly.

```
number++;   or   ++number;
number--;   or   --number;
```

Example: [IncrementDecrement.java]

The Increment and Decrement Operators (of 3)

- Point to ponder #1:

```
int number = 2;
number++;
System.out.println(number);  3

number++;
System.out.println(number);  4

number--;
System.out.println(number);  3
```

Operators here are used in postfix mode!

You will also find the expression post-increment out there!

The Increment and Decrement Operators (1 of 3)

- Point to ponder #2:

```
int number = 2;
++number;
System.out.println(number);
```
3

```
++number;
System.out.println(number);
```
4

```
--number;
System.out.println(number);
```
3

Operators here are used in prefix mode!

You will also find the expression pre-increment out there!

Differences Between Prefix and Postfix (1 of 4)

- When an increment or decrement are the **ONLY** operations in a statement, there is no difference between prefix and postfix notation.
- When used in an expression:
  - prefix notation indicates that the variable will be incremented or decremented prior to the rest of the expression being evaluated.
  - postfix notation indicates that the variable will be incremented or decremented after the rest of the expression has been evaluated.

Differences Between Prefix and Postfix (2 of 4)

- Point to ponder #3:

```
int number = 4;
System.out.println(number++);        4
System.out.println(number);          5

int x = 1, y;
y = x++;
System.out.println(y);               1
System.out.println(x);               2
```

Differences Between Prefix and Postfix (3 of 4)

- Point to ponder #4:

```
int number = 4;
System.out.println(++number);        5
System.out.println(number);          5
```

```
int x = 1, y;
y = ++x;
System.out.println(y);               2
System.out.println(x);               2
```

Differences Between Prefix and Postfix (4 of 4)

- Point to ponder #5:

```
int x = 1, y = 1;
y = --y+(x++);
System.out.println(y);
```

1

```
int x = 1, y = 1;
y = --y-(--x);
System.out.println(y);
```

0

Loops (1 of 5)

- Suppose you want to print the integer numbers in a small interval, for instance [1,5]. How would you approach that considering what we have learned so far?

```
System.out.println(1);
System.out.println(2);
System.out.println(3);
System.out.println(4);
System.out.println(5);
```

Easy!!!

Loops (2 of 5)

- How about a larger interval, for instance [1,20]. How would you approach that?

```
System.out.println(1);
System.out.println(2);
System.out.println(3);
System.out.println(4);
System.out.println(5);
System.out.println(6);
System.out.println(7);
System.out.println(8);
System.out.println(9);
System.out.println(10);
System.out.println(11);
System.out.println(12);
System.out.println(13);
System.out.println(14);
System.out.println(15);
System.out.println(16);
System.out.println(17);
System.out.println(18);
System.out.println(19);
System.out.println(20);
```

It is getting complicated!!!

Loops (3 of 5)

- How about a much, much, much larger interval, for instance [1,1000]. How would you approach that?

```
System.out.println(1);
System.out.println(2);
System.out.println(3);
System.out.println(4);
System.out.println(5);
System.out.println(6);
System.out.println(7);
System.out.println(8);
System.out.println(9);
System.out.println(10);
System.out.println(11);
System.out.println(12);
System.out.println(13);
System.out.println(14);
System.out.println(15);
System.out.println(16);
System.out.println(17);
System.out.println(18);
System.out.println(19);
System.out.println(20);
System.out.println(21);
System.out.println(22);
System.out.println(23);
System.out.println(24);
System.out.println(25);
System.out.println(26);
System.out.println(27);
System.out.println(28);
System.out.println(29);
System.out.println(30);
System.out.println(31);
System.out.println(32);
System.out.println(33);
System.out.println(34);
System.out.println(35);
System.out.println(36);
System.out.println(37);
System.out.println(38);
System.out.println(39);
System.out.println(40);
```

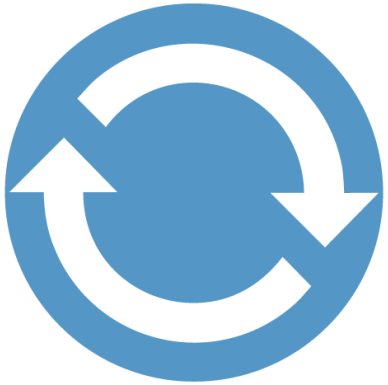
. . .

OMG!!!

There must be another way

Loops (4 of 5)

- What you need is to use a control structure called loop. But what is a loop?



A part of a program that repeats! More formally, a loop is a control structure that causes a statement or group of statements to repeat.

Loops (5 of 5)

- Java provides three different looping structures to control repetitions.
  - `while loop`
  - `do-while loop`
  - `for loop`

The `while` Loop (1 of 9)

- The `while` loop has the form:

Loop header

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

boolean expression

loop body

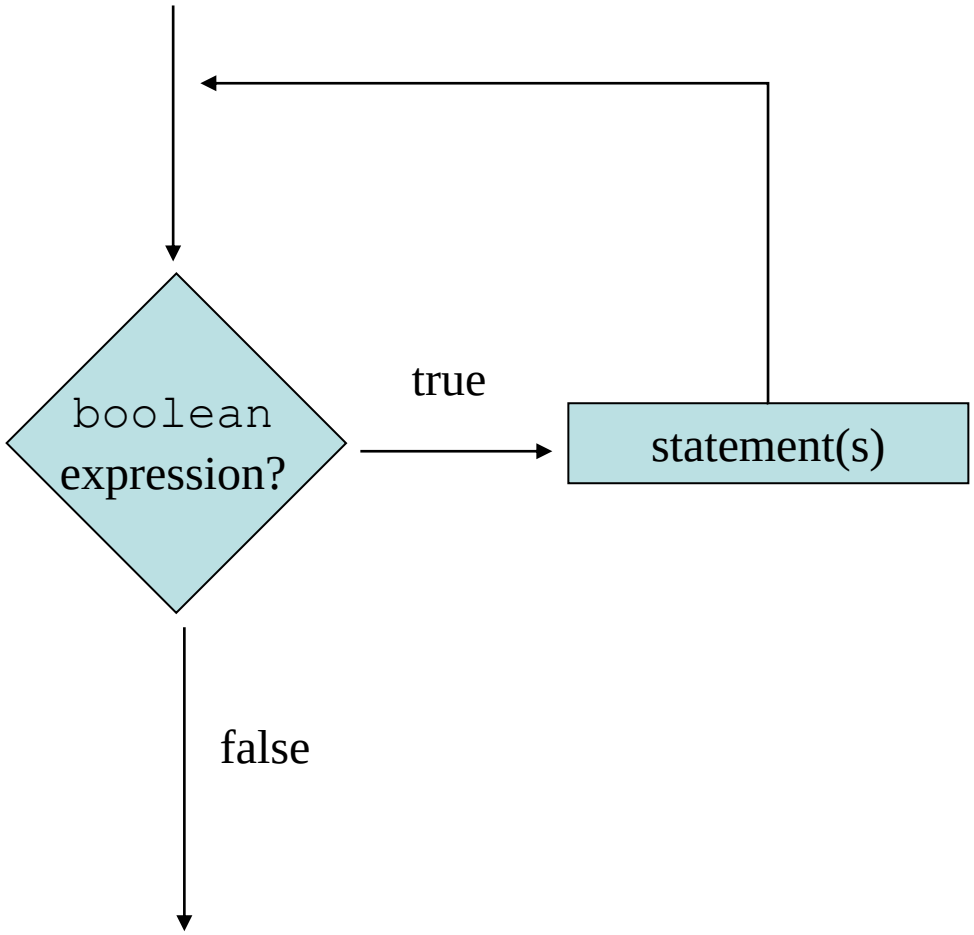
- While the condition is true, the statements will execute repeatedly.
- The `while` loop is a *pretest* loop, which means that it will test the value of the condition prior to executing the loop.

