

More on Conditionals



Objectives

After completing this lesson, you should be able to:

- Use a `ternary` statement
- Test equality between strings
- Chain an `if/else` statement
- Use a `switch` statement
- Use the NetBeans debugger



Topics

- Relational and conditional operators
- More ways to use `if/else` statements
- Using a `switch` statement
- Using the NetBeans debugger



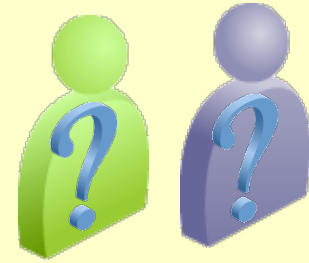
Review: Relational Operators

Condition	Operator	Example
Is equal to	==	<pre>int i=1; (i == 1)</pre>
Is not equal to	!=	<pre>int i=2; (i != 1)</pre>
Is less than	<	<pre>int i=0; (i < 1)</pre>
Is less than or equal to	<=	<pre>int i=1; (i <= 1)</pre>
Is greater than	>	<pre>int i=2; (i > 1)</pre>
Is greater than or equal to	>=	<pre>int i=1; (i >= 1)</pre>

Testing Equality Between String variables

Example:

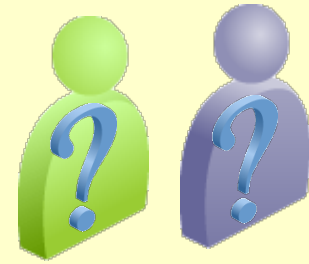
```
public class Employees {  
  
    public String name1 = "Fred Smith";  
    public String name2 = "Sam Smith";  
  
    public void areNamesEqual() {  
        if (name1.equals(name2)) {  
            System.out.println("Same name.");  
        }  
        else {  
            System.out.println("Different name.");  
        }  
    }  
}
```



Testing Equality Between String variables

Example:

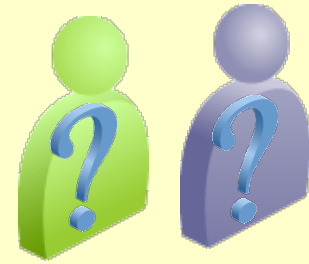
```
public class Employees {  
  
    public String name1 = "Fred Smith";  
    public String name2 = "fred smith";  
  
    public void areNamesEqual() {  
        if (name1.equalsIgnoreCase(name2)) {  
            System.out.println("Same name.");  
        }  
        else {  
            System.out.println("Different name.");  
        }  
    }  
}
```



Testing Equality Between String variables

Example:

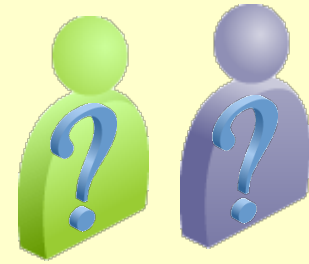
```
public class Employees {  
  
    public String name1 = "Fred Smith";  
    public String name2 = "Fred Smith";  
  
    public void areNamesEqual() {  
        if (name1 == name2) {  
            System.out.println("Same name.");  
        }  
        else {  
            System.out.println("Different name.");  
        }  
    }  
}
```



Testing Equality Between String variables

Example:

```
public class Employees {  
  
    public String name1 = new String("Fred Smith");  
    public String name2 = new String("Fred Smith");  
  
    public void areNamesEqual() {  
        if (name1 == name2) {  
            System.out.println("Same name.");  
        }  
        else {  
            System.out.println("Different name.");  
        }  
    }  
}
```



Common Conditional Operators

Operation	Operator	Example
If one condition AND another condition	&&	<pre>int i = 2; int j = 8; ((i < 1) && (j > 6))</pre>
If either one condition OR another condition		<pre>int i = 2; int j = 8; ((i < 1) (j > 10))</pre>
NOT	!	<pre>int i = 2; (!(i < 3))</pre>

