Flow of Control

Chapter 5

Objectives

You will be able to:

- Use the Java "if" statement to control flow of control within your program.
- Use the Java relational operators
  - ==, !=, <, >, <=, and >=
- Compare strings for equality using the string method **`equals`**.
- Use repetition statements to create loops in your Java programs

Flow of Control

- The order of statement execution within a program is called the *flow of control.*
- Unless specified otherwise, the order of statement execution through a method is linear: one statement after another in sequence.
  - Top to bottom within the method.

The "if" Statement

- We can use an if statement to alter the flow of control within a method.
  - Execute a statement only if some condition is true.

Recall the Boolean Type

- A **`boolean`** value represents a true or false condition.
- The reserved words **`true`** and **`false`** are the only valid values for a boolean type.

```
boolean done = false;
```

Boolean Comparison Operators

- Java's provides six *comparison operators* that return boolean results:

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

Example:

```
enrollment >= 75
```

The if Statement

The *if statement* has the following syntax:

`if` is a Java reserved word

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

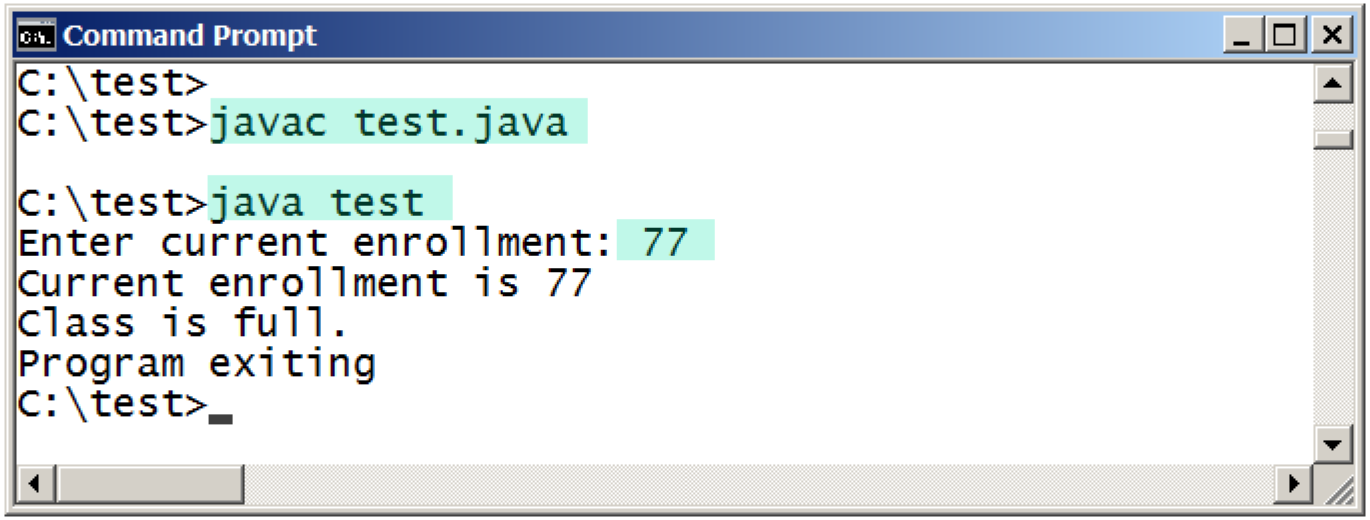
```
if ( condition )
   statement;
```

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

Using the "if" Statement

```
import java.util.Scanner;
class test
{
    public static void main(String[] args)
    {
       Scanner scanner = new Scanner(System.in);
       final int MAX_ENROLLMENT = 75;
       int current_enrollment = 0;

       System.out.print("Enter current enrollment: ");
       current_enrollment = scanner.nextInt();
       System.out.println("Current enrollment is " +
                          current_enrollment);

       if (current_enrollment >= MAX_ENROLLMENT)
           System.out.println("Class is full.");


       System.out.print("Program exiting");
    }
}
```

