6. More Conditionals and Loops

- Objectives - when we have completed this set of notes, you should be familiar with:
  - switch statement
  - the conditional (ternary) operator
  - do-while statement
  - for statement (a.k.a., loop)
  - for-each statement

switch Statement

- Consider the following if statement, where `input` is a `char` value:

```
String answer;
if (input == 't') {
   answer = "true";
}
else if (input == 'f') {
   answer = "false";
}

else {
   answer = "invalid";
}
```

switch Statement

- The switch statement is very similar to the if statement (assume `input` is a `char` and `answer` is a `String`)*:*

```
if (input == 't') {
   answer = "true";
}
else if (input == 'f') {
   answer = "false";
}
else {
   answer = "invalid";
}
```

```
switch(input) {
   case 't':
      answer = "true";
      break;
   case 'f':
      answer = "false";
      break;
   default:
      answer = "invalid";
}
```


switch Statement

- Now that you know the syntax, let's look a little more closely.
  - Expression in the switch is evaluated.
  - Its value is matched to one of the cases. Suppose input is equal to `'f'`… answer will be set to `"false"`
  - The *break* statement breaks out of the switch

```
switch (input) {
   case 't':
      answer = "true";
      break;
   case 'f':
      answer = "false";
      break;
   default:
      answer = "invalid";
}
```

TrueOrFalse.java

switch Statement

Q1

- What happens when there is no break statement? Suppose *input* is `'t'`.
  - It will jump to the appropriate case…
  - And then move to every other case under it until a break or the end of the switch statement. In this case, answer will be invalid even if input is true

```
switch (input) {
    case 't':
        answer = "true";
    case 'f':
        answer = "false";
    default:
        answer = "invalid";
}
```

  - Sometimes it is necessary (think of someone passing through multiple toll booths and getting charged at each one depending on where they started). However, in the example above we probably meant to include breaks.

switch Statement

- When to use a *switch* statement?
  - You need to check to see if one value is equal to others (e.g., you have a lot of == logic)
  - You need put things in categories based on an integral value.
  - Java 6 and earlier: the *switch* statement works on **char**, **byte**, **short**, **int**
  - Java 7 and later: *switch* statement also works on the **wrapper** classes of the types above, as well as **String** and **enum** types

TaxesWithIfElseIf.java TaxesWithSwitch.java

switch Statement

- Why use a switch statement?
  - Depending on the circumstances, it can reduce a code's visual complexity
    - Think of the toll booth example; that would be a messy if statement!
  - A switch statement can jump directly to the correct case, whereas an if-else-if-else has to evaluate each boolean expression until one is true or all are false
    - In other words, using a switch statement can make your program more efficient
  - Example: consider how the OS handles character input from the keyboard

Conditional (Ternary) Operator ___?___ :___

- It's a very concise if-else <u>expression</u>:

  boolean expression ? *do_this_if_true* : *do_this_if_false*

- Examples:
  - Print "Right!" if isCorrect is true, "Wrong." if false.

```
System.out.println(isCorrect ? "Right!" : "Wrong.");
```

  - Subtract *discount* (a double) from *price* (a double) only if *discount* is above 0.

```
double total = (discount > 0) ? (price – discount) : price;
```

  - Print "unit" or "units" with respect to the value of unit

```
System.out.println("Total: " + units + (units == 1 ? " unit":" units"));
```

Conditional (Ternery) Operator

- When to use the ternary operator:
  - It can make a simple if-else statement more concise:

```
if (isCorrect) {
    System.out.println("Right!");
}
else {
    System.out.println("Wrong.");
}
```

can be converted to…

```
System.out.println(isCorrect ? "Right!" : "Wrong.");
```


Conditional (Ternery) Operator

- When not to use the ternary operator:
  - It can make the logic of your code hard to follow.
  - The following method creates a number of small or large chocolate bars based on the amount of chocolate available.

```
public int makeChocolate(int sm, int big, int goal) {

return sm-(goal-(big*5>goal?goal/5:big)*5)>=0?(goal-(big*5>goal?goal/5:big)*5):-1;

}
```

