Chapter 5

Wednesday, November 4, 2020 11:54 PM

•• 1. Translate the following conditions to Java, where `n` is an integer and `s` is a string.

GOOD JOB! ✓

Condition	Java Code	Explanation
`n` is at least 10.	`n >= 10`	10 is included in this condition.
`n` is negative.	`n < 0`	Zero is not negative.
`n` is zero.	`n == 0`	Remember to use == for equality testing.
`n` is not zero.	`n != 0`	In Java, ≠ is denoted as `!=`.
`s` is `"Hello"`.	`s.equals("Hello")`	Never use == to compare strings.

2 correct, 6 errors, 0%

Start over

•• 2. Give the opposites of the conditions in the table. Assume that `i` and `j` are integers.

GOOD JOB! ✓

Condition	Opposite	Explanation
`i < j`	`i >= j`	The opposite of < is >=. If `i` and `j` are the same, then it isn't true that `i < j`.
`j <= i`	`j > i`	For the same reason, the opposite of <= is >. `i < j` is also a valid answer.
`i == j`	`i != j`	In Java, ≠ is denoted as `!=`.
`i != j`	`i == j`	Remember to use == to test for equality.

3 correct, 3 errors, 0%

Start over

•• 3. Check whether the following conditions evaluate to `true` or `false`, or whether they contain an error. Answer with `true`, `false`, or `error`.

GOOD JOB! ✓

Condition	Outcome	Explanation
`4.35 * 100 == 435`	`false`	Due to roundoff, 4.35 * 100 is 434.99999999999999. It does not usually make sense to use == with floating-point numbers.
`"Hello".substring(0, 1) == "H"`	`false`	The extracted substring is a different object. Use `equals` to compare strings.
`"Hello".substring(0, 2).equals("He")`	`true`	This is the correct way of comparing strings.
`"Hello" == 5`	error	You cannot compare strings with integers.

3 correct, 8 errors, 0%

Start over

•• 4. Which of the following conditions are true, provided `a` is 3 and `b` is 4?

•• 4. Which of the following conditions are true, provided a is 3 and b is 4?

- ✓ True False `a + 1 <= b`
- ✓ True False `a > b - 1`
- ✓ True False `a >= b - 1`
- ✓ True False `a * 2 != b + 2`

4 correct, 2 errors, 50%

Start over

•• 5. Write a program that reads a number and prints whether

- it is zero.
- it is even.
- it has a single digit (positive or negative).

Numbers.java

```
1
2  import java.util.Scanner;
3
4  public class Numbers
5  {
6     public static void main(String[] args)
7     {
8        System.out.print("Enter an integer: ");
9        Scanner in = new Scanner(System.in);
10       int n = in.nextInt();
11       if (n == 0)
12       {
13          System.out.println("The input is zero.");
14       }
15       if (n % 2 == 0)
16       {
17          System.out.println("The input is even.");
18       }
19       if (n%2 == 1)
20       {
21          System.out.println("The input has a single digit.");
22       }
23    }
24 }
```

CodeCheck Reset

Score: 2/4

• 1. In this activity, observe the inputs. They denote hours in "military time" between 0 and 23. For each input, click on the conditions of the `if` statement that are tested and, when a test is successful, on the statement that is executed.

GOOD JOB! ✓

```
int hour = in.nextInt();
if (hour < 6)
{
   time = "night";
}
else if (hour < 12)
{
   time = "morning";
}
else if (hour < 18)
{
   time = "afternoon";
}
else
{
   time = "evening"
}
```

hour	time
~~11~~	~~morning~~
~~15~~	~~afternoon~~
18	evening

8 correct, 4 errors, 50%

Start over

•• 2. In this activity, observe what happens when the alternatives are tested in the wrong order. For each input, click on the conditions of the `if` statement that are tested and, when a test is successful, on the statement that is executed. Follow the actual execution flow, even though it will produce the wrong results.