Ternary Conditional Operator

Operation	Operator	Example
If some condition is true, assign the value of <code>value1</code> to the result. Otherwise, assign the value of <code>value2</code> to the result.	<code>?:</code>	<code>condition ? value1 : value2</code> Example: <code>int x = 2, y = 5, z = 0;</code> <code>z = (y < x) ? x : y;</code>

Equivalent statements

```
z = (y < x) ? x : y;
```

```
if (y < x) {  
    z = x;  
}  
else {  
    z = y;  
}
```

Using the Ternary Operator





Advantage: Usable in a single line

```
int numberOfGoals = 1;  
String s = (numberOfGoals==1 ? "goal" : "goals");  
  
System.out.println("I scored " +numberOfGoals + " "  
+s );
```

Advantage: Place the operation directly within an expression

```
int numberOfGoals = 1;  
  
System.out.println("I scored " +numberOfGoals + " "  
+(numberOfGoals==1 ? "goal" : "goals") );
```

Disadvantage: Can have only two potential results

(numberOfGoals==1 ?	"goal" :	"goals" :	"More goals") ;
			
boolean	true	false	???

The diagram illustrates the limitation of the ternary operator by showing its components: a boolean condition, a true result, a false result, and a third, unused result. The third result, "More goals", is crossed out with a large 'X' and labeled with "???", indicating it cannot be utilized.

Topics

- Relational and conditional operators
- **More ways to use `if/else` statements**
- Using a `switch` statement
- Using the NetBeans debugger



Handling Complex Conditions with a Chained `if` Construct

The chained `if` statement:

- Connects multiple conditions together into a single construct
- Often contains nested `if` statements
- Tends to be confusing to read and hard to maintain

Determining the Number of Days in a Month

```
01  if (month == 1 || month == 3 || month == 5 || month == 7
02      || month == 8 || month == 10 || month == 12) {
03      System.out.println("31 days in the month.");
04  }
05  else if (month == 2) {
06      if(!isLeapYear){
07          System.out.println("28 days in the month.");
08      }else System.out.println("29 days in the month.");
09  }
10  else if (month ==4 || month == 6 || month == 9
11          || month == 11) {
12      System.out.println("30 days in the month.");
13  }
14  else
15      System.out.println("Invalid month.");
```

Chaining if/else Constructs

Syntax:

```
01  if <condition1> {  
02      //code_block1  
03  }  
04  else if <condition2> {  
05      // code_block2  
06  }  
07  else {  
08      // default_code  
09  }
```

Topics

- Relational and conditional operators
- More ways to use `if/else` statements
- **Using a `switch` statement**
- Using the NetBeans debugger



Handling Complex Conditions with a `switch` Statement

The `switch` statement:

- Is a streamlined version of chained `if` statements
- Is easier to read and maintain
- Offers better performance

Coding Complex Conditions: switch

```
01 switch (month) {  
02     case 1: case 3: case 5: case 7:  
03     case 8: case 10: case 12:  
04         System.out.println("31 days in the month.");  
05         break;  
06     case 2:  
07         if (!isLeapYear) {  
08             System.out.println("28 days in the month.");  
09         } else  
10             System.out.println("29 days in the month.");  
11         break;  
12     case 4: case 6: case 9: case 11:  
14         System.out.println("30 days in the month.");  
15         break;  
16     default:  
17         System.out.println("Invalid month.");  
18 }
```

switch Statement Syntax

Syntax:

```
01  switch (<variable or expression>) {  
02      case <literal value>:  
03          //code_block1  
04          [break;]  
05      case <literal value>:  
06          // code_block2  
07          [break;]  
08      default:  
09          //default_code  
10 }
```

When to Use `switch` Constructs

Use when you are testing:

- Equality (not a range)
- A *single* value
- Against fixed known values at compile time
- The following data types:
 - Primitive data types: `int`, `short`, `byte`, `char`
 - `String` or `enum` (enumerated types)
 - Wrapper classes (special classes that wrap certain primitive types):
`Integer`, `Short`, `Byte` and `Character`

Only a single variable can be tested.



```
01 switch (month) {  
02     case 1: case 3: case 5: case 7: } Known values  
03     case 8: case 10: case 12:  
04         System.out.println("31 days in the month.");  
05         break;  
06     case 2:  
07         if (!isLeapYear) {  
08             System.out.println("28 days in the month.");  
09         } else  
10             System.out.println("29 days in the month.");
```

Quiz



Which of the following sentences describe a valid case to test in a `switch` construct?