The `while` Loop (2 of 9)

- Care must be taken to set the condition to false somewhere in the loop so the loop will end.
- Loops that do not end are called *infinite loops.*
- A `while` loop executes 0 or more times. If the condition is false, the loop will not execute.

The `while` Loop (3 of 9)

- Flowchart



The `while` Loop (4 of 9)

- Example: `WhileLoop.java`

```
1 public class WhileLoop
2 {
3   public static void main(String[] args)
4   {
5      int number = 1;
6
7      while (number <= 5)
8      {
9         System.out.println("Hello");
10        number++;
11     }
12     System.out.println("End!");
13  }
14}
```

- Point to ponder #6:

Output?

```
Hello
Hello
Hello
Hello
Hello
End!
```

The `while` Loop (5 of 9)

Test this `boolean` expression.

If the `boolean` expression is `true`, perform these statements.

```
while (number <= 5)
{
      System.out.println("Hello");
      number++;
}
```

After executing the body of the loop, start over.

Debugging (1 of 19)

```
1  public class WhileLoop
2  {
3    public static void main(String[] args)
4    {
5       int number = 1;
6
7       while (number <= 5)
8       {
9          System.out.println("Hello");
10         number++;
11      }
12      System.out.println("End!");
13   }
14 }
```

Variables/output:

iteration	number	output
	1	

Debugging (2 of 19)

```
1 public class WhileLoop
2 {
3   public static void main(String[] args)
4   {
5       int number = 1;
6
7       while (number <= 5)
8       {
9           System.out.println("Hello");
10          number++;
11      }
12      System.out.println("End!");
13  }
14}
```

Variables/output:

iteration	number	output
	1	

Debugging (3 of 19)

```
1 public class WhileLoop
2 {
3   public static void main(String[] args)
4   {
5       int number = 1;
6
7       while (number <= 5)
8       {
9           System.out.println("Hello");
10          number++;
11      }
12      System.out.println("End!");
13  }
14}
```

Variables/output:

iteration	number	output
1	1	Hello

Debugging (4 of 19)

```
1  public class WhileLoop
2  {
3    public static void main(String[] args)
4    {
5       int number = 1;
6
7       while (number <= 5)
8       {
9          System.out.println("Hello");
10         number++;
11      }
12      System.out.println("End!");
13   }
14 }
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello

Debugging (5 of 19)

```
1 public class WhileLoop
2 {
3   public static void main(String[] args)
4   {
5       int number = 1;
6
7       while (number <= 5)
8       {
9           System.out.println("Hello");
10          number++;
11      }
12      System.out.println("End!");
13  }
14}
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
	2	

Debugging (6 of 19)

```
1  public class WhileLoop
2  {
3    public static void main(String[] args)
4    {
5       int number = 1;
6
7       while (number <= 5)
8       {
9          System.out.println("Hello");
10         number++;
11      }
12      System.out.println("End!");
13   }
14 }
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	2	Hello

Debugging (7 of 19)

```
1 public class WhileLoop
2 {
3   public static void main(String[] args)
4   {
5       int number = 1;
6
7       while (number <= 5)
8       {
9           System.out.println("Hello");
10          number++;
11      }
12      System.out.println("End!");
13  }
14}
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~2~~3	Hello

Debugging (8 of 19)

```
1  public class WhileLoop
2  {
3    public static void main(String[] args)
4    {
5       int number = 1;
6
7       while (number <= 5)
8       {
9          System.out.println("Hello");
10         number++;
11      }
12      System.out.println("End!");
13   }
14 }
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~23~~	Hello
	3	

Debugging (9 of 19)

```
1 public class WhileLoop
2 {
3   public static void main(String[] args)
4   {
5       int number = 1;
6
7       while (number <= 5)
8       {
9           System.out.println("Hello");
10          number++;
11      }
12      System.out.println("End!");
13  }
14}
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~23~~	Hello
3	3	Hello

Debugging (10 of 19)

```
1  public class WhileLoop
2  {
3    public static void main(String[] args)
4    {
5       int number = 1;
6
7       while (number <= 5)
8       {
9          System.out.println("Hello");
10         number++;
11      }
12      System.out.println("End!");
13   }
14 }
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~2~~3	Hello
3	~~3~~4	Hello

Debugging (11 of 19)

```
1  public class WhileLoop
2  {
3    public static void main(String[] args)
4    {
5       int number = 1;
6
7       while (number <= 5)
8       {
9          System.out.println("Hello");
10         number++;
11      }
12      System.out.println("End!");
13   }
14 }
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~23~~	Hello
3	~~34~~	Hello
	4	

Debugging (12 of 19)

```
1  public class WhileLoop
2  {
3    public static void main(String[] args)
4    {
5       int number = 1;
6
7       while (number <= 5)
8       {
9          System.out.println("Hello");
10         number++;
11      }
12      System.out.println("End!");
13   }
14 }
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~23~~	Hello
3	~~34~~	Hello
4	4	Hello

Debugging (13 of 19)

```
1 public class WhileLoop
2 {
3   public static void main(String[] args)
4   {
5       int number = 1;
6
7       while (number <= 5)
8       {
9           System.out.println("Hello");
10          number++;
11      }
12      System.out.println("End!");
13  }
14}
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~23~~	Hello
3	~~34~~	Hello
4	~~45~~	Hello

Debugging (14 of 19)

```
1  public class WhileLoop
2  {
3    public static void main(String[] args)
4    {
5       int number = 1;
6
7       while (number <= 5)
8       {
9          System.out.println("Hello");
10         number++;
11      }
12      System.out.println("End!");
13   }
14 }
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~23~~	Hello
3	~~34~~	Hello
4	~~45~~	Hello
	5	

Debugging (15 of 19)

```
1  public class WhileLoop
2  {
3    public static void main(String[] args)
4    {
5       int number = 1;
6
7       while (number <= 5)
8       {
9          System.out.println("Hello");
10         number++;
11      }
12      System.out.println("End!");
13   }
14 }
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~23~~	Hello
3	~~34~~	Hello
4	~~45~~	Hello
5	5	Hello

Debugging (16 of 19)

```
1 public class WhileLoop
2 {
3   public static void main(String[] args)
4   {
5       int number = 1;
6
7       while (number <= 5)
8       {
9           System.out.println("Hello");
10          number++;
11      }
12      System.out.println("End!");
13  }
14}
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~23~~	Hello
3	~~34~~	Hello
4	~~45~~	Hello
5	~~56~~	Hello

Debugging (17 of 19)

```
1 public class WhileLoop
2 {
3   public static void main(String[] args)
4   {
5       int number = 1;
6
7       while (number <= 5)
8       {
9           System.out.println("Hello");
10          number++;
11      }
12      System.out.println("End!");
13  }
14}
```

Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~23~~	Hello
3	~~34~~	Hello
4	~~45~~	Hello
5	~~56~~	Hello
	6	

Debugging (18 of 19)

