CMSC131

The WHILE Loop

Sequential Code

In terms of the code that we've seen up until now, each "unit of code" essentially executes once.

We've seen that there are ways to have conditional execution of code by using an **if** or **if-else** as a guard to specific lines or blocks of code.

We've also seen that if we invoke a method multiple times, then the code within the method is invoked the same number of times, but it's just once per invocation.

Iteration

Sometimes we want some code executed multiple times where the number of iterations is controlled by the data given to the program.

- Sum up all the integers from 1 to 100 (inclusive).
- Ask the user for input until they provide valid input.

There are several types of iteration:

- **while** loop, **do-while** loop, **for** loop, recursion

The WHILE family of loops

Both the **while** and the **do-while** loop structures contain:

- A "body" of the loop (could be one statement, could be a block of statements.
- A Boolean test (condition) which is used to determine whether to perform another iteration of the loop.

The basic idea is that while the condition evaluates to true, the body of the loop will be executed over and over.

- Sum up all the integers from 1 to 100 (inclusive).
- Ask the user to provide a value until they give a valid answer.

WHILE -vs- DO-WHILE

The difference between the **while** and the **do-while** loop structures is the in the **while** loop the condition is tested *before* the body is executed but in the **do-while** loop the condition is tested *after* the body has been executed.

The essential result is that for a **do-while** loop, the body is always executed at least once.

while example

```
int i = 1, sum = 0;
while (i <= 100) {
   sum = sum + i;
   i = i + 1;
}
```

NOTE: We will see this example implemented several different ways.

do-while example

```
int userValue;
Scanner sc = new Scanner(System.in);
do {
   System.out.print(
      "Enter an odd number to continue: ");
   userValue = sc.nextInt();
} while ((userValue%2)==0);
System.out.println("Thank you.");
```

NOTE: If you used a **while** loop you would need to initialize the **userValue** variable with a value that you knew would cause the while loop to execute that first time.

Looking forward…

```
import java.util.Scanner;

public class SimpleDoWhileWithMethod {
   public static void main(String[] args) {
      int userValue;
      Scanner sc = new Scanner(System.in);
      do {
         System.out.print(
            "Enter an odd number to continue: ");
         userValue = sc.nextInt();
      } while (!isOdd(userValue));
      System.out.println("Thank you.");
   }

   public static boolean isOdd (int num) {
      return (!((num%2)==0));
   }
}
```

Will this work the "right" way?

```
do {
  System.out.print("Have you formed a more perfect union?");
  answerHolder = sc.next();

  answerHolder = answerHolder.toLowerCase();

  unionFormed =
     answerHolder == "true" || answerHolder == "yes";

} while (!unionFormed);
```

“Infinite” Loops

One thing that we need to take great care of is making sure we know how iteration ends.

Some things we DON’T want to end (like the running of an operating system perhaps) but usually we have an end trigger in mind.

If you are “stuck” in a loop, you can end your program’s run by clicking on the red square icon in Eclipse.