CMSC131

The WHILE Loop

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

NOTE: We have seen this computational problem before and will see this example implemented several different ways, including a way that many prefer.

Trace this code segment…

```
int counter = 5;
int answer = 0;
while (counter < 4) {
      answer = answer + 1;
      counter = counter - 1;
}
System.out.println(answer);
```

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 (!isOdd(userValue));
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.

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);
```

Out of Scope

When a variable goes "out of scope" its memory should be thought of as being "free" again and it is no longer valid to attempt to access it.

What will this do?

```
do {
    int userResponse;
    System.out.print("Enter an even number: ");
    userResponse = sc.nextInt();
} while ((userResponse%2) != 0);
System.out.println("Thank you.");
```

“Infinite” Loops

We need to take great care of making sure we know how iteration ends. Some programs we might not want to end but usually we have an end trigger in mind.

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