```
1 public class WhileLoop
2 {
3   public static void main(String[] args)
4   {
5       int number = 1;
6
7       while (number <= 5)
8       {
9           System.out.println("Hello");
10          number++;
11      }
12      System.out.println("End!");
13  }
14}
```

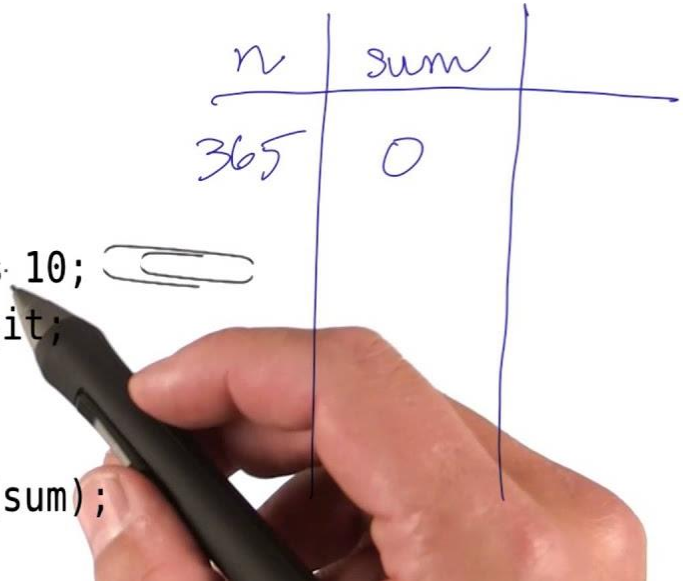
Variables/output:

iteration	number	output
1	~~1~~2	Hello
2	~~23~~	Hello
3	~~34~~	Hello
4	~~45~~	Hello
5	~~56~~	Hello
-	6	End!

Debugging (18 of 19)

- How do you call the technique that simulates the programs activity in a sheet of paper?

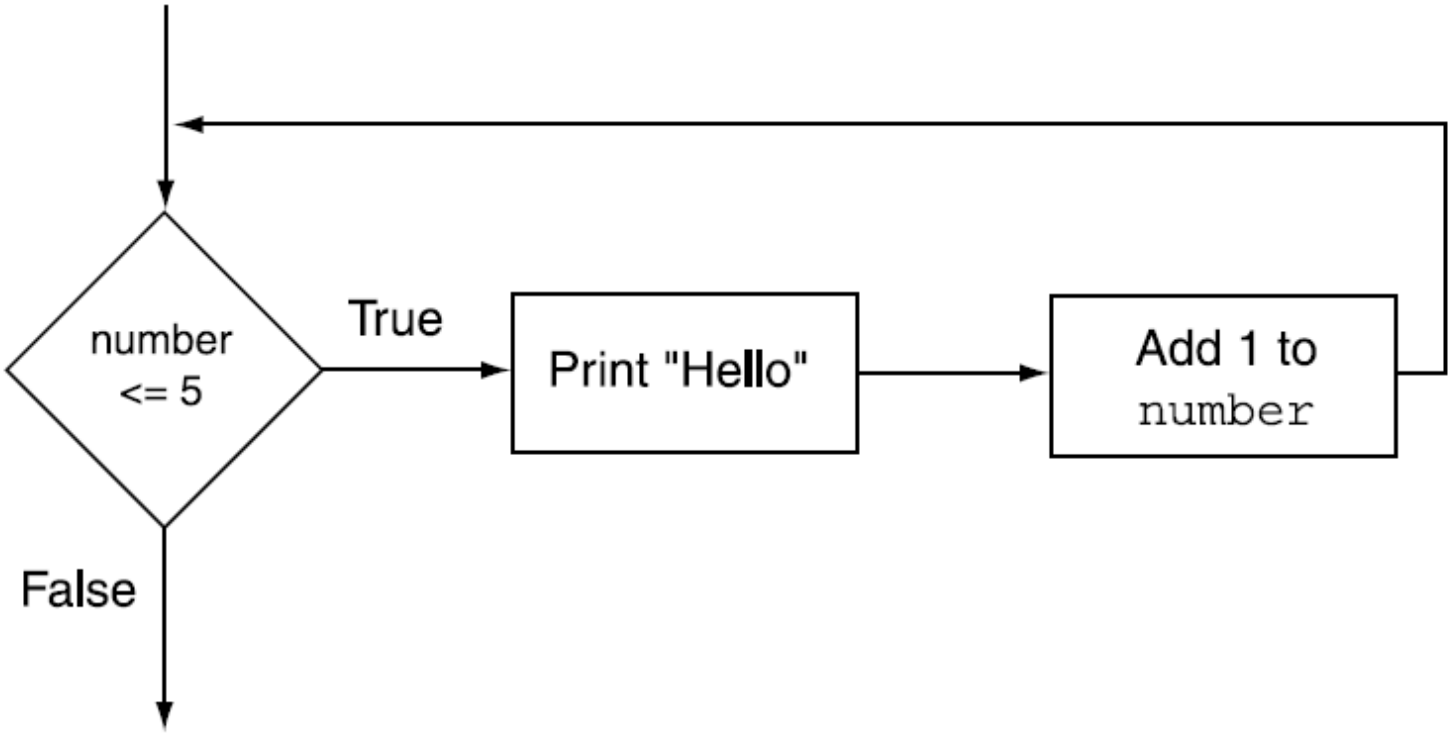
```
int n = 365;
int sum = 0;
while (n > 0)
{
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
System.out.println(sum);
```



Hand-tracing!

The `while` Loop (6 of 9)

- Logic of the loop



The `while` Loop (7 of 9)

```
while (number <= 5)
{
    System.out.println("Hello");
    number++;
}
```

- Point to ponder #7:

  What is the name of the variable that controls the number of times the loop iterates?

  `Loop control variable (number)`

The `while` Loop (8 of 9)

- Point to ponder #8:

  Is the while loop a pre-test or a post-test loop? Why?

  Pretest loop since it tests its expression before each iteration.

```
while (number <= 5)
{
    System.out.println("Hello");
    number++;
}
```

Required test from the start

The `while` Loop (9 of 9)

- Point to ponder #9:

How many iterations will be produced here?

```
int number = 6;
while (number <= 5)
{
    System.out.println("Hello");
    number++;
}
```

Zero iterations! The loop condition (boolean expression) is false from the start.

Infinite Loops (1 of 3)

- In order for a `while` loop to end, the condition must become false at some point. The following loop will not end:

```
int x = 20;
while(x > 0)
{
   System.out.println("x is greater than 0");
}
```

- The variable `x` will never reach the value 0.

Infinite Loops (2 of 3)

- This version of the loop decrements `x` during each iteration:

```
int x = 20;
while(x > 0)
{
    System.out.println("x is greater than 0");
   x--;
}
```

Infinite Loops (3 of 3)

- Point to ponder #10:

  What would be the output of this loop?

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

No output. It is also an infinite loop! Be careful with semicolons.

Block Statements in Loops (1 of 2)

- Curly braces are required to enclose block statement while loops (like block `if` statements). Otherwise, only the very next statement is conditionally executed (part of the loop).

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

Block Statements in Loops (2 of 2)

- Point to ponder #11:

  What would be the output of this loop?

