Winter 2020 CMPE212

CMPE212 – Reminders

• How did the lab go this morning?
• Assignment 1 due a week from tomorrow.

Today

• Continue Basic Java Syntax:
  – Finish Building Expressions. One topic left:
  – Punctuation
  – Conditionals
  – Loops (if we have time)

Conditionals or “Selection Statements”

• We will consider if, if-else and switch statements.
• Simple if statement syntax:

\[
\text{if (boolean expression)} \\
\hspace{1cm} \text{statement when true;}
\]

• Example:

\[
\text{if (capacitance < 0)} \\
\hspace{1cm} \text{System.out.println(“Illegal capacitance”);} \\
\]

Punctuation - Whitespace

• Multiple spaces are treated as one space.
• Leading and trailing spaces are ignored.
• Tabs and empty lines (line feeds) are ignored.
• Line continuation:
  – Long lines can be continued on the line below – break the line anywhere there is a space, but not in the middle of a string (!).
  – For good style, indent the line to show it is a continuation.

Java Punctuation, Cont.

• Comments: inline //, block /* . . . . */
• Comma , used in parameter lists and array literals.
• Semi-colon ; used to end a statement and with for loop syntax.
• Colon : used with switch statements and “for each” loop.
• Double colon :: used with method references.
• “Arrow” -> used with lambda expressions and new switch statements.
• Period or “dot operator” . used with objects to obtain members.
• Also [ ], ( ) and { }

if-else Statement

• Syntax of if-else statement:

\[
\text{if (boolean expression)} \\
\hspace{1cm} \text{statement when true;} \\
\hspace{1cm} \text{else} \\
\hspace{2cm} \text{statement when false;}
\]

• Example:

\[
\text{if (stress > maxStress / 1.5)} \\
\hspace{1cm} \text{result = “failure”;} \\
\hspace{1cm} \text{else} \\
\hspace{2cm} \text{result = “pass”;} \\
\]

Prof. Alan McLeod
if-else Statement, Cont.

- With statement blocks:

```java
if (boolean_expression) {
    block_of_code_when_true
} else {
    block_of_code_when_false
}
```

- Note indentation and bracket alignment.
- You can use {} with a single statement, but it is not necessary.

Abbreviated if Statement

- Uses the "ternary operator" - ?

```java
expression1 ? expression2 : expression3
```

- expression1 must evaluate to a boolean
- expression2 (when true) and expression3 (when false) must evaluate to the same type.
- You could use this with an assignment, for example:
  ```java
  int smaller = a <= b ? a : b;
  ```
  stores the smaller of the two numbers in smaller.

“Chained” if Statements

- Syntax:

```java
if (condition1) { block1 }
else if (condition2) { block2 }
else if (condition3) { block3 }
else if (condition4) { block4 }
else { blocks }
```

- Each condition is tested in turn, until one is evaluated to true. If none of them are true then the else block is executed.

Dangling else Problem

- It is not unusual to nest if statements inside each other.
- One issue that can arise is the “Dangling else” problem.
- See DanglingElse.java
- Indentation might give a visual clue, but has no syntactic meaning.
- The else should be associated with the closest if.
- Use {} if necessary.

switch Statement

- Syntax:

```java
switch (expression) {
    case val1:
        // statements if expression produces val1
        break;
    case val2:
        // statements if expression produces val2
        break;
    case val3:
        ...
    default:
        // statements if none of the above is true
        break;
} // end switch
```

- The code to be run depends on which val# value matches expression.
- If none match, the statements after the default: clause are run.
- The expression and val# values (or "Case Labels") must all be of the same char, integer or String type.
- The break; statements make sure that following cases are not executed after a match has been made.
- It is possible to do multiple cases on one line, but it is clumsy.
**switch Statement - Cont.**

```java
switch (expression) {
    case val1: case val2: case val3:
        // statements if expression is val1, val2 or val3
        break;
    case val4: case val5:
        // statements if expression is val4 or val5
        break;
    case val6:
        default:
            // statements if none of the above is true
            break;
} // end switch
```

- Not too useful a construct.
- Menu coding is a possible use:
  - Provide a number of options to the user, like “(A)dd, (E)dit or (D)elete”.
  - The user presses a, e, d, A, E, D, or some other key.
  - In a switch statement, you would have:

```java
switch (userResponse) { // userResponse is a char
    case 'a': case 'A':
        // Add operation code
        break;
    case 'e': case 'E':
        // Edit operation code
        break;
    case 'd': case 'D':
        // Delete operation code
        break;
    default:
        // Tell user wrong key pressed
        break;
} // end switch
```

**switch Statement with Strings**

- See Switch.java.
- Comparisons are case sensitive. As if `.equals()` is being used.
- Generates more efficient bytecode than what you would get from a chained if construct.

