Chapter 3: Decisions

CS 2070
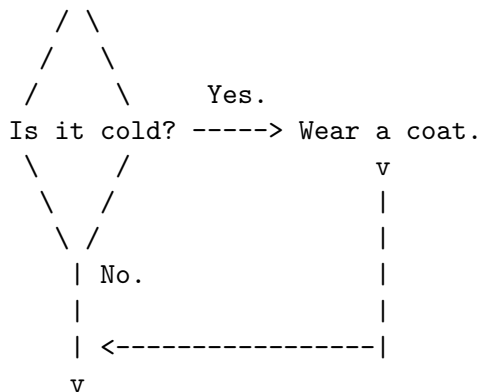
October 11, 2019

If Statements

If Statements

- An if statement will make a decision as to whether it should or should not execute a block of code.

Code.

```
if (boolean expression is true) {
    execute these statements;
}
```

- If the boolean expression in the if condition is false, then the statements are never executed.

If Statements as a flowchart

```
    / \
   /   \
  /     \     Yes.
 Is it cold? -----> Wear a coat.
  \     /                  v
   \   /                   |
    \ /                    |
     | No.                 |
     |                     |
     | <-----------------|
     v
```

Marvel at the beauty of my home made flow charts.

If Statements

There are two forms of the if statement:

```
if (boolean expression)
    statement;
```

Java doesn't require curly brackets if you execute exactly one statement.

```
if (boolean expression) {
    statement;
    statement;
    statement;
}
```

If you only need to execute a single statement, you don't need the curly brackets. However, I always recommend using curly brackets regardless of how many statements you need to execute.

Relational Operators

- Relational operators are used to compare numbers to determine relative order.
  - $a > b$ True if a is greater than b.
  - $a < b$ True if a is less than b.
  - $a >= b$ True if a is greater than or equal to b.
  - $a <= b$ True if a is less than or equal to b.
  - $a == b$ True if a is equal to b.
  - $a != b$ True if a is not equal to b.
- Helpful to remember: There are 4 relational operators comprised of 2 characters. In each case, the = comes second.
- Also helpful: = means "assignment", == means "equals".

Examples

```
if (x > y)
    System.out.println("X is greater than Y");

if (x == y)
    System.out.println("X is equal to Y");

if (x != y) {
    System.out.println("X is not equal to Y");
    x = y;
    System.out.println("However, now it is.");
}
```

Style

Each of these if statements are functionally the same, but only the last example is instructor approved.

```
if (score >= 90) grade = 'A';

if (score >= 90)
    grade = 'A';

if (score >= 90) { grade = 'A'; }

if (score >= 90) {
    grade = 'A';
}
```

Relational Operators and their opposites.

- It's instinctive to think that the opposite of "less than" is "greater than", but this is not correct.
- The opposite of "less than" is "greater than or equal to".
- The opposite of "greater than" is "less than or equal to".
- The opposite of "equals" is "not equals".

Quick Example.

You should pay special attention to the next few examples. They represent simple uses or mistakes that new programmers will make. All four examples represent Java code which may or may not compile.

Quick Example.

What will this code print?

```
int x = 0;
int y = 1;
if (x == y) {
    System.out.println("1 is equal to 0!");
}
```

Quick Example.

What will this code print?

```
int x = 0;
int y = 1;
if (x == y) {
    System.out.println("1 is equal to 0!");
}
```

This code prints nothing to the screen because 1 is not 0.

Quick Example.

What will this code print?

```
int x = 0;
int y = 1;
if (x == 0) {
    System.out.println("x is equal to 0!");
}
```

Quick Example.

What will this code print?

```
int x = 0;
int y = 1;
if (x == 0) {
    System.out.println("x is equal to 0!");
}
```

This code prints "x is equal to 0!" because x does equal 0.

Quick Example.

What will this code print?

```
int x = 0;
int y = 1;
if (x = y) {
    System.out.println("1 is equal to 0!");
}
```

Quick Example.

What will this code print?

```java
int x = 0;
int y = 1;
if (x = y) {
    System.out.println("1 is equal to 0!");
}
```

This code will not compile because "x = y" will not resolve to a boolean expression.

Quick Example.

What will this code print?

```
int x = 0;
int y = 1;
if (y) {
    System.out.println("y is equal to 1!");
}
```

Quick Example.

What will this code print?

```
int x = 0;
int y = 1;
if (y) {
    System.out.println("y is equal to 1!");
}
```

This code will not compile because "y" is not boolean.

Quick Example.

What will this code print?

```java
int x = 0;
int y = 1;
if (x == 1); {
    System.out.println("x is equal to 1!");
}
```

Quick Example.

What will this code print?

```java
int x = 0;
int y = 1;
if (x == 1); {
    System.out.println("x is equal to 1!");
}
```

This code prints "x is equal to 1!". The semicolon represents a full statement. The following curly braces represent an unrelated code block. This is yet another easy mistake to make.

Quick Example.

What will this code print?

```
int x = 0;
int y = 1;
if (x == y)
    System.out.println("x is equal to y!");
    System.out.println("x is equal to 0!");
    System.out.println("y is equal to 1!");
```

Quick Example.