Using the "if" Statement

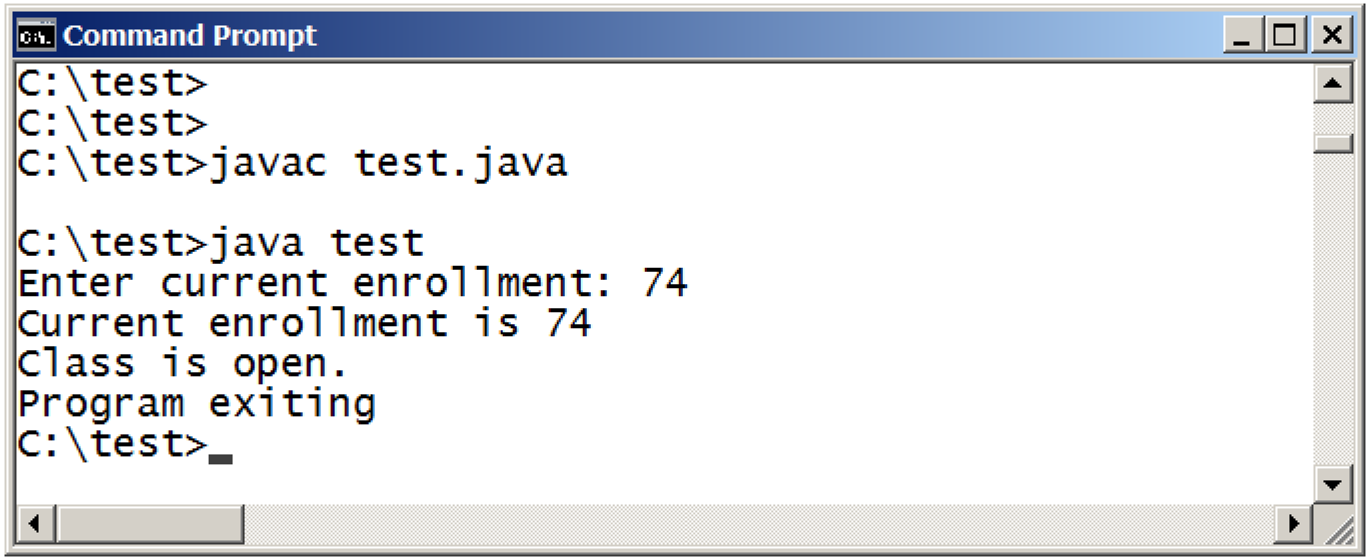

Command Prompt

```
C:\test>
C:\test>javac test.java

C:\test>java test
Enter current enrollment: 77
Current enrollment is 77
Class is full.
Program exiting
C:\test>_
```

An Alternative Action

- We can add the keyword else to provide an action to be taken if the condition is *not* true.

```
if (current_enrollment >= MAX_ENROLLMENT)
    System.out.println("Class is full.");
else
    System.out.println("Class is open.");
```



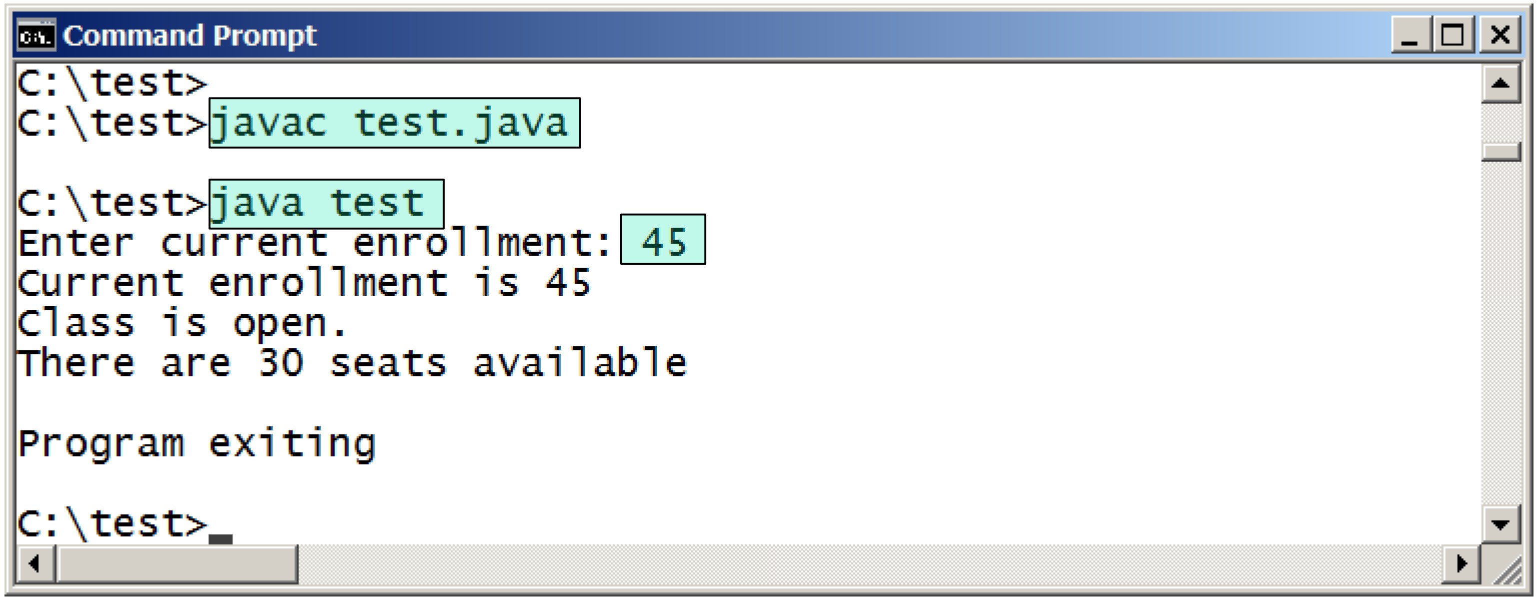
```
C:\test>
C:\test>
C:\test>javac test.java

C:\test>java test
Enter current enrollment: 74
Current enrollment is 74
Class is open.
Program exiting
C:\test>_
```

