Lecture Four – Branching and Looping

- if/else if
- MyMenu program
- switch/break
- MyNewMenu program
- SumWhile program
- while loops
- SumFor program
- for loops
- continue vs. break

Read Chapter 3
In our last lesson we looked at the ShowOdd program that used the if/else structure.

ShowOdd only accounted for two cases: even and odd. But what if we had more than just two cases?

The if/else if structure can be used when more than two options are needed.

Let’s look at an example in which we display a menu of options for the user to select.
// this code is on two slides for readability

import javax.swing.JOptionPane;
public class MyMenu
{
    public static void main(String[] args)
    {
        String input;    // to hold input from JOptionPane
        int choice;       // to hold the choice from the menu

        input = JOptionPane.showInputDialog(null,
                "Please choose your favorite primary color:
                1. Red
                2. Blue
                3. Green
                4. Yellow");
        choice = Integer.parseInt(input);
    }
}
MyMenu – code (cont.)

if (choice == 1)
    JOptionPane.showMessageDialog(null, "Red is your favorite color!");
else if (choice == 2)
    JOptionPane.showMessageDialog(null, "Blue is your favorite color!");
else if (choice == 3)
    JOptionPane.showMessageDialog(null, "Green is your favorite color!");
else if (choice == 4)
    JOptionPane.showMessageDialog(null, "Yellow is your favorite color!");
else
    JOptionPane.showMessageDialog(null, "You did not enter a valid number!");
}
MyMenu - breakdown

```
public static void main(String[] args)
{
    . . .
    input = JOptionPane.showInputDialog(null,
        "Please choose your favorite primary color:
        1. Red
        2. Blue
        3. Green
        4. Yellow");
    choice = Integer.parseInt(input);
    . . .
}
```

1. This part of the code should be mostly familiar to you. To save space on the slide I omitted the last two arguments of the `JOptionPane` (those arguments are optional). You can add them back in if you wish.

2. Note the line "2. Blue\n3. Green\n4. Yellow"); You are not allowed to break a String between lines, but the string “Please choose your favorite ... \n 4. Yellow”); is too long for a single line. To fix this, break it into two Strings with a ‘+’ between: “first part” + “second part” Then you can put the parts on two or more lines, the compiler concatenates (combines) them into a single String.

3. Also, note the use of \n. Characters preceded with a backslash ( \ ) are special characters called escape characters. \n is known as the newline character. It is used to position text on the next line instead of continuing on the current line. Other common escape characters include: tab (\t), single quote (\'), double quote (\"), and the backslash (\\).
public static void main(String[] args) {
    . . .
    if (choice == 1)
        JOptionPane.showMessageDialog(null, "Red is your favorite color!");
    else if (choice == 2)
        JOptionPane.showMessageDialog(null, "Blue is your favorite color!");
    else if (choice == 3)
        JOptionPane.showMessageDialog(null, "Green is your favorite color!");
    else if (choice == 4)
        JOptionPane.showMessageDialog(null, "Yellow is your favorite color!");
    else
        JOptionPane.showMessageDialog(null, "You did not enter a valid number!");
    . . .
}

4. Here we have a sequence of if/else if statements which allow us to select between a number of alternatives.

5. The final else statement provided a default case when an incorrect option is entered. On the next slide we will look at a common error that occurs with this structure.
6. A common mistake in the **if/else if** structure (called an if/else ladder) is to forget the **if** part of the **else if** statements. A **test expression** can only occur after an **if** keyword, not an **else** keyword.

So, the following would be **INCORRECT!**

```cpp
if (choice == 1)
    . . .
else (choice == 2)    // INCORRECT! a test expression can not
                    // follow an else
    . . .
else
    . . .
```

// INCORRECT! a test expression can not
// following an else
The **if/else ladder** structure works OK, but it is error prone and not very efficient. The **switch** achieves the same purpose in a more efficient way.

Here is the same code written with a **switch**.

```java
// This code replaces the if/else ladder in the earlier version

switch (choice)
{
    case 1: JOptionPane.showMessageDialog(null, "Red is your favorite color!"); break;
    case 2: JOptionPane.showMessageDialog(null, "Blue is your favorite color!"); break;
    case 3: JOptionPane.showMessageDialog(null, "Green is your favorite color!"); break;
    case 4: JOptionPane.showMessageDialog(null, "Yellow is your favorite color!"); break;
    default: JOptionPane.showMessageDialog(null, "You did not enter a valid number!");
}
```
A Closer look at the Switch Code

. . .

switch (choice)
{
    case 1: JOptionPane.showMessageDialog(null, "Red is your favorite color!"); break;
    case 2: JOptionPane.showMessageDialog(null, "Blue is your favorite color!"); break;
    case 3: JOptionPane.showMessageDialog(null, "Green is your favorite color!"); break;
    case 4: JOptionPane.showMessageDialog(null, "Yellow is your favorite color!"); break;
    default: JOptionPane.showMessageDialog(null, "You did not enter a valid number!");
}

1. We begin with the keyword switch followed by the integer or character variable (choice) that will determine which alternative (case) to select.