What will this code print?

```
int x = 0;
int y = 1;
if (x == y)
    System.out.println("x is equal to y!");
    System.out.println("x is equal to 0!");
    System.out.println("y is equal to 1!");
```

This code prints "x is equal to 0!" and "y is equal to 1!". Since there is no use of curly brackets, the first statement under the if is associated with the if statement and the last two are not.

if Statement Notes

- ► Do not place ; after (expression).
- ► Be careful testing floats and doubles for equality
- ► Java requires all conditional expressions to resolve to a boolean value of **true** or **false**.

Flags

- A flag is a boolean variable that is used to track for a change in the program.
- First, you set a boolean variable to default status (usually false).
- During the course of the program, you check for a condition, then update the flag to the new status (usually to true).
- Later on, you can then react to the status change.

Flags

```
boolean didAnyoneMakeAnA = false;
...
if (score >= 90) {
    didAnyoneMakeAnA = true;
}
...
if (didAnyoneMakeAnA) {
    System.out.println("Someone made an A!");
}
```

Comparing Characters

- ▶ Characters can be compared using the same relational operators used for numbers.
- ▶ The **char** data type is ordinal, which means every character has an order to it.
  - ▶ 'A' (65) comes before 'Z' (90).
  - ▶ 'a' (97) comes before 'z' (122).
  - ▶ 'Z' (90) comes before 'a' (97).
- ▶ Every character can be translated into a binary UTF-16 number which is used to determine order.
- ▶ If you aren't sure about the order of two characters, check the ASCII table (older standard) or the UTF-16 table (modern standard).

Comparing Characters

Try it.

```java
System.out.print("Enter a character: ");
String line = keyboard.nextLine();
char first = line.charAt(0);
System.out.print("Enter a second character: ");
line = keyboard.nextLine();
char second = line.charAt(0);
if (first < second) {
    System.out.println(first + " comes before " + second);
}
```

Char: Java and C++

- The difference between Java's char and C++'s char is...
  - Java's char is 2 bytes (and uses UTF-16).
  - C++'s char is 1 byte (and uses ASCII).
- The first 128 characters in UTF-16 are identical in ASCII (except being 16 bit instead of 8 bit).

If-Else Statements

If-Else Statements

- The else clause to an if statement adds the ability to conditionally execute code when a condition is false.
- Thank the developers of Java: This is again identical to C++.

Code.

```
if (boolean expression is true) {
    // These Statements are executed.
}
else {
    // These Statements are executed
    // only if the boolean expression is false.
}
```

Nested If Statements

- ▶ Nested if statements are when an if-statement or an if-else-statement appears fully inside of another if-statement.
- ▶ The inner if-statement will only be evaluated if the outer if-statement condition is true. Otherwise it will be ignored completely.
- ▶ In the opinion of your instructor, code which uses nested if-statements are harder to read and understand than code which uses our next construct, so the use of nested-if statements are discouraged.
- ▶ This is like your English teacher teaching about long sentences: Yes, you can write a perfectly valid long sentence, but they are often hard to understand.

Nested If Statements

```java
char first = line.charAt(0);
if ('a' <= first) {
  if ('z' >= first) {
      System.out.println("This is a lower case letter.");
  }
  else {
      System.out.println("This is not a lower case letter.")
  }
}
else {
 System.out.println("This is not a lower case letter.");
}
```

This example requires two identical else clauses to check for lower case letters. I think this difficult to read.

If-Else-If Statements

If-Else-If Statements

- ▶ Because nested if-statements can become complex, I recommend using if-else-if statements as an alternative.
- ▶ After every `else`, continue with another `if` to create chains of if-statements.
- ▶ The last `else` does not require another `if` statement, thus creating a default case.
- ▶ See next slide.

If-Else-If Example

```
int score = keyboard.nextInt();
code grade = 'F';
if (score >= 90)     { grade = 'A'; }
else if (score >= 80) { grade = 'B'; }
else if (score >= 70) { grade = 'C'; }
else if (score >= 60) { grade = 'D'; }
System.out.println("Your grade based on this score is "
                   +grade);
```

After every else, there's another if-statement. This example does
not contain a default case.

Logical Operators

Logical Operators

- **Relational operators** are used to compare numbers.
- **Logical operators** are used to compare boolean expressions or boolean variables.
- Once again, the common logical operators that you remember from C++ appear in the same form in Java.

Logical Operators

- AND (&&): Binary. Evaluates as true if two boolean expressions are both true. False otherwise.
- OR (||): Binary. Evaluates as true if either two boolean expressions are true or both are true. False otherwise.
- NOT (!): Unary. Evaluates as true if an expression is false. Evaluates as false if an expression is true.

AND Truth Table (&&)

- AND (&&): Binary. Evaluates as true if two boolean expressions are both true. False otherwise.
- true && true is true
- false && true is false
- true && false is false
- false && false is false

OR Truth Table (||)

- OR (||): Binary. Evaluates as true if either two boolean expressions are true or both are true. False otherwise.
- true || true is true
- false || true is true
- true || false is true
- false || false is false

NOT Truth Table (!)

