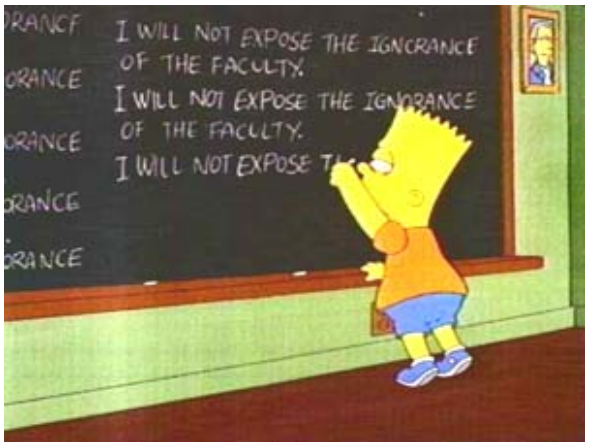
WIT COMP1000

`while` Loops

Loops

- Often, you need to repeat the same computation, action, or sequence of steps many times
- Example: Writing "I will not expose the ignorance of the faculty." 100 times



Loops

- Of course, you *could* use 100 `println()` statements to accomplish this, but that's a lot of copy and pasting work
- Instead, programming languages have control flow mechanisms called *loops* that allow you to loop over (repeat) the same section of code as many times as you need
- Two of the most common types of loops are **`while`** loops and **`for`** loops

`while` Loops

- **`while`** loops are used to repeat a set of Java statements *while* some condition is *true*
- Example:

```
int iteration = 1;
while (iteration <= 100) {
    System.out.println("I will not expose the ignorance of the faculty.");
    iteration = iteration + 1;
}
```

Generic Form of the `while` Loop

```
while (BOOLEAN EXPRESSION) {
   STATEMENT1;
   STATEMENT2;
   …
}
```

Loop body

- The loop body executes over and over as long as the expression is true
  - » Expressions are the same as for `if`/`else if` statements
- Each repetition is called an *iteration* of the loop

Example Behavior of a `while` Loop

```
int input_value;
System.out.print("Enter an integer: ");
input_value = input.nextInt();

while (input_value > 0) {
   System.out.println(input_value);
   input_value = input_value / 2;
}

System.out.println("Done.");
```

Console Output

```
Enter an integer: 7
7
3
1
Done.
```

Current value of `input_value`: ~~?~~ 7 ~~7~~ 3 ~~3~~ 1 ~~1~~ 0

Another Example

```
int i = 1;
int sum = 0;

while (i <= 4) {
   sum = sum + i;
   i = i + 1;
}

System.out.println(sum);
```

Console Output

```
10
```

Current value of `i`: ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

Current value of `sum`: ~~0~~ ~~1~~ ~~3~~ ~~6~~ 10

Wait, `i = i + 1`??

- Yes, that is valid Java!
- Always remember that "=" is NOT a statement of fact, it is a one time assignment of a value to a variable
- For example, if `i` currently has a value of 3, then it will plug that in to the right hand side of the equal sign, add one to get 4, then assign 4 back to the variable `i`
- The same is true for `sum = sum + i`

Notes

- When the program reaches a `while` loop for the first time, it checks the condition
  - » If it is true it begins running the statements inside the loop (in the loop body, between the curly braces)
  - » If the condition is false, it skips past the `while` loop entirely
- When it executes the last statement inside a loop and reaches the }, it goes *back* to the original `while` line and checks the condition again
  - » The condition is only checked when the `while` line itself is executing, not after each statement inside the loop body

Exercise

- Write a program that prints out all the numbers from 0 to N, where N is provided by the user. That is, ask the user for a number then print out all the numbers from 0 to that number, each on their own line.

Answer

```
Scanner input = new Scanner(System.in);

System.out.print("Enter N: ");
int n = input.nextInt();

int i = 0;
while (i <= n) {
   System.out.println(i);
   i = i + 1;
}
```

Infinite Loops

- Always be careful to ensure that your loop conditions will be false eventually
- Loops that have conditions that are always true are *infinite* loops, and are usually a mistake
- You can halt a program stuck in an infinite loop by pressing the terminate button (red square) in the console window

```
int iteration = 1;
while (iteration < 100) {
    System.out.println("This will repeat forever...");
}
```

Increment/Decrement Operators

- Java includes shorthand increment and decrement operators that are often useful with loops (and plenty of other times)
- ++ is the increment operator, used to increase a variable's value by one
  - Example: `count++; // same as count = count + 1;`
- -- is the decrement operator, used to decrease a variable's value by one
  - Example: `i--; // same as i = i - 1;`

do-while Loops

- A **while** loop body might be executed zero times if the condition is never true
- If you need to always execute the body at least once, use a **do-while** loop
- Example:

```
int input_value;
do {
    System.out.print("Enter 1 to print this message again: ");
    input_value = input.nextInt();
} while (input_value == 1);
```

Generic Form of the `do-while` loop

```
do {
   STATEMENT1;
   STATEMENT2;
   …
} while (BOOLEAN EXPRESSION);
```

- Note that you need a semicolon after the `while (EXPRESSION)` in `do-while` loops, but NOT in `while` loops

Example: Sanitizing Inputs

```
double input_value;