

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 (do-while loop)
 - for statement (for loop)
 - for-each statement (for-each loop)

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
- Note that **case 't':**, **case 'f':**, and **default:** are labels, not executable statements

```
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 it will execute each statement in the switch until a break or the end of the switch statement. In this case, answer will be "invalid" even if input is 't' or 'f'

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

- We probably meant to include breaks here, but consider how to print the remaining days in the week using a “fall through” `switch` (i.e., a `switch` with one or more missing `break` statements)

[FallThroughSwitch.java](#)

switch Statement

- When to use a **switch** statement?
 - When checking to see if a result is equal to different values (i.e., a lot of `==` logic)
 - When you can have alternatives based on an acceptable switch expression type
- Java 6 and earlier: the **switch** statement works on the primitive types: **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 and possibly the logic
 - Think of the "remaining days of week" example with the fall through switch; an *if* statement version would have replicated print statements
 - 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

____?____:____

- Like a concise if-else but an expression:

boolean expression ? do_this_if_true : do_this_if_false

- Examples:

- Print "Right!" or "Wrong." depending on `isCorrect`

```
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`" based on `unit`

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


Conditional (Ternary) 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 (Ternary) Operator

- Conciseness vs. Readability
 - May make the logic of your code hard to follow.
 - The following method returns the number of small bars needed to reach the goal based on small and large chocolate bars available. Having all the logic in a single return expression using multiple ternary operators likely makes the code harder to understand than multiple if statements had been used.

```
public int makeChocolate(int small, int big, int goal) {  
    return small - (goal - (big * 5 > goal ? goal / 5 : big) * 5) >= 0  
        ? (goal - (big * 5 > goal ? goal / 5 : big) * 5) : -1;  
}
```

[MakeChocolateExample.java](#)

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 boolean expression is true

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

***do-while* Statement**

- A good use of a **do-while** is evaluating user input
- Suppose the user is to enter either a y or n, and you want to repeat the request until the input is y or n:

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

[YesOrNoInput.java](#)

[YesOrNoMaxInput.java](#)

for Statement

- **for** loop - Similar to the **while** loop, but well-suited for iterating a specific number of times or over a range of values

```
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., $1+2+3+\dots+n$)
 - 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., $1+2+3+\dots+n$)

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

[AddMultiply.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-each Statement

- An `ArrayList` is an `Iterable` object, which means it can be the target of the "for-each" statement
- "for-each" (a.k.a. enhanced `for` loop) can be used to loop through `list`:

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

- Read the loop header as:
`for` each `String name` in `list` . . .

for-each 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`

Type of object held
in the `ArrayList`

Variable used to
reference the
current item in
each iteration

The variable name
for the `ArrayList`

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

[GroupRoster.java](#)

break and continue

- A **break** statement in a loop immediately exits the loop
- The **continue** statement will skip the rest of the code in that iteration and attempt to do the next iteration of the loop
- Usually the **break** and **continue** statements in loops are used in conjunction with an **if** statement inside a loop

[YesOrNoMaxInput.java](#)

[BreakForExample.java](#)

[ContinueForExample.java](#)

TriangleListMenuApp

- Displays a menu of options then uses a **do-while** loop with 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](#) (in separate folder)