- a. The `switch` construct tests whether values are greater than or less than a single value.
- b. Variable or expression where the expression returns a supported `switch` type.
- c. The `switch` construct can test the value of a `float`, `double`, `boolean`, or `String`.
- d. The `switch` construct tests the outcome of a `boolean` expression.



Topics

- Relational and conditional operators
- More ways to use `if/else` statements
- Using a `switch` statement
- Using the NetBeans debugger



Working with an IDE Debugger

Most IDEs provide a debugger. They are helpful to solve:

- Logic problems
 - (Why am I not getting the result I expect?)
- Runtime errors
 - (Why is there a `NullPointerException`?)

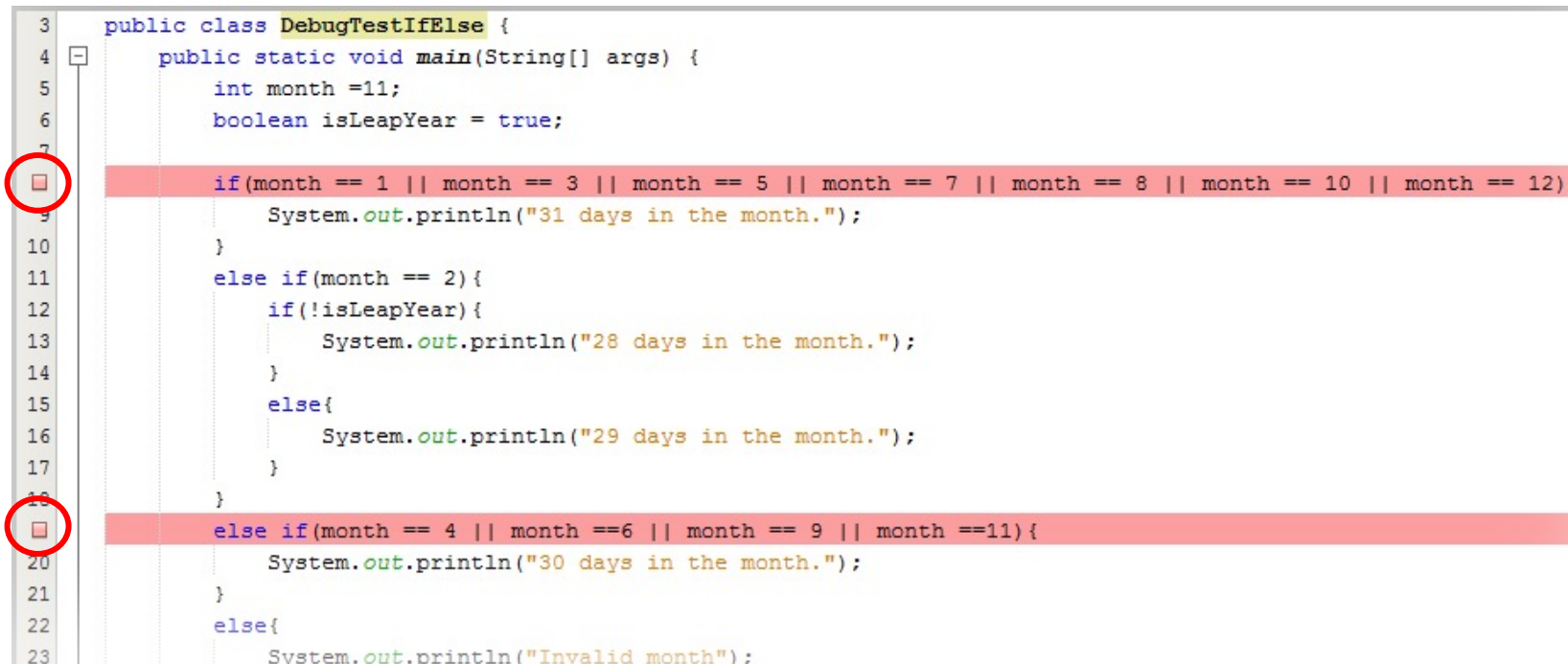


Debugger Basics

- Breakpoints:
 - Are stopping points that you set on a line of code
 - Stop execution at that line so you can view the state of the application
- Stepping through code:
 - After stopping at a break point, you can “walk” through your code, line by line to see how things change.
- Variables:
 - You can view or change the value of a variable at run time.
- Output:
 - You can view the System output at any time.

Setting Breakpoints

- To set breakpoints, click in the margin of a line of code.
- You can set multiple breakpoints in multiple classes.

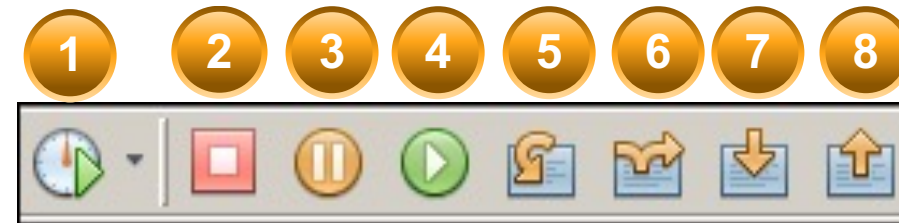


```
3 public class DebugTestIfElse {
4     public static void main(String[] args) {
5         int month = 11;
6         boolean isLeapYear = true;
7
8         if(month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10 || month == 12){
9             System.out.println("31 days in the month.");
10        }
11        else if(month == 2){
12            if(!isLeapYear){
13                System.out.println("28 days in the month.");
14            }
15            else{
16                System.out.println("29 days in the month.");
17            }
18        }
19        else if(month == 4 || month == 6 || month == 9 || month == 11){
20            System.out.println("30 days in the month.");