GOOD JOB! ✓

```
int hour = in.nextInt();
if (hour < 18)
{
   time = "afternoon";
}
else if (hour < 12)
{
   time = "morning"
}
else if (hour < 6)
{
   time = "night";
}
else
{
   time = "evening"
}
```

hour	time
~~11~~	~~afternoon~~
~~15~~	~~afternoon~~
~~5~~	~~afternoon~~
23	evening

6 correct, one error, 83%

• 3. Which of the following fixes the problem that was displayed in the preceding problem?

- ☐ Set `time` to `"morning"` before the `if` statement.
- ☒ Swap the first two tests.
- ☑ Test the threshold values in increasing order.
- ☐ Use `>=` instead of `<` in the tests.

• 1. In this activity, observe the inputs. They denote hours in "military time" between 0 and 23. For each input, click on the conditions of the `if` statement that are tested and, when a test is successful, on the statement that is executed.

GOOD JOB! ✓

```
int hour = in.nextInt();
if (hour < 6)
{
   time = "night";
}
else if (hour < 12)
{
   time = "morning";
}
else if (hour < 18)
{
   time = "afternoon";
}
else
{
   time = "evening"
}
```

hour	time
~~11~~	~~morning~~
~~15~~	~~afternoon~~
18	evening

8 correct, 4 errors, 50%

Start over

•• 2. In this activity, observe what happens when the alternatives are tested in the wrong order. For each input, click on the conditions of the `if` statement that are tested and, when a test is successful, on the statement that is executed. Follow the actual execution flow, even though it will produce the wrong results.

GOOD JOB! ✓

```
int hour = in.nextInt();
if (hour < 18)
{
   time = "afternoon";
}
else if (hour < 12)
{
   time = "morning"
}
else if (hour < 6)
{
   time = "night";
}
else
{
   time = "evening"
```

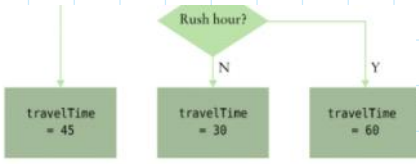
hour	time
~~11~~	~~afternoon~~
~~15~~	~~afternoon~~
~~5~~	~~afternoon~~
23	evening

•• 5. Rearrange these lines of code to yield the color for the lamp of a traffic light, given its position (0 = top, 1 = middle, 2 = bottom). *Hint:* The red light is at the top.

GOOD JOB! ✓

```
if (position == 0)
{
   color = "red";
}
else if (position == 1)
{
   color = "yellow";
}
else
{
   color = "green";
}
```

6 correct, 0 errors, 100%



Which program snippet represents this flow chart?

☐
```
if (rushHour.equals("Y"))
{
   if (route.equals("Bay Bridge"))
   {
      travelTime = 60;
   }
   else
   {
      travelTime = 30;
   }
}
else
{
   travelTime = 45;
}
```

☑
```
if (route.equals("Bay Bridge"))
{
   if (rushHour.equals("Y"))
   {
      travelTime = 60;
   }
   else
   {
      travelTime = 30;
   }
}
else
{
```

•• 1. Consider this code segment for computing income tax, then complete the table below.

```
double income = in.nextDouble();
String maritalStatus = in.next();
if (maritalStatus.equals("s"))  // Condition 1
{
   if (income <= 30000) // Condition 2
   {
      tax = 0.10 * income; // Branch 1
   }
   else
   {
      tax = 3000 + 0.25 * (income - 30000); // Branch 2
   }
}
else
{
   if (income <= 60000) // Condition 3
   {
      tax = 0.10 * income; // Branch 3
   }
   else
   {
      tax = 6000 + 0.25 * (income - 60000); // Branch 4
   }
}
```

GOOD JOB! ✓

Which branch is tested by the input `40000 s` ?	Branch 2	Condition 1 is fulfilled, but condition 2 is not because the input is > 30000.
Which branch is tested by the input `70000 m`?	Branch 4	Both conditions 1 and 3 are not fulfilled, leading to branch 4.
Which branch is tested by the input `30000 s`?	Branch 1	The tested value is at the boundary of condition 2.
Which branch is not tested by any of these test cases?	Branch 3	
Provide a test case for that branch.	30000 m	Any income less than 60000 is a valid answer.
Give a boundary test case for condition 3.	60000 m	60000 is at the boundary between the two branches.