Block Statements

- What if we need to execute more than one statement when a condition is true (or not)?
- Use curly brackets to define a *block* of statements.
  - if and else treat a block like a single statement.
  - Execute or not execute the enter block according to whether the condition is true.

```
if (current_enrollment >= MAX_ENROLLMENT)
   System.out.println("Class is full.");
else
{
    System.out.println("Class is open.");
    int available = MAX_ENROLLMENT - current_enrollment;
    System.out.println("There are " + available +
                       " seats available");
}
```

Block Statements



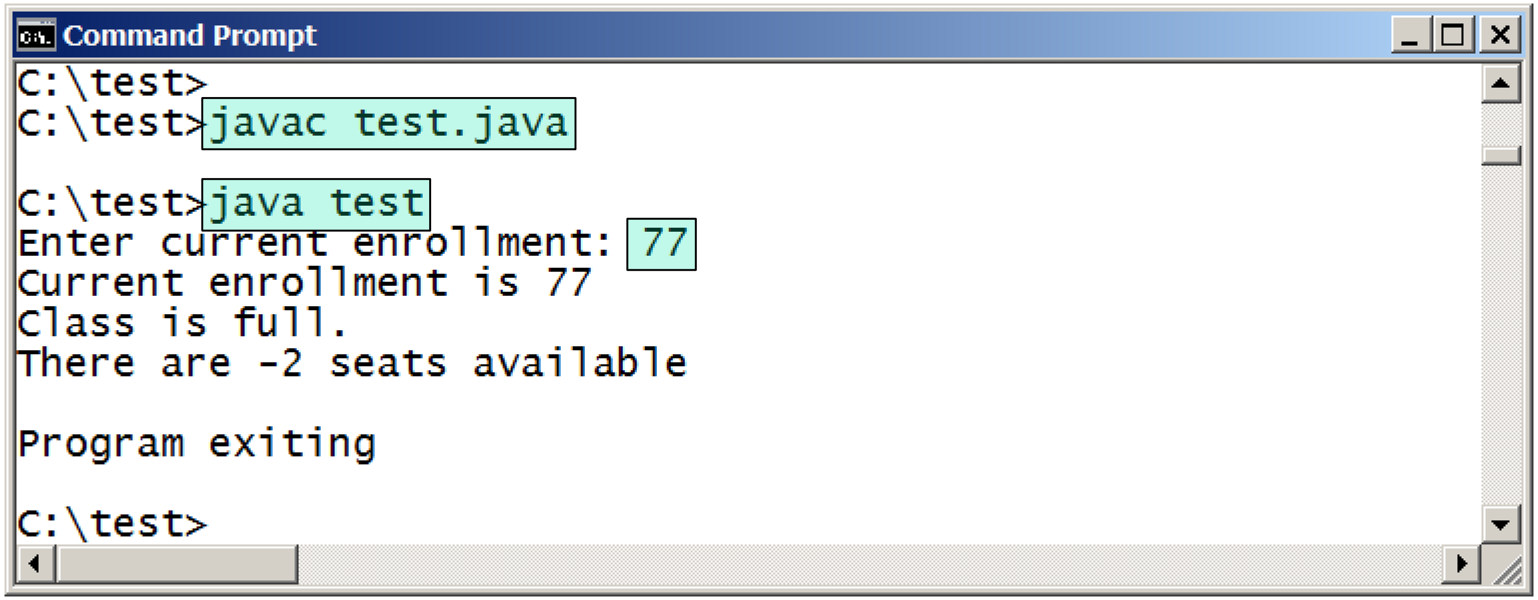
```
C:\test>
C:\test>javac test.java

C:\test>java test
Enter current enrollment: 45
Current enrollment is 45
Class is open.
There are 30 seats available

Program exiting

C:\test>_
```

A Hazard

- What if we forget to add the curly brackets for a block of statements in an if statement?

```
if (current_enrollment >= MAX_ENROLLMENT)
    System.out.println("Class is full.");
else
    System.out.println("Class is open.");
    int available = MAX_ENROLLMENT - current_enrollment;
    System.out.println("There are " + available +
                       " seats available");
```

A Hazard



```
C:\test>
C:\test>javac test.java

C:\test>java test
Enter current enrollment: 77
Current enrollment is 77
Class is full.
There are -2 seats available

Program exiting

C:\test>
```

A Hazard

- What if we forget to add the curly brackets for a block of statements in an if statement?

```
if (current_enrollment >= MAX_ENROLLMENT)
    System.out.println("Class is full.");
else
    System.out.println("Class is open.");

    int available = MAX_ENROLLMENT - current_enrollment;
    System.out.println("There are " + available +
                       " seats available");
```

These two statements are executed in either case.
They are not a part of the if-else statement.