Q2

do-while Statement

- do-while loop:
  - Similar to a while loop, except that the boolean expression is evaluated at the end of the loop (the do-while statement is a *post-test* loop whereas the while statement is a *pre-test* loop).
  - This means the body of the do-while will always be executed at least once, regardless of whether the condition is true.

```
do {
  /* code performed on each iteration */
} while (/* boolean expression */);
```


do-while Statement

Q3

- A good use of a do-while is evaluating user input.
- Suppose the user is entering either a y or n value, and you want to repeat the code until the input is y or n:

```
Scanner stdIn = new Scanner(System.in);
char yOrN = ' ';
do {
   System.out.print("Continue? (enter y or n): ");
   yOrN = stdIn.nextLine().trim().charAt(0);
} while (yOrN != 'y' && yOrN != 'n');
```

YesOrNoInput.java YesOrNoStringInput.java

for Statement

- A.k.a. ***for*** loop - Similar to the while loop, but well-suited for iterating through a loop for predetermined number of times.

```
for (_____; _____; _____) {
  /* code performed on each iteration */
}
```

Initialization - Performed before the first iteration.

Termination - boolean expression checked before each iteration

Increment - Performed after each iteration.

for Statement

- Suppose that you wanted code that would calculate the sum of all numbers from 1 to n. (i.e., 1+2+3+…+n)
  - Initialize a sum to 0.
  - Set up an index to count from 1 to n.
  - On each iteration of the loop…
    - Add the current index to a the sum
    - Increment the index
  - Break out of the loop if the index exceeds n.

for Statement

- Suppose that you wanted code that would calculate the sum of all numbers from 1 to n. (i.e., 1+2+3+…+n)

```
int n = 5;
int sum = 0;
for (int i = 1; i <= n; i++) {
    sum += i;
}
```

AddMultiplyInts.java

for loop vs. while loop

```
// for loop to add 1 to n:
int n = 5;
int sum = 0;
for (int i = 1; i <= n; i++) {
    sum += i;
}

// Equivalent while loop to add 1 to n:
int n = 5;
int sum = 0;
int j = 1;
while (j <= n) {
     sum += j;
     j++;
}
```

for Statement

- Suppose that *list* is an ArrayList holding names of type String, and that you wanted to print out each name. You could use the following code:

```
for (int i = 0; i < list.size(); i++) {
   System.out.println(list.get(i));
}
```

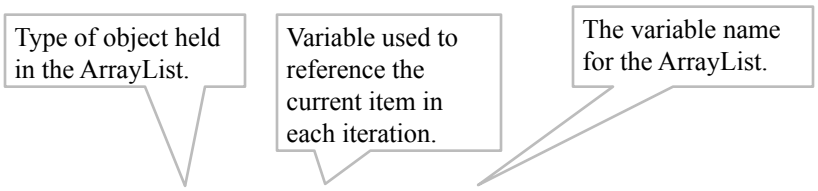

for Statement

- An ArrayList, however, is an Iterable object, meaning that it has a built-in method of iterating through its contents.
- Because of this property, you can use a "for-each" statement (a.k.a. enhanced for loop) to loop through *list*:

```
for (String name : list) {
   System.out.println(name);
}
```

for Statement

- The loop header assigns each String object in order to name. On each iteration, the String object can be accessed using the variable name




```
for (String name : list) {
    System.out.println(name);
}
```

GroupRoster.java

break and *continue*

- A ***break*** statement in a loop will skip the rest of the code in that iteration and exit the loop
- The ***continue*** statement will skip the rest of the code in that iteration and move to the next iteration of the loop
- The *break* and *continue* statements for loops are generally used in conjunction with an *if* statement inside a loop

YesOrNoInput2.java

BreakForExample.java ContinueForExample.java

TriangleListMenuApp

- Displays a menu of options then uses a switch statement to take action based on the user's selection
- Options include:

```
R - Read in File and Create TriangleList
P - Print TriangleList
S - Print Smallest Perimeter
L - Print Largest Perimeter
T - Print Total of Perimeters
A - Add Triangle Object
D - Delete Triangle Object
Q - Quit"
```

TriangleListMenuApp.java

On Your Own

- See the examples in the book (GradeReport, ReverseNumber, Multiples, and Stars).
- Also run EmployeeReviewer.java in the examples folder on your own.
- Review.java and EmployeeReviewer.java contain examples of correct Javadoc documentation for a class.