```
int number = 1;
while (number <= 5)
      System.out.println("Hello");
      number++;
```

Infinite “Hello”. It is also an infinite loop! Be careful with the absence of braces.

Block Statements in Loops (2 of 2)

- Fixing the issue …

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

Programming Style

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

- Do not place the loop body in the same line of the loop header
- Indent all statements in the loop body

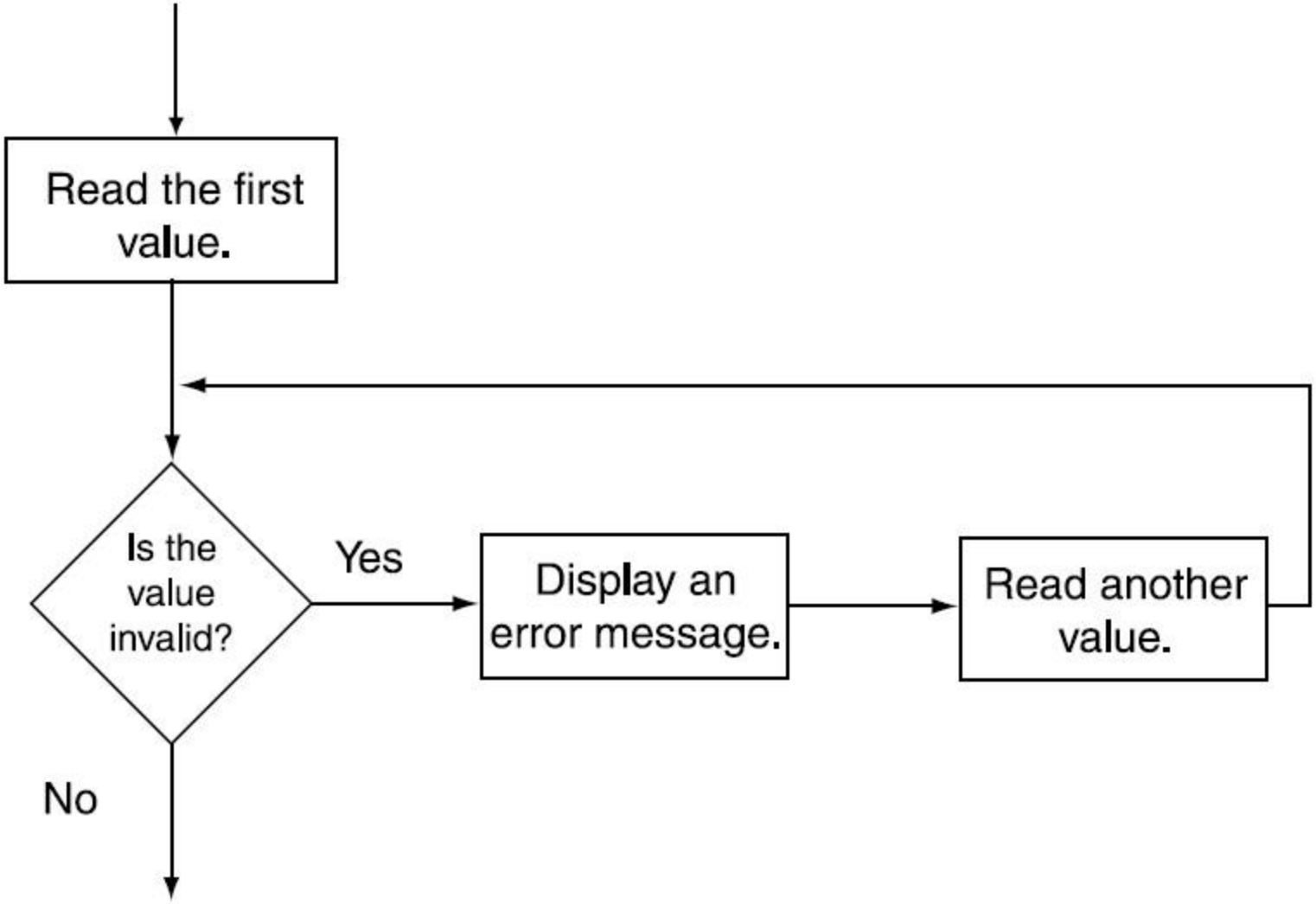
The `while` Loop for Input Validation

- *Input validation* is the process of ensuring that user input is valid.

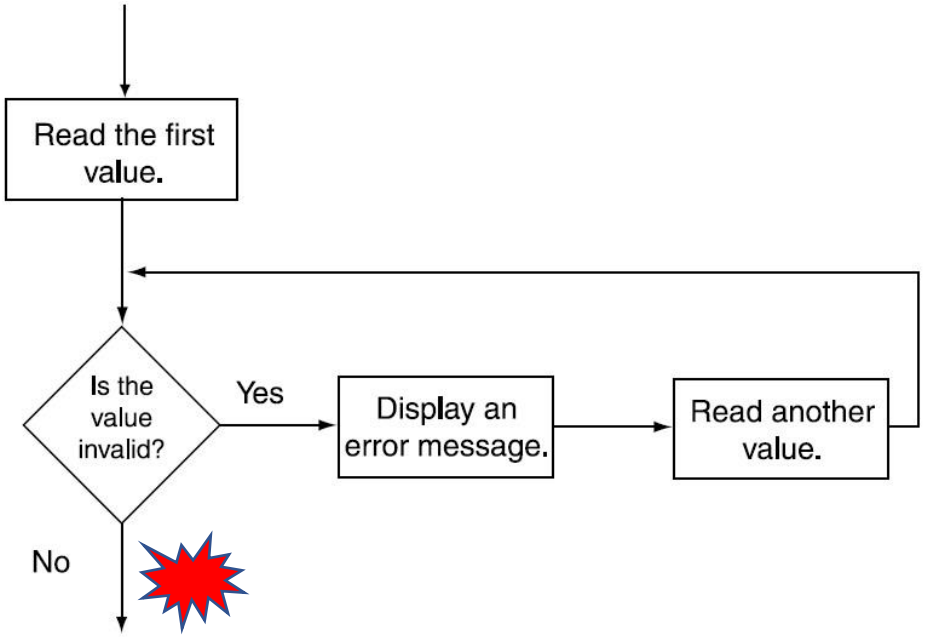
```
System.out.print("Enter a number in the " +
                 "range of 1 through 100: ");
number = keyboard.nextInt();
// Validate the input.
while (number < 1 || number > 100)
{
  System.out.println("That number is invalid.");
  System.out.print("Enter a number in the " +
                   "range of 1 through 100: ");
  number = keyboard.nextInt();
}
```

- Example: [SoccerTeams.java]

The `while` Loop for Input Validation



The `while` Loop for Input Validation



- Point to ponder #12:

  Which are the main three groups of people that programmers hate because they discover not validated-inputs crashing the system?

Users 

Testers 

Hackers 

Loops

Lecture 4b

Topics (1 of 2)

- The Increment and Decrement Operators
- The `while` Loop
- Using the `while` Loop for Input Validation
- Mixing Calls to nextLine with Calls to Other Scanner Methods
- The `do-while` Loop
- The `for` Loop
- Running Totals and Sentinel Values

Topics (2 of 2)

- Nested Loops
- The `break` and `continue` Statements
- Deciding Which Loop to Use
- Generating Random Numbers with the `Random` class

Mixing Calls to nextLine with Calls to Other Scanner Methods (1 of 4)

- The `nextInt`, `nextDouble`, and `next` methods do not read the newline character that follows a number or word entered by a user.
- This can be a problem if you alternate between calling `nextInt/nextDouble/next` and `nextLine`.
- When you enter a number then press <Enter>, `input.nextInt()` consumes only the number, not the “\n”). When `input.nextLine()` executes, it consumes the "\n" still in the buffer from the first input.

Mixing Calls to nextLine with Calls to Other Scanner Methods (2 of 4)

- For instance: user input (“John”, 10, “Pomona”)

```
Scanner scanner = new Scanner(System.in);
System.out.println("Enter your name: ");
String name = scanner.nextLine();
System.out.println("Enter your age: ");
int age = scanner.nextInt();
System.out.println("Enter your city: ");
String city = scanner.nextLine();
System.out.println("Hi " + name + ", your age is
                " + age + " and your city is " + city);
```

- Point to ponder #6:

  Output?

Hi John, your age is 10 and your city is

Mixing Calls to nextLine with Calls to Other Scanner Methods (3 of 4)

- In a sequence, the user input is:

J	o	h	n	\n	1	0	\n	P	o	m	o	n	a	\n

- After the first call to the `nextLine` method, the input is:

1	0	\n	P	o	m	o	n	a	\n

- After the call to the `nextInt` method, the input is:

\n	P	o	m	o	n	a	\n

- Note that the `nextInt` call did not consume the newline character. Therefore, the second call to `nextLine` reads an empty string!

Mixing Calls to nextLine with Calls to Other Scanner Methods (4 of 4)

- The remedy is to add a call to `nextLine` after reading the user's age:

```
Scanner scanner = new Scanner(System.in);
System.out.println("Enter your name: ");
String name = scanner.nextLine();
System.out.println("Enter your age: ");
int age = scanner.nextInt();
scanner.nextLine(); //consume the newline
System.out.println("Enter your city: ");
String city = scanner.nextLine();
System.out.println("Hi " + name + ", your age is
                " + age + " and your city is " + city);
```

Output: Hi John, your age is 10 and your city is Pomona

The `do-while` Loop (1 of 4)

- The `do-while` loop is a *posttest* loop, which means it will execute the loop prior to testing the condition (the `boolean` expression is tested after each iteration).
- The `do-while` loop (sometimes called a `do` loop) takes the form:

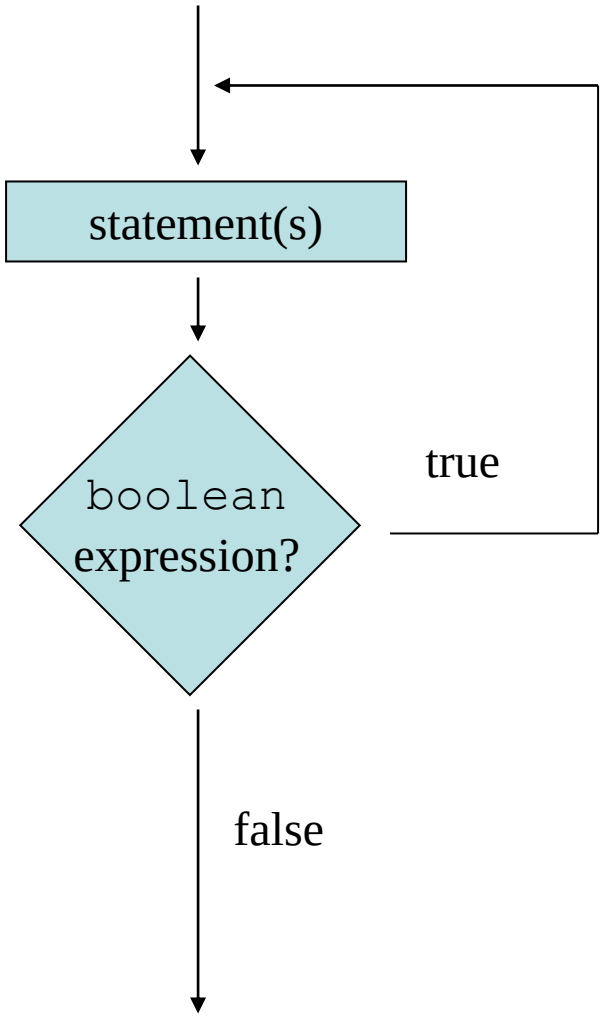
```
do
{
  statement(s);
}
while (condition);
```

← Don't forget the semicolon here!

- Example: TestAverage1.java

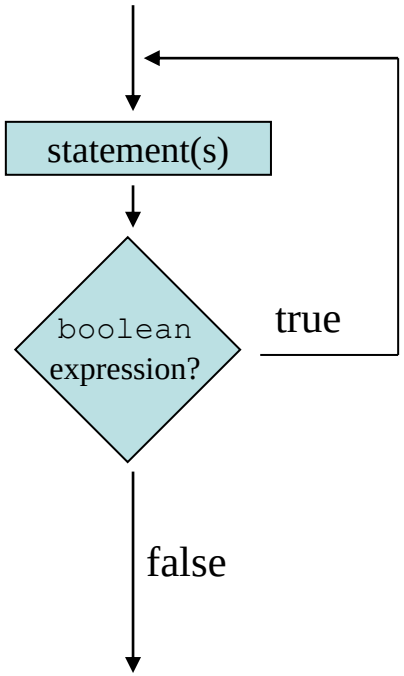
The `do-while` Loop (2 of 4)

- Flowchart

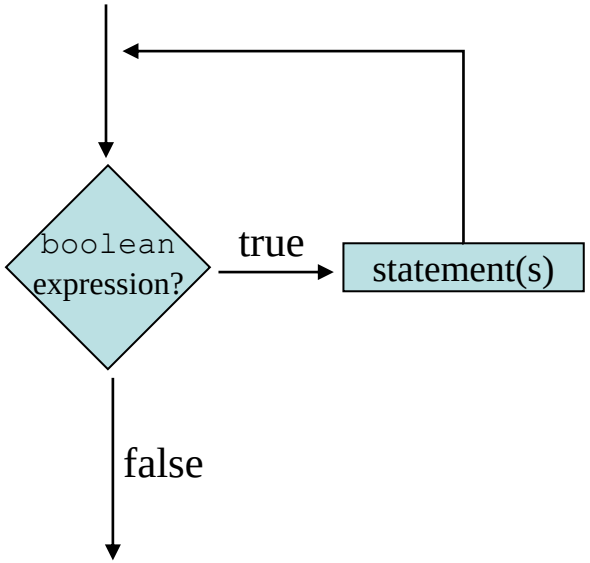


The do-while Loop (3 of 4)

do-while loop



while loop



do-while loop ALWAYS performs at least one iteration, even if the boolean expression is false to begin with.

The `do-while` Loop (4 of 4)

- Point to ponder #13:

  What would be the output of this loop?

```
int number = 1;
do {
    System.out.println("Hello");
    number++;
}
while (number <= 1);
```

```
Hello
```

The `for` Loop (1 of 26)

- In general, there are two categories of loops: **conditional loops** and **count-controlled loops**.
- A **conditional loop** executes as long as a particular condition exists.

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

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

- You have no way of knowing the number of times it will iterate!

The `for` Loop (2 of 26)

- Point to ponder #1:

  How many times this loop will iterate?

```
Scanner input = new Scanner(System.in);
int x = input.nextInt();
while(x <= 10)
{
   System.out.println(x);
   x++;
}
```

Depends on the user input. For x <= 10, 10 – x + 1 iterations, otherwise 0 iteration.

The `for` Loop (3 of 26)

- Point to ponder #2:

  How about now?