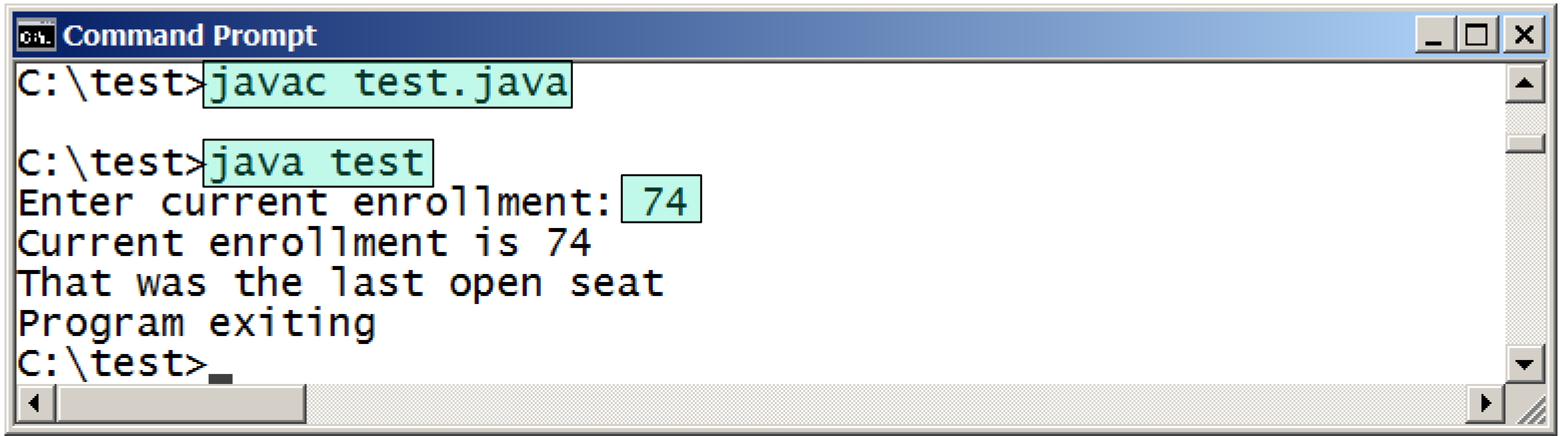
Recommendation

- Always use curly brackets for the conditional parts of an if-else statement.
- Avoid the hazard of forgetting to add the curly brackets if you have only one conditional statement initially and later decide to add some more.

Comparing Integers

- Any of the comparison operators can be used to compare integers.
  - You can put the comparison operator directly into an "if" statement.
  - Example:

```
if (++current_enrollment == MAX_ENROLLMENT)
{
    System.out.println("That was the last open seat");
}
```



Command Prompt

```
C:\test>javac test.java

C:\test>java test
Enter current enrollment: 74
Current enrollment is 74
That was the last open seat
Program exiting
C:\test>_
```

The if Statement

- The precedence of the arithmetic operators is higher than the precedence of the equality and relational operators

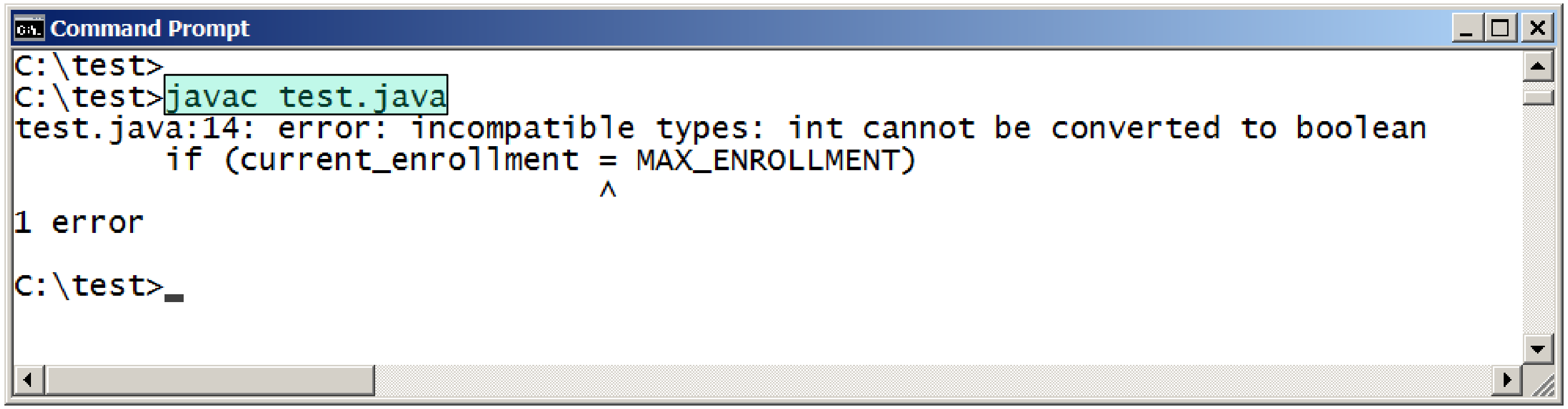
```
if (total != stock + warehouse)
{
   inventoryError = true;
}
```