2. We then create our cases using the case keyword followed by the values that choice can take, and a colon :

3. The final case is labeled with the keyword default and will be executed if choice does not equal any of the other case numbers

4. Each case has one or more statements to be executed when choice is equal to that case number. In our example the only statement executed is the showMessageDialog method.

5. Each case also contains a break statement at the end to terminate execution. If the break is omitted, additional statements will be executed until a break is encountered.
The switch statement takes an integer or a char argument in parentheses and is then followed with a series of cases which are enclosed in braces {}.

Each case contains one or more lines of code that are to be executed when the corresponding case number is entered.

Each case is typically terminated with a break statement to prevent from executing other cases by accident.

There is a special case known as the default case. This case is executed when the argument passed to the switch statement does not correspond with any of the given case numbers.
The SumWhile Program

- Suppose we want to write a program that asks the user to enter a series of numbers and returns the sum of those numbers.

- Each time the user enters a new number, we want the program to add it to a variable called `sum`.

- This requires the program to perform the same work each time a new number is entered. We will use a repetition control structure (while loop) to accomplish this.

- When the user is finished, we can output the value of `sum` which contains the sum of all the numbers entered.

- The code for this program is on the next slide. Go ahead and type the program into Java and get it to run. Then we will talk about it.
import javax.swing.JOptionPane;

public class SumWhile {
    public static void main(String[] args) {
        String input; // to hold input from JOptionPane
        int number; // to hold each number entered
        int count; // loop counter
        int sum = 0; // to keep track of the sum

        count = 0; // initialization

        while (count < 10) // test expression
        {
            input = JOptionPane.showInputDialog(null, "Please enter a number");
            number = Integer.parseInt(input);
            sum += number;
            count++; // increment loop counter
        }

        // output
        JOptionPane.showMessageDialog(null, "Sum of numbers is " + sum);
    }
}
SumWhile - breakdown

public static void main(String[] args) {
    ...
    count = 0;    // initialization
    
    while (count < 10)  // test expression
    {
        input = JOptionPane.showInputDialog(null, "Please enter a number");
        number = Integer.parseInt(input);
        sum += number;
        count++;    // increment loop counter
    }
}

1. This program uses a while loop. The while loop has a test expression in parentheses, followed by a block of executable code, similar to the if statement that we looked at earlier.

2. If the test expression is true, the lines of code in the block (executable code) will be executed. And it will continue to execute this executable code over and over until the test expression is no longer true. Each time this block of code is executed is called an iteration.

3. Note that there must be some code in the block that can change the test expression so that the loop will eventually stop.

4. This particular loop will execute 10 times. Each time, it will ask the user to enter a number and will add that number to the sum variable. The final step in our loop is to increment our loop counter (in this case – count).

5. Note that sum += number is equivalent to saying sum = sum + number (see page 30 of your textbook for more explanation)
while loops

- When writing loops it is important to remember three major steps. In some rare cases (cases you will probably not encounter in this class) some of these steps can be omitted.
  1. **Initialization** – make sure to initialize your loop counter or loop control variable (i.e. the variable being evaluated in your test expression)
  2. **Test Expression** – create a test expression that is logical and capable of eventually being changed from true to false
  3. **Modification of Loop Control Variable** – this is the most important step! Some modification needs to be made to the loop control variable in order to eventually change your test expression from true to false. Otherwise, you may end up with an infinite loop (a loop in which the test expression never evaluates to false, thus causing the code the loop’s block to be executed forever)!
    - In our SumWhile example, if we omitted the count++ at the end of our block then our variable count would always equal 0, thus our test expression (count < 10) would always be true so we would then have an infinite loop!
  - For a **while** loop these steps typically occur in this sequence:

```java
// initialization
while (test expression)
{
    // perform some action
    // modify loop control variable
}
```
The other looping structure that is commonly used is called a **for** loop.

The **for** loop is similar to **while** loop except it combines all three loop steps into a single line of code.