**If You Forget a break; Statement**

- You get what is called a “fall-through” to the code inside the next case statement, even if that case statement would not have been a match.
- Not good!
- The new Java 12 & 13 switch syntax eliminates the fall-through problem and compresses the entire structure.
- See Switch13.java

**Switch in Java 12 & 13**

- Still only works with integer primitive types and char and their equivalent Wrapper classes along with Strings and Enums.
- Use -> (borrowed from lambda expressions) instead of case:
  - No break; statements needed!
  - Multiple case labels (“compounding”) is much tidier. Just separate them with commas.
  - Now can be used as a “switch expression”, which resembles the use of the ternary operator, except with many more conditions.
Switch in Java 12 & 13, Cont.

- We will talk more about Enumerated Types later, but you can always use a switch statement to see if a value matches a value in an Enum.
- In Java 12 & 13, you will get an error if your switch statement does not test for every possible match in the Enum.
- You can have multiple lines of code after the -> using {}. If it is a switch expression in Java 13, use yield to spit out a value from the block.

Repetition or Using “Loops”

- Java has:
  - while
  - do/while
  - for
  - The “for each” loop
- Also, discuss the use of break and continue

“while” loop - Cont.

- while loop syntax:

```java
while (boolean_expression) {
  block_of_code
}
```
- As long as boolean_expression evaluates to true the statements in the block_of_code continue to execute.
- One statement inside the loop does not need {}.
- By mistake, you might write the following - what would happen?

```java
while (boolean_expression);
line_of_code
```

“do/while” loop

- Syntax:

```java
do {
  block_of_code
} while (boolean_expression);
```
- Note the “;” at the end of the while statement.
- Since the conditional test is at the end of the loop, it will always execute the loop at least once.

“for” loop

- The kind of while loop shown below:

```java
int i = 1;
while (i < 21) {
  // other statements
  i = i + 1;
}
```
- is used so often, that Java has provided another looping structure that does all that is shown above, but needs only one line:

```java
for (int i = 1; i < 21; i = i + 1) {
  // other statements
}
```

“for” loop - Cont.

- Syntax:

```java
for (initialisation; boolean_expression; update) {
  block_of_code
}
```
- for loops are used when you know, in advance, the number of repetitions desired.
- If there is only one statement inside the loop you don’t need the {} brackets.
“for” loop - Cont.

- You don’t have to declare the counter inside the for loop, if you have declared it earlier in your program.
- But if you do declare it in the for statement then the scope of that variable will only be inside the loop block.

“for each” Loop

- Often, you will want to visit every element in a collection, not just a part.
- Syntax of the “for each” loop:

```java
for (type_variable : collection) {
    // statements
}
```

“for each” Loop, Cont.

- For example, suppose we have an array called `data`, containing a collection of `double` type numbers, and you want to add them all up:

```java
double sum = 0;
for (double e : data)
    sum = sum + e;
```

“for each” Loop, Cont.

- Equivalent normal for loop:

```java
double sum = 0;
for (int i = 0; i < data.length; i++)
    sum = sum + data[i];
```

- The for each loop is a bit easier with arrays, but is even better suited for other kinds of collections.
- A for each loop does not allow you to change elements in the collection. Use [] and an index value instead in the LHS of an assignment.

Loops - Misc.

- A for each loop can use var for the element type.
- Don’t declare variables inside loops, as the repeated declaration process uses up time and memory unnecessarily.
- There is no limit in Java to how many levels you can nest loops.
- It is customary, but not necessary, to use the variables i, j, k as loop counters, when the counter has no intrinsic meaning.

Other Java Keywords Used With Loops

- `break` and `continue`
- The `continue` statement interrupts the execution of a loop, and returns control to the top of the loop.
- The `break` statement interrupts the execution of a loop, and transfers control to the first statement after the loop.
Use of `continue` 

- For example:
  ```java
  for (i = 1; i <= 5; i++) {
    if (i == 3)
      continue;
    System.out.println("i = " + i);
  }
  System.out.println("End of Loop!");
  
  • Displays:
    - i = 1
    - i = 2
    - i = 4
    - i = 5
    - End of Loop!
  ```

Use of `break` 

- For example:
  ```java
  for (i = 1; i <= 5; i++) {
    if (i == 3)
      break;
    System.out.println("i = " + i);
  }
  System.out.println("End of Loop!");
  
  • Displays:
    - i = 1
    - i = 2
    - End of Loop!
  ```

Use of “`break`” & “`continue`” 

- Only use these keywords when it makes your code easier to read.
- Avoid the use of more than one `break` or `continue` inside a loop.
- If you use a simple condition to issue a `break` statement, then why can’t you put that condition in the loop test?
- Overuse of `break` statements can lead to “spaghetti” code - just like the use of “goto” statements!