```
Scanner input = new Scanner(System.in);
int x = input.nextInt();
do
{
   System.out.println(x);
   x++;
} while(x <= 10);
```

Depends on the user input. For x <= 10, 10 – x + 1 iterations, otherwise 1 iteration.

The `for` Loop (4 of 26)

- Sometimes you do know the exact number of iterations that a loop must perform
- A loop that repeats a specific number of times is known as a count-controlled loop
- In Java, the `for` loop is ideal for writing count-controlled loops.

The `for` Loop (5 of 26)

- It allows the programmer to initialize a control variable, test a condition, and modify the control variable all in one line of code.
- The `for` loop takes the form:

```
for(initialization; test; update)
{
   statement(s);
}
```

The `for` Loop (6 of 26)

- Sequence of events in the for loop

Step1: Perform the initialization expression (done only once)

Step2: Evaluate the test expression. If it is true, go to Step 3, otherwise, terminate the loop

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

Step3:Execute the body of the loop.

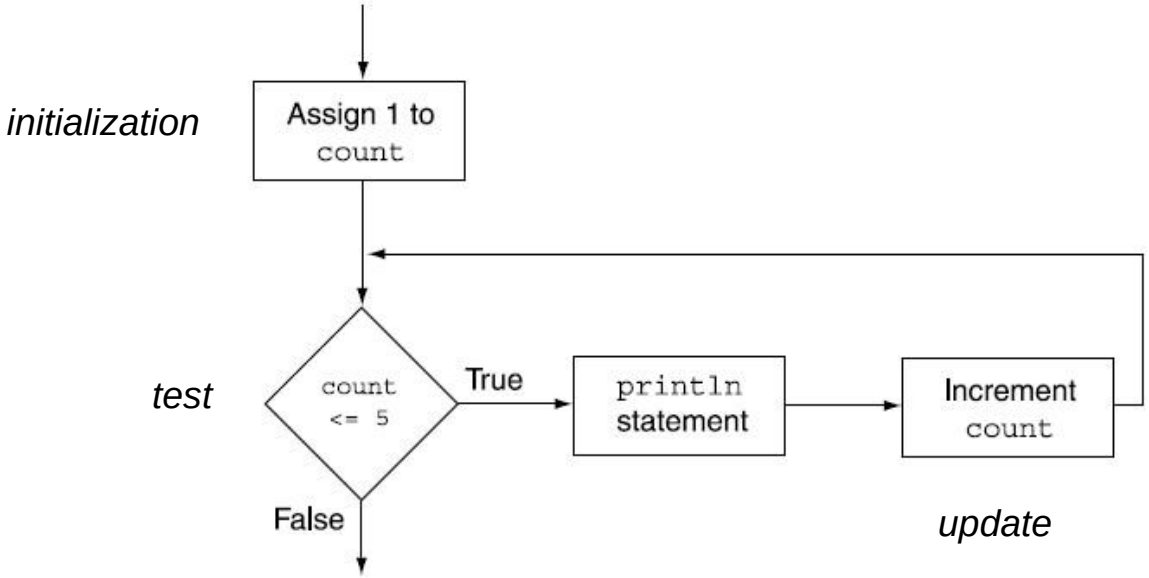
Step4:Perform the update expression, then go back to Step 2.

- Because `count` keeps a count of the number of iterations, it is called a counter variable, making `for` a count-controlled loop.

The `for` Loop (7 of 26)

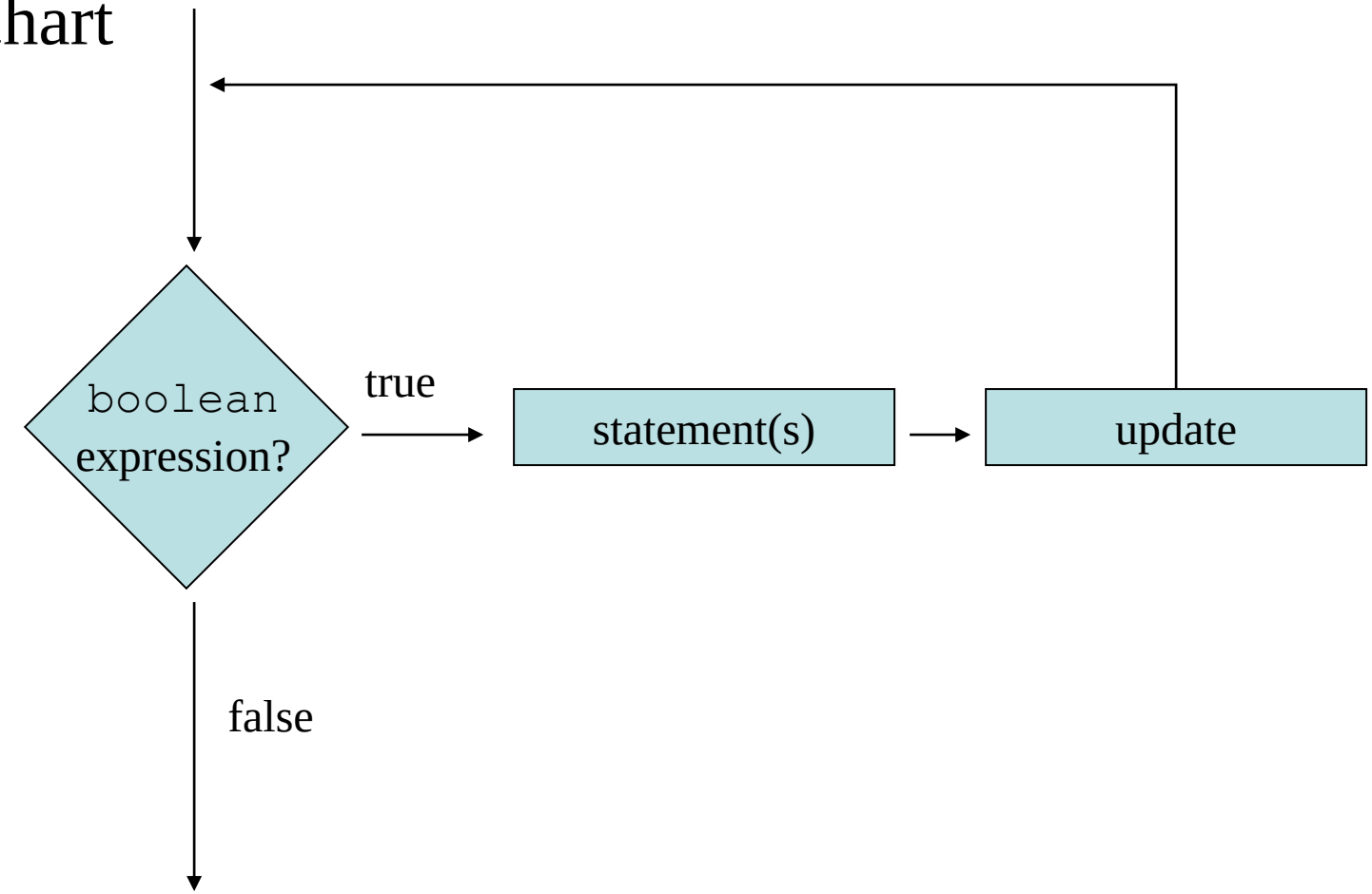
- Sequence of events in the for loop

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



The `for` Loop (8 of 26)

- Flowchart



The `for` Loop (9 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

- Point to ponder #2:

  Output?

```
Number    Number Squared
------------------------
1         1
2         4
3         9
4         16
5         25
```

The `for` Loop (10 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
	1	

The `for` Loop (11 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
1	1	1 1

The `for` Loop (12 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
1	1	1 1
	2	

The `for` Loop (13 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
1	1	1 1
2	2	2 4

The `for` Loop (14 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
1	1	1 1
2	2	2 4
	3	

The `for` Loop (15 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
1	1	1 1
2	2	2 4
3	3	3 9

The `for` Loop (16 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
1	1	1 1
2	2	2 4
3	3	3 9
	4	

The `for` Loop (17 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
1	1	1 1
2	2	2 4
3	3	3 9
4	4	4 16

The `for` Loop (18 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
1	1	1 1
2	2	2 4
3	3	3 9
4	4	4 16
	5	

The `for` Loop (19 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
1	1	1 1
2	2	2 4
3	3	3 9
4	4	4 16
5	5	5 25

The `for` Loop (20 of 26)

See example: `Squares.java`

```
public class Squares
{
  public static void main(String[] args)
  {

    int number; // Loop control variable

    System.out.println("Number \t\t Number Squared");
    System.out.println("------------------------");

    for (number = 1; number <= 5; number++)
    {
       System.out.println(number + "\t\t" +
                          number * number);
    }
  }
}
```

Variables/output:

iteration	number	output
1	1	1 1
2	2	2 4
3	3	3 9
4	4	4 16
5	5	5 25
	6	

The `for` Loop (21 of 26)

- The `for` loop is a pre-test loop.