21        }
22        else{
23            System.out.println("Invalid month");
24        }
25    }
26 }
```

The image shows a code editor with a Java file named `DebugTestIfElse.java`. The code is a `main` method that calculates the number of days in a month based on the month number and whether it's a leap year. Two breakpoints are set on the left margin, indicated by red squares and red circles, on lines 8 and 19. The code is as follows:

The Debug Toolbar

1. Start debugger
2. Stop debug session
3. Pause debug session
4. Continue running
5. Step over
6. Step over an expression
7. Step into
8. Step out of



Viewing Variables

Breakpoint

Current line of execution

```
switch(month) {  
    case 1: case 3: case 5: case 7:  
    case 8: case 10: case 12:  
        System.out.println("31 days in the month.");  
        break;  
    case 2:  
        if(!isLeapYear){  
            System.out.println("28 days in the month.");  
        }  
        else{  
            System.out.println("29 days in the month.");  
        }  
        break;  
    case 4: case 6: case 9: case 11:  
}
```

DebutTestSwitch > main > switch (month) > case 2: >

Output	Watches	Variables %	Breakpoints															
<table border="1"><thead><tr><th>Name</th><th>Type</th><th>Value</th></tr></thead><tbody><tr><td>Static</td><td></td><td>...</td></tr><tr><td>args</td><td>String[]</td><td>#72(length=0)</td></tr><tr><td>month</td><td>int</td><td>2</td></tr><tr><td>isLeapYear</td><td>boolean</td><td>true</td></tr></tbody></table>				Name	Type	Value	Static		...	args	String[]	#72(length=0)	month	int	2	isLeapYear	boolean	true
Name	Type	Value																
Static		...																
args	String[]	#72(length=0)																
month	int	2																
isLeapYear	boolean	true																

Value of variables

Summary

In this lesson, you should have learned how to:

- Use a `ternary` statement
- Test equality between strings
- Chain an `if/else` statement
- Use a `switch` statement
- Use the NetBeans debugger