Sets inventoryError to true if the value of `total` is not equal to the sum of `stock` and `warehouse`

Another Hazard

- Using = instead of ==

```
if (current_enrollment = MAX_ENROLLMENT)
{
    System.out.println("That was the last open seat");
}
```



```
C:\test>
C:\test>javac test.java
test.java:14: error: incompatible types: int cannot be converted to boolean
        if (current_enrollment = MAX_ENROLLMENT)
                               ^
1 error

C:\test>_
```

Fortunately, in Java the compiler catches the error.
(Not always true in other languages!)

Nested if Statements

- The statement executed as a result of an `if` statement or *else* clause could be another `if` statement.

- These are called *nested if statements*

```
if (num1 > num2)
{
    System.out.println ("greater");
}
else
{
    if (num1 == num2)
    {
        System.out.println ("same");
    }
    else
    {
        System.out.println ("less");
    }
}
```

Nested if Statements

- An *else* clause is matched to the last unmatched `if`.
  - No matter what the indentation implies

- Recommendation:
  - Don't do this!
  - Braces can be used to specify the `if` statement to which an *else* clause belongs

Exercise

- Read in three integers from the user and determine the minimum.

Comparing Floating Point Values

- It usually doesn't make sense to compare floating point values for equality.
  - Remember that these are "fuzzy" values.
  - They only compare as equal if they are exactly equal.

- You have to be careful with less than and greater than comparisons.
  - Typically consider floating point values *effectively* equal if they are within some small amount of each other.

Comparing Strings

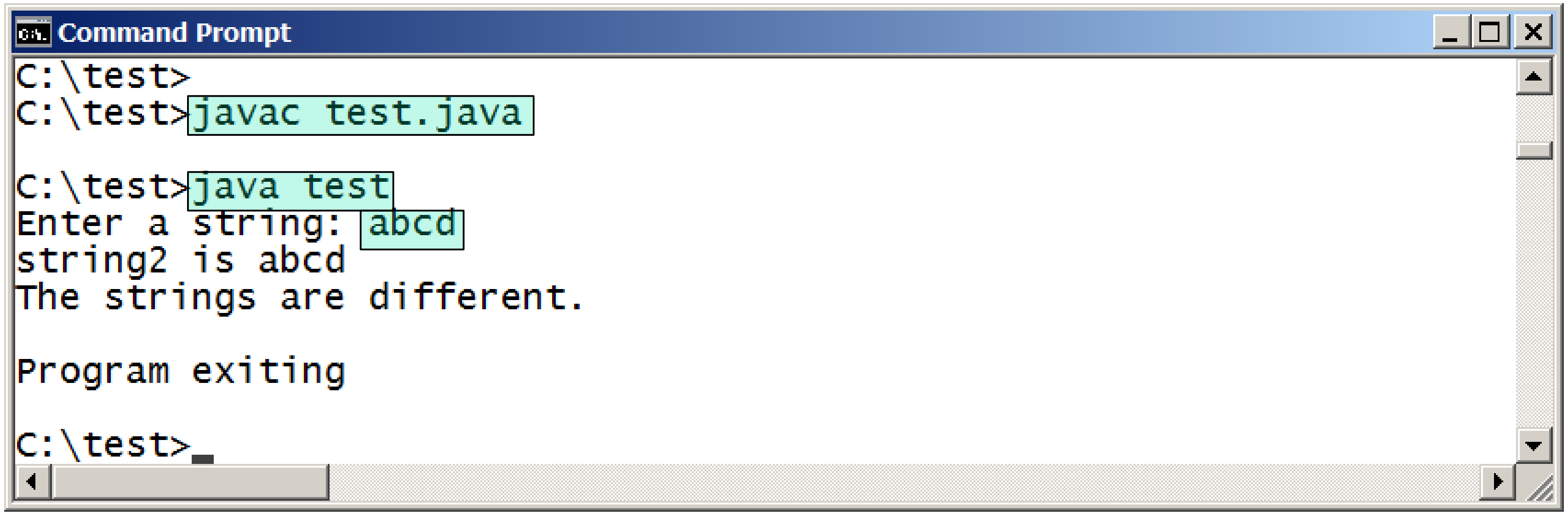
Key Concept

The == operator for Strings tests whether two String references refer to the *same String object*.