0 correct, 12 errors, 0%

Start over

•• 2. The program segment below has been tested by giving variable `number` the values 777, 1000, and 1035. What additional value should be used in order to achieve coverage of all decision points?

```
if (number > 1000)
{
   if (number % 2 == 0)
   {
      System.out.print("A large even number");
   }
   else
   {
      System.out.print("A large odd number");
   }
}
else
{
   if (number % 2 == 0)
   {
      System.out.print("A small even number");
   }
   else
   {
      System.out.print("A small odd number");
   }
}
```

- ☐ 7
- ☐ 778
- ☒ 1005
- ☑ 1006

•• 1. Assume that `n` is 5 and `k` is 2. What are the values of the following Boolean expressions? Answer true, false, or error.

GOOD JOB! ✓

`0 <= n || n < k`	`true`	The first condition is true.
`0 <= n < k`	error	You need to use two conditions, as in the line
`n >= 0 && k > 0`	`true`	Both conditions are true.
`n && k > 0`	error	You need to use two conditions, as in the line
`5 < n && n < k || k < 10`	`true`	The `&&` operator has a higher precedence than
`!(n <= 5)`	`false`	`n <= 5` is `true`
`!(n < 5)`	`true`	`n < 5` is `false`
`!(0 <= n && n <= k)`	`true`	This condition can be simplified to `0 > n || n` therefore the "or" expression, is `true`.

5 correct, 15 errors, 0%

Start over

•• 2. Suppose `x` and `y` are two integers. Find Boolean expressions for the following conditions.

GOOD JOB! ✓

Condition	Your answer	Explanation
Both `x` and `y` are zero.	`x == 0 && y == 0`	Use the `&&` operator to check whether both con
At least one of `x` and `y` are zero.	`x == 0 || y == 0`	Use the `||` operator to check whether at least
Exactly one of `x` and `y` are zero. Use both `&&` and `||` operators.	`x == 0 && y != 0 || x != 0 && y == 0`	Parentheses are not needed. The `&&` operator bi
Neither `x` nor `y` are zero.	`x != 0 && y != 0`	You can also write this as `!(x == 0 || y == 0)` negation inside (see Special Topic 5.7.)

4 correct, 2 errors, 50%

Start over

•• 1. Assume that the user input

```
12 12.5 $12.50
```

is processed by the sequence of method calls in the first column of the table below, where `in` is a `Scanner` reading the input. Assume each method call is made immediately after the prior one. Fill in the return values of all method calls.

GOOD JOB! ✓

Method call	Return Value	Explanation
`in.hasNextInt()`	`true`	The next input is an integer.
`in.nextInt()`	12	It is safe to read the integer if `hasNextInt` returns true.
`in.hasNextInt()`	`false`	The next input (12.5) is not an integer.
`in.hasNextDouble()`	`true`	The input is a floating-point number.
`in.nextDouble()`	12.5	It is safe to read the number if `hasNextDouble` returns true.
`in.hasNextDouble()`	`false`	The input is not a number because it starts with a `$`.
`in.next()`	`"$12.50"`	The input is a string. To get the dollar value as a number, extract a `Double.parseDouble`.

6 correct, 5 errors, 14%

Start over

•• 2. Your task is to rewrite lines 19–26 of the `ElevatorSimulation2` program so that there is a single `if` statement with a complex condition. What is the condition?

ElevatorSimulation2.java

```
1  import java.util.Scanner;
2
3  /**
4     This program simulates an elevator panel that skips the 13th floor, checking
5     for input errors.
6  */
7  public class ElevatorSimulation2
8  {
9     public static void main(String[] args)
10    {
11       Scanner in = new Scanner(System.in);
```