6. More Conditionals and Loops

- Objectives - when we have completed this chapter, 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

- What happens when there is no break statement? Suppose *input* is now true.
  - 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), but in this case 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 (you have a lot of == logic)
  - You need put things in categories based on an integral value.
  - The switch statement only works on char, byte, short, int. in Java 6 and earlier
    - In Java 7 they now work with String objects as well! Don't use a String in the switch in your projects; we are still using Java 6 in Web-CAT for our Mac users.

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 has to evaluate the if and every else if until the appropriate condition is met.
    - 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 like a very concise if-else. Syntax:

  boolean expression ? d*o_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 much 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 is 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;

}
```

Q

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

Q

- 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().charAt(0);
} while (yOrN != 'y' && yOrN != 'n');
```

YesOrNoInput.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 */
}
```

Performed before the first iteration.

boolean expression

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 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));
}
```

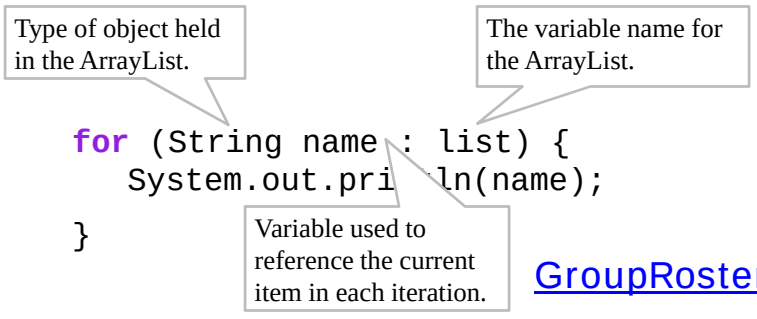
For Statement

- 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 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



GroupRoster.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.