```
for(count = 11; count <= 10; count ++)
{
    System.out.println("Hello");
}
```

- Point to ponder #3:

  How many times this loop will iterate?

  Zero iterations.

The `for` Loop (22 of 26)

- Avoid modifying the control variable inside the loop. The loop probably will not terminate when you expect it to.

```
for (x = 1; x <= 10; x++)
{
   System.out.print(x + " ");
   x++; // Wrong!
}
```

- Point to ponder #4:

  How many times this loop will iterate? 5 iterations.

- Point to ponder #5:

  What is the output? 1 3 5 7 9

The `for` Loop (23 of 26)

- Other forms of the update expression

```
for (x = 0; x <= 10; x+=5)
{
   System.out.print(x + " ");
}
```

- Point to ponder #6:

  How many times this loop will iterate? 3 iterations.

- Point to ponder #7:

  What is the output? 0 5 10

The `for` Loop (24 of 26)

- Declaring a variable in the for loop's initialization expression

```
for (int x = 1; x <= 10; x++)
{
   System.out.print("Hello");
}
System.out.print(x);
```

- Point to ponder #8:

  What is the output?

Compile time error. Variable `x` is not recognized.

The `for` Loop (25 of 26)

- Multiple initializations and updates

```
for(int i = 5, int j = 0; i < 10 || j < 20; i++, j+=2)
{
   statement(s);
}
```

The `for` Loop (26 of 26)

- Be careful with infinite loops

```
for(int i = 1; i < 2; i--)
{
   System.out.println("Test");
}

for(;;)
{
   System.out.println("Test");
}
```

Loops

Lecture 4c

Topics (1 of 2)

- The Increment and Decrement Operators
- The `while` Loop
- Using the `while` Loop for Input Validation
- The `do-while` Loop
- The `for` Loop
- Running Totals and Sentinel Values

Topics (2 of 2)

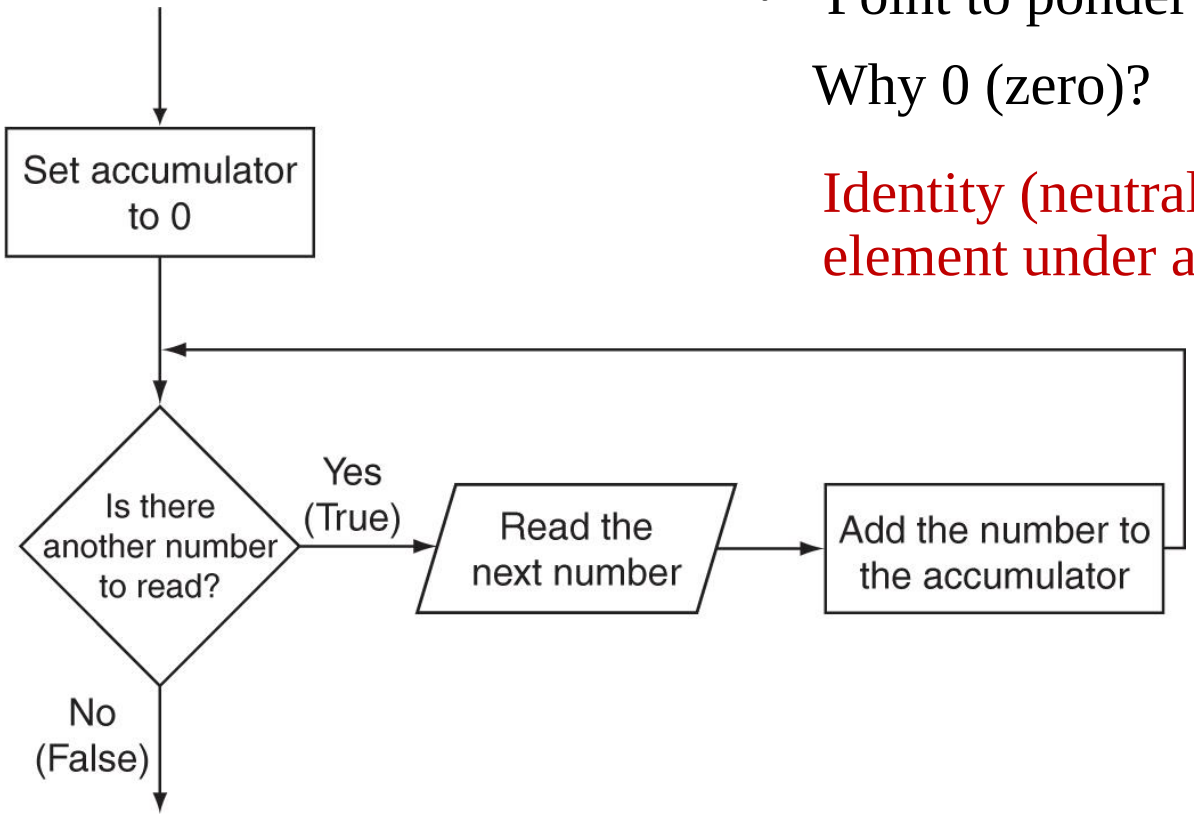
- Nested Loops
- The `break` and `continue` Statements
- Deciding Which Loop to Use
- Generating Random Numbers with the `Random` class

Running Totals (1 of 3)

- A running total is a sum of numbers that accumulates with each iteration of a loop.
- It can be used to calculate the total of a series of read numbers
- The variable used to keep the running total is called an *accumulator*

Running Totals (2 of 3)

- Logic for Calculating a Running Total



- Point to ponder #1:

  Why 0 (zero)?

  Identity (neutral) element under addition.

Running Totals (3 of 3)

Example. `TotalSales.java`

```
double sales; // A day's sales figure

System.out.println("For how many days do you have sales figures?");
int days = scanner.nextInt();

//Set the accumulator to 0.
double totalSales = 0.0;

// Get the sales figures and calculate a running total.
for (int count = 1; count <= days; count++)
{
  System.out.println("Enter the sales for day " + count + ": ");
  sales = scanner.nextDouble();
  totalSales += sales; // Add sales to totalSales.
}
// Display the total sales.
System.out.printf("The total sales are $%,.2f", totalSales);
```

Sentinel Values (1 of 3)

- Sometimes the end point of input data is not known.
- A *sentinel value* can be used to notify the program to stop acquiring input.
- The user can be prompted to input data that is not normally in the input data range (i.e., –1, where normal input would be positive.)
- Programs that get file input typically use the end-of-file marker to stop acquiring input data.

Sentinel Values (2 of 3)

Example.

```
double totalSales = 0.0;

System.out.println("Enter the sales or -1 to end: ");
double sales = scanner.nextDouble();

while (sales != -1) //Set a sentinel value
{
  totalSales += sales; // Add sales to totalSales.

 // Get the next sales.
  System.out.println("Enter the sales or -1 to end: ");
  sales = scanner.nextDouble();
}
// Display the total sales.
System.out.printf("The total sales are $%,.2f", totalSales);
```

Sentinel Values (3 of 3)

Same example (refactoring the code).

```
double totalSales = 0.0;
double sales = 0.0;

while (sales != -1) //Set a sentinel value
{
  totalSales += sales; // Add sales to totalSales.

 // Get the next sales.
  System.out.println("Enter the sales or -1 to end: ");
  sales = scanner.nextDouble();
}
// Display the total sales.
System.out.printf("The total sales are $%,.2f", totalSales);
```

Nested Loops (1 of 16)

- Like `if` statements, loops can be nested.
- If a loop is nested, the inner loop will execute all of its iterations for each time the outer loop executes once.