Most programmers recognize that a **for** loop is preferred for a situation where you know in advance how many iterations you want to have, and a **while** loop is ideal for when you want to loop through as many iterations as necessary for something to happen. So...
- to loop 1000 times – use a for loop
- to loop until the error coefficient is less than .0001 – use a while loop

In reality though, which type of loop to use is usually based on the programmer’s preference. There is really no real advantage of using one type of loop over another (in fact, anywhere you have a **while** loop you could put a **for** loop, and anywhere you have a **for** loop you could put a **while** loop)

Let’s take a look at our Sum program, but this time let’s use a **for** loop in place of the **while** loop. In addition, we are going to add a condition that if the user enters a negative number we will stop summing numbers and will display the results.
import javax.swing.JOptionPane;
public class SumFor
{
    public static void main(String[] args)
    {
        String input;     // to hold input from JOptionPane
        int number;      // to hold the number entered
        int count;       // loop counter
        int sum = 0;     // to keep track of the sum

        for (count = 0; count < 10; count++)
        {
            input = JOptionPane.showInputDialog(null, "Please enter a number");
            number = Integer.parseInt(input);
            if (number < 0) break;         // if negative then stop summing numbers
            sum += number;
        }
        JOptionPane.showMessageDialog(null, "Sum of numbers is " + sum);
    }
}
for loops

public static void main(String[] args)
{
    . . .
    for (count = 0; count < 10; count++)
    {
        input = JOptionPane.showInputDialog(null, "Please enter a number");
        number = Integer.parseInt(input);
        if (number < 0) break;  // if negative then stop summing numbers

        sum += number;
    }
    JOptionPane.showMessageDialog(null, "Sum of numbers is " + sum);
}

1. Notice that just like SumWhile we have the three loop steps defined.
2. The difference is that the for loop defines these steps in a for loop header line in a set of parentheses next to the for keyword. Each step is divided by a semicolon.
3. When using for loops all three steps must be defined and must appear in this order: (initialization; test expression; modification)
4. NOTE: the modification part of the for loop gets executed AFTER the block of code following the for loop (i.e. if we were to print out the value of count in the first iteration of our loop it would print 0 not 1)
5. One addition that we made was the condition that if a number is negative we would stop summing numbers and display the result. This can be achieved with the break statement.
6. Once the test expression number < 0 becomes true we execute a break that causes us to leave our loop prematurely
break vs. continue

```java
{  
    input = JOptionPane.showInputDialog(null, "Please enter a number");
    number = Integer.parseInt(input);
    if (number < 0) break; // if negative then stop summing numbers.
    sum += number;
}
```

- When using a `break` statement inside a loop, the `break` statement should always be preceded with a conditional test (if statement). The `break` is going to terminate the loop.

- Sometimes you may have a loop within a loop. In cases like these it is important to remember that a `break` statement will only break out of the inner-most loop (not both loops).

- In our example we stop summing numbers when we encounter a negative value, but what if we just wanted to ignore negative values and continue summing?

- In this case we would want to use a `continue` statement instead of a `break` statement.

- `continue` is much like `break` except instead of breaking out of the entire loop it simply breaks the current iteration of the loop, returns to the control line and continues to iterate.

- Something to keep in mind when using `continue` with a `while` loop is where you modify your loop control variable. It is typically best to modify it before your `continue` to avoid the potential of an infinite loop.
The Do While Loop

- There is one more loop structure. It is called the do while loop.
- You may encounter this in someone else’s code, so you should know that it works like a while loop, except that it checks the test statement after it has executed each iteration instead of before.
- while: if test is true – execute
do while: execute – if test is true, execute again
- Not much to gain, and the do while has some issues that cause errors.
- Recommendation: Avoid the do while loop, use the while and for loops.
Drill Four

- Copy the code from MyNewMenu but remove all `break` statements. Run the program and observe what happens when `break` statements are not present after `case` statements.
- Modify the SumFor program and replace the `break` statement with a `continue` statement. Compare the differences.
- Create a program that asks the user to enter five grades using a `while` loop. Convert those grades to doubles and calculate the average of the five grades. Then, create a variable called “`grade`” that is of type `char`. Using `if/else if` constructs determine the letter grade of the calculated average (i.e. 90-100 = ‘A’, 80-89 = ‘B’, etc.). Finally, create a `switch` that takes the `grade` as an argument (recall how to write case numbers when using chars instead of ints) and displays a message showing the user’s letter grade and average. (yes, you don’t necessarily need a `switch` statement to do this, but it’s good practice :)
- Modify the program from above to use a `for` loop instead of a `while` loop.