- Two different String objects with exactly the same contents will compare as unequal.

```
import java.util.Scanner;
class test
{
    public static void main(String[] args)
    {
       Scanner scanner = new Scanner(System.in);
       String string1, string2;
       System.out.print("Enter a string: ");
       string1 = scanner.nextLine();
       string2 = "abcd";
       System.out.println("string2 is " + string2);
       if (string1 == string2)
       {
           System.out.println("The strings are identical");
       }
       else
       {
           System.out.println("The strings are different.");
       }
       System.out.println("\nProgram exiting");
    }
}
```

Comparing Strings



```
C:\test>
C:\test>javac test.java

C:\test>java test
Enter a string: abcd
string2 is abcd
The strings are different.

Program exiting

C:\test>_
```

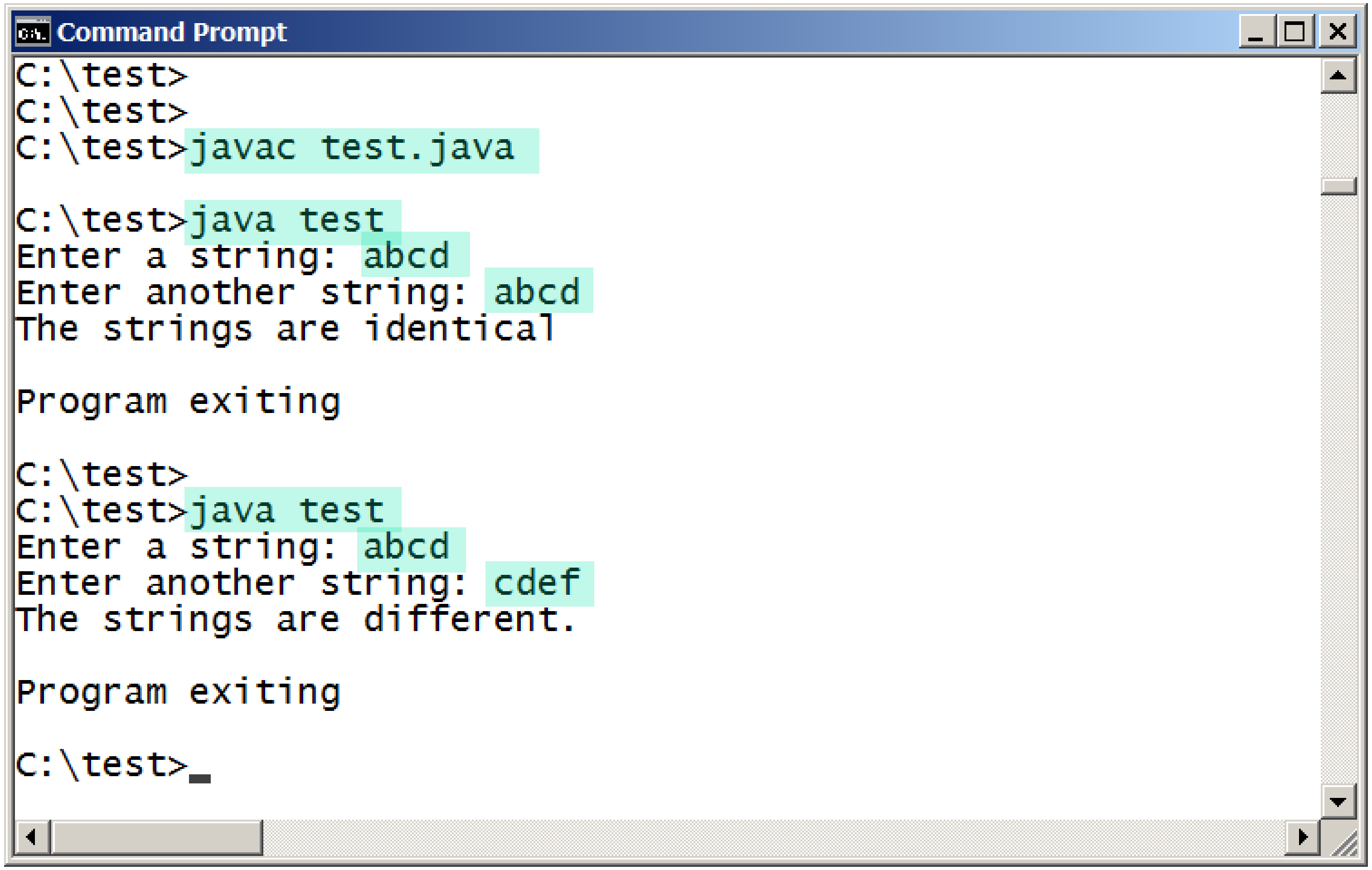
Comparing Strings

- Remember that in Java a character string is an *object.*
- The **`equals`** method can be called with strings to determine if two strings contain exactly the same characters in the same order.
- The **`equals`** method returns a boolean result.

```
if (string1.equals(string2))
{
   System.out.println("The strings are identical");
}
else
{
   System.out.println("The strings are different.");
}
```

```
import java.util.Scanner;
class test
{
    public static void main(String[] args)
    {
       Scanner scanner = new Scanner(System.in);
       String string1, string2;
       System.out.print("Enter a string: ");
       string1 = scanner.nextLine();
       System.out.print("Enter another string: ");
       string2 = scanner.nextLine();

       if (string1.equals(string2))
       {
           System.out.println("The strings are identical");
       }
       else
       {
           System.out.println("The strings are different.");
       }
       System.out.println("\nProgram exiting");
    }
}
```

Comparing Strings



```
C:\test>
C:\test>
C:\test>javac test.java

C:\test>java test
Enter a string: abcd
Enter another string: abcd
The strings are identical

Program exiting

C:\test>
C:\test>java test
Enter a string: abcd
Enter another string: cdef