```
for(int i = 0; i < 10; i++)
    for(int j = 0; j < 10; j++)
        loop statements;
```

Outer loop

Inner loop

- Point to ponder #2:

  How many times the loop statements here will execute? 100

Nested Loops (2 of 16)

Clock.java

```
public class Clock {
  public static void main(String[] args)  {
    for (int hours = 1; hours <= 12; hours++)  {
        for (int minutes = 0; minutes <= 59; minutes++)  {
          for (int seconds = 0; seconds <= 59; seconds++)  {
            System.out.printf("%02d:%02d:%02d\n", hours, minutes, seconds);
          }
        }
      }
  }
}
```

See `TestAverage2.java`

Nested Loops (3 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0		

Nested Loops (4 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	

Nested Loops (5 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0

Nested Loops (6 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0
0	1	

Nested Loops (7 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0
0	1	0 1

Nested Loops (8 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0
0	1	0 1
0	2	

Nested Loops (9 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0
0	1	0 1
0	2	-
1		

Nested Loops (10 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0
0	1	0 1
0	2	-
1	0	

Nested Loops (11 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0
0	1	0 1
0	2	-
1	0	1 0

Nested Loops (12 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0
0	1	0 1
0	2	-
1	0	1 0
1	1	

Nested Loops (13 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0
0	1	0 1
0	2	-
1	0	1 0
1	1	1 1

Nested Loops (14 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0
0	1	0 1
0	2	-
1	0	1 0
1	1	1 1
1	2	

Nested Loops (15 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

Variables/output:

i	j	output
0	0	0 0
0	1	0 1
0	2	-
1	0	1 0
1	1	1 1
1	2	-
2		

Nested Loops (16 of 16)

```
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++)
       System.out.println(i + " " + j);
```

…

Output

0 0

0 1

1 0

1 1

Variables/output:

i	j	output
0	0	0 0
0	1	0 1
0	2	-
1	0	1 0
1	1	1 1
1	2	-
2	-	-

The `break` Statement (1 of 19)

- The `break` statement can be used to abnormally terminate a loop.
- The use of the `break` statement in loops bypasses the normal mechanisms and makes the code hard to read and maintain.
- It is “considered bad” form to use the `break` statement in this manner.

The `break` Statement (2 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(j);
```

```
int i = 0, j = 0;
while (i < 3){
    j += i + 2;
    i++;
}
System.out.println(j);
```

- Point to ponder #3:

  Outputs?

9

9

The `break` Statement (3 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0

The `break` Statement (4 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0

The `break` Statement (5 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
	2

The `break` Statement (6 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2

The `break` Statement (7 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2

The `break` Statement (8 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2

The `break` Statement (9 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2
	5

The `break` Statement (10 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2
2	5

The `break` Statement (11 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2
2	5

The `break` Statement (12 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2
2	5

The `break` Statement (13 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2
2	5
	9

The `break` Statement (14 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2
2	5
3	9

The `break` Statement (15 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2
2	5
3	9

The `break` Statement (16 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

Variables:

i	j
0	0
1	2
2	5
3	9

The `break` Statement (17 of 19)

```
int i = 0, j = 0;
while (true){
    j += i + 2;
    i++;

    if (i == 3)
        break;
}
System.out.println(i + " " + j);
```

3 9

Variables:

i	j
0	0
1	2
2	5
3	9

The `break` Statement (18 of 19)

- Good use of the `break` statement. Searching a character in a string …

```
System.out.println("Enter the word");
String word = scanner.nextLine(); //read the input until a space
System.out.println("Enter the character to be searched");
String character = scanner.next(); //read the input until the end

boolean found = false;
for(int i = 0; i < word.length(); i++) {

    if (String.valueOf(word.charAt(i)).equals(character)) {
        System.out.println("Character found!");
        found = true;
        break;
    }
}
if (found == false)
   System.out.println("Character not found!");
```

- Point to ponder #4:

Inputs:

word = “amazon”

character = “a”

Output?

Character found!

The `break` Statement (19 of 19)

- Good use of the `break` statement. Searching a character in a string …

```
System.out.println("Enter the word");
String word = scanner.nextLine(); //read the input until a space
System.out.println("Enter the character to be searched");
String character = scanner.next(); //read the input until the end

boolean found = false;
for(int i = 0; i < word.length(); i++) {

    if (String.valueOf(word.charAt(i)).equals(character)) {
        System.out.println("Character found!");
        found = true;
        //break;
    }
}
if (found == false)
   System.out.println("Character not found!");
```

- Point to ponder #5:

Inputs:

word = “amazon”

character = “a”

Output?

Character found!

Character found!

The `continue` Statement (1 of 2)

- The `continue` statement will cause the currently executing iteration of a loop to terminate and the next iteration will begin.
- The `continue` statement will cause the evaluation of the condition in `while` and `for` loops.
- Like the `break` statement, the `continue` statement "should be avoided" because it makes the code hard to read and debug.

The `continue` Statement (2 of 2)

```
for(int x = 1; x <= 5; x++) {
    if(x == 3) {
        continue;
    }
    System.out.print(x + " ");
}
```

- Point to ponder #6: Output?

  1 2 4 5

Deciding which loop to use

- The `while` loop:
  - Pretest loop
  - Use it where you do not want the statements to execute if the condition is false in the beginning.
- The `do-while` loop:
  - Post-test loop
  - Use it where you want the statements to execute at least one time.
- The `for` loop:
  - Pretest loop
  - Use it where there is some type of counting variable that can be evaluated.

Generating Random Numbers (1 of 4)

- Some applications, such as games, simulations, statistics, and security require the use of randomly generated numbers.
- The Java API has a class, `Random`, for this purpose. To use the `Random` class, use the following `import` statement and create an instance of the class.

```
import java.util.Random;
Random randomNumbers = new Random();
```

Generating Random Numbers (2 of 4)

Method	Description
`nextDouble()`	Returns the next random number as a `double`. The number will be within the range of 0.0 and 1.0.
`nextFloat()`	Returns the next random number as a `float`. The number will be within the range of 0.0 and 1.0.
`nextInt()`	Returns the next random number as an `int`. The number will be within the range of an `int`, which is –2,147,483,648 to +2,147,483,648.
`nextInt(int n)`	This method accepts an integer argument, n. It returns a random number as an `int` that will be within the range of 0 to n-1.
`nextLong()`	Returns the next random number as a `long`. The number will be within the range of a `long`, which is –9,223,372,036,854,775,808 to +9,223,372,036,854,775,808

Generating Random Numbers (3 of 4)

Example:

```
 // Create a Random object to generate random numbers.
 Random rand = new Random();

String again = "y";
int die1, die2;
while (again.equalsIgnoreCase("y"))
{
  System.out.println("Rolling the dice ...");
  die1 = rand.nextInt(6) + 1;
  die2 = rand.nextInt(6) + 1;
  System.out.println("Their values are:");
  System.out.println(die1 + " " + die2);

  System.out.print("Roll them again (y = yes)? ");
  again = keyboard.nextLine();
}
```

Generating Random Numbers (4 of 4)

- Point to ponder #7:

  Possible range of the output?

```
System.out.println(new Random().nextInt(7) + 3);
```
[3,9]

```
System.out.println(new Random().nextInt(7) + -8);
```
[-8,-2]

```
System.out.println(new Random().nextInt(1) + 3);
```
[3,3]

```
System.out.println(new Random().nextInt(0) + 3);
```
Exception!