- NOT (!): Unary. Evaluates as true if an expression is false. Evaluates as false if an expression is true.
- !true is false
- !false is true

Exclusive Or Truth Table

- "Exclusive OR" is similar to regular OR (||) except that if two boolean expressions are true, the result is false.
- There is no "exclusive or" operator in Java.
    - We can still make our own using a combination of AND, OR, and NOT.
- p is a boolean expression
- q is a boolean expression
- Exclusive OR can be created using this: (p && !q) || (!p && q)

Short Circuiting.

- Java doesn't do any work that it doesn't have to do.
- In an `if` statement, when the boolean expression is false, statements associated with the true branch are not evaluated.
- Likewise, if Java can determine the results of a binary logical expression by looking at only the first boolean expression, it will not evaluate the second boolean expression.
  - This is called "short circuiting".

Short Circuiting.

- The only two binary logical operators are AND (&&) and OR (||).
- If the first expression of AND (&&) is false, the entire expression is false. The second expression isn't touched.
- If the first expression of OR (||) is true, then entire expression is true. The second expression isn't touched.

Short Circuiting: Why?

Look at this example. Assume that doUnsafeOperation will
return a boolean expression of true when it is successful.

```
boolean itIsSafeToDoOperation = ...

if (itIsSafeToDoOperation && doUnsafeOperation()) {
    System.out.println("The unsafe operation was successful
}
```

- ▶ Under what circumstance will doUnsafeOperation be called?
- ▶ Under what circumstance will "The unsafe operation was
  successful." be displayed?

Short Circuiting: Why?

Look at this example. Assume that `doUnsafeOperation` will return a boolean expression.

- Under what circumstance will `doUnsafeOperation` be called?
  - When `itIsSafeToDoOperation` is true. Otherwise it is never called. This is an example of short circuiting.
- Under what circumstance will "The unsafe operation was successful." be displayed?
  - When `itIsSafeToDoOperation` is true AND `doUnsafeOperation()` returns true.

Comparison of String objects

Quick: is `String` a primitive type or a class?

Comparison of String objects

Quick: is `String` a primitive type or a class?

- ▶ It's a class because `String` begins with an upper case letter.
- ▶ All variables which store objects are created dynamically.
  - ▶ The variable reference is placed on the memory stack.
  - ▶ The string data is placed on the memory heap.
  - ▶ The variable reference on the stack points to the data in the memory heap.

Comparison of String : What is happening here?

```
String password = "popcorn";
System.out.print("What is the password? ");
String guess = keyboard.nextLine();

if (password == guess) {
    System.out.println("The passwords match!");
}
```

- This is comparing the reference variables `password` and `guess` and not the actual data.
- In other words, **this code is incorrect**.
- If you test this code, you may find that it still works because of the way Java optimizes String memory.

Comparison of String : Right Way

In this code, we remove the "$==$" and replace it with the method "equals".

```
String password = "popcorn";
System.out.print("What is the password? ");
String guess = keyboard.nextLine();

if (password.equals(guess)) {
    System.out.println("The passwords match!");
}
```

- ► We use a special method built into the **String** class called **equals**.
- ► This method is called from one string and requires a second string as a passed parameter. It will return `true` if two **String** objects match.

Switch Statement

Switch Statement

- The switch statement is used to replace if-else-if statements with shorter code.
- The switch statement can evaluate an integer type or character type variable and make decisions based on the value.
  - You can also use **String** objects in newer versions of Java.
- The switch statement allows you to use an ordinal value to determine how a program will branch.

The switch Statement

The switch statement takes the form:

```
switch (SwitchExpression) {
  case CaseExpression:
    // place one or more statements here
    break;
  case CaseExpression:
    // place one or more statements here
    break;
  default:
    // place one or more statements here
}
```

The switch Statement

The switch statement takes an ordinal value (byte, short, int, long, or char) as the SwitchExpression.

```
switch (SwitchExpression) {
    ...
}
```

The switch Statement

- ▶ If there is an associated case statement that matches that value, program execution will be transferred to that case statement.
- ▶ Each case statement will have a corresponding CaseExpression that must be unique.

Code.

```
case CaseExpression:
    // place one or more statements here
    break;
```

If the SwitchExpression matches the CaseExpression, the Java statements between the colon and the break statement will be executed, even through other case statements.

The case Statement

- ▶ The break statement ends the case statement.
- ▶ The break statement is optional.
- ▶ If a case does not contain a break, then program execution continues into the next case.
- ▶ The default section is optional and will be executed if no CaseExpression matches the SwitchExpression.

Rest of the class.

Write a "fortune cookie" program.

- ▶ A "fortune cookie" is food that has a secret message on a sheet of paper that is suppose to give you good luck.

Code.

```
Random random = new Random();
int value = random.nextInt(10);
```

This code will generate a random value from 0 to 9. Using a switch statement, write a fortune cookie program that will produce one of 10 random messages. You will need to run "Fix imports" within NetBeans or type "import java.util.*;" on the first line of the code.