The strings are different.

Program exiting

C:\test>_
```

Repetition Statements

- *Repetition statements* allow us to execute a statement multiple times.
  - Referred to as *loops*
- 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.

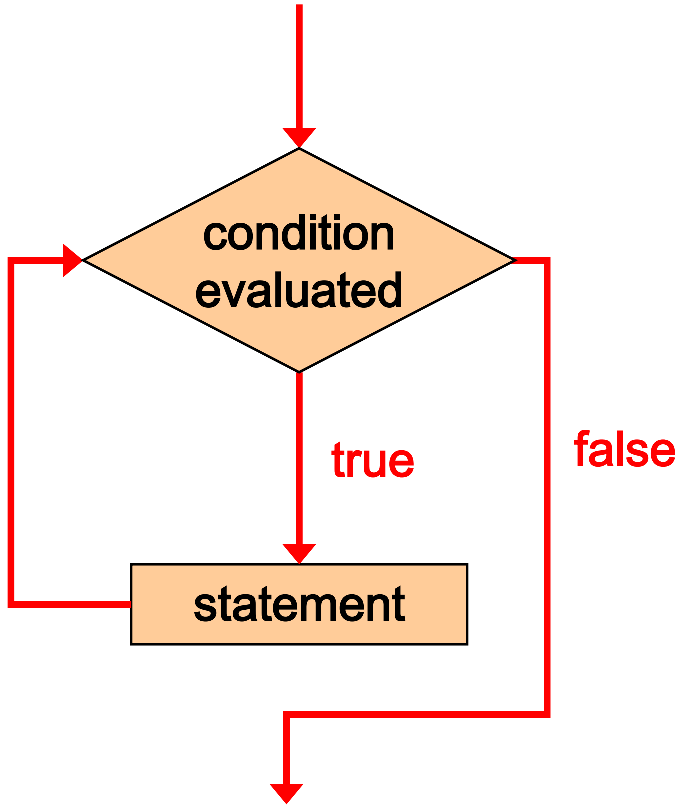
The while Statement

- 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



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

Exercise

- What output is produced by the following code fragment?

```
int num = 0, max = 20;
while (num < max)
{
   System.out.println(num);
   num +=4;
 }
```

```
//********************************************************************
//  Average.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a while loop, a sentinel value, and a
//  running sum.
//********************************************************************

import java.text.DecimalFormat;
import java.util.Scanner;

public class Average
{
   //-----------------------------------------------------------------
   //  Computes the average of a set of values entered by the user.
   //  The running sum is printed as the numbers are entered.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int sum = 0, value, count = 0;
      double average;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter an integer (0 to quit): ");
      value = scan.nextInt();
```

continue

```
continue

      while (value != 0)  // sentinel value of 0 to terminate loop
      {
         count++;

         sum += value;
         System.out.println ("The sum so far is " + sum);

         System.out.print ("Enter an integer (0 to quit): ");
         value = scan.nextInt();
      }

continue
```

continue

```
      System.out.println ();

      if (count == 0)
         System.out.println ("No values were entered.");
      else
      {
         average = (double)sum / count;

         DecimalFormat fmt = new DecimalFormat ("0.###");
         System.out.println ("The average is " + fmt.format(average));
      }
   }
}
```

Exercise: Input Validation

- Modify the Ideal Weight program. If the user enters a number less than 5 for feet, the program should print an error message and keep asking for another input.

- http://www.csee.usf.edu/~turnerr/Programming_Concepts/Downloads/Project_2/Solutions/IdealWeight.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.

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

The do Statement

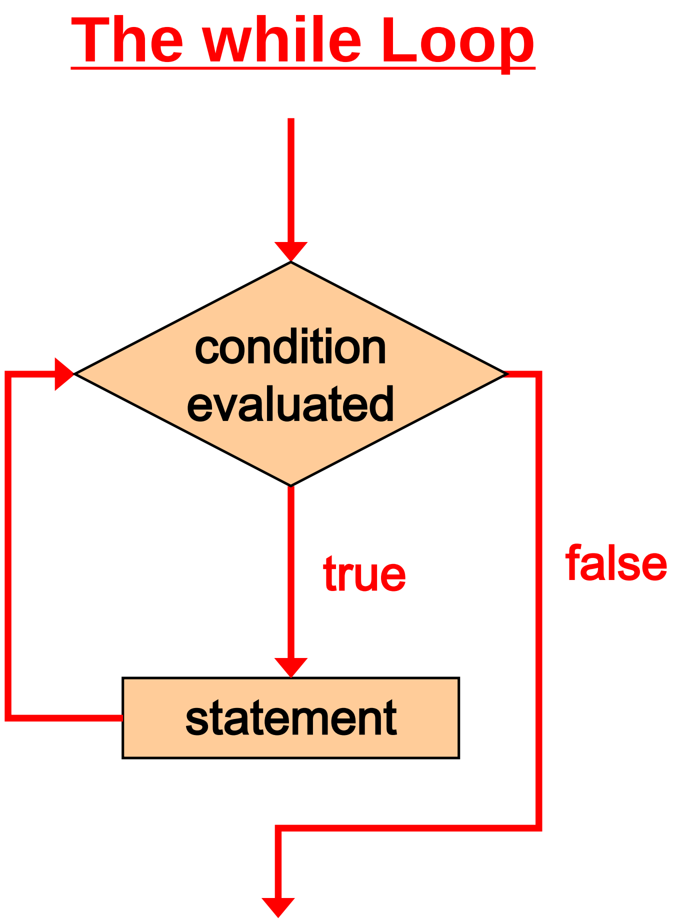
- A *do statement* has the following syntax:

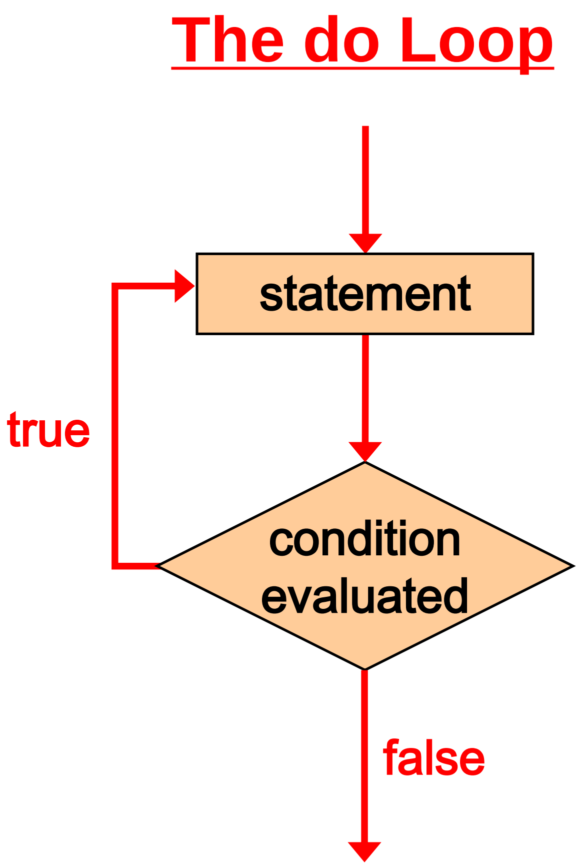
```
do
{
    statement;
}
while ( condition );
```

There is a semicolon

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

Comparing while and do





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**

Readings and Assignments

- Reading: Chapter 5.1 - 5.5
- Self-Assessment Exercises:
  - Do the exercises in this presentation
  - Self-Review Questions
    SR 5.3, 5.4, 5.8, 5.13, 5.14
  - After Chapter Exercises
    EX 5.3, 5.12

- These are not to be submitted in Canvas.
- Check your own answers
  - SR Answers in back of the book
  - EX Write a program if you are not sure.

Readings and Assignments

- Lab Assignment:
  - Project 5: Leap Years
  - Project 6: Going to Extremes

- Projects to be submitted in Canvas.
- Note: There are two separate assignments in Canvas
  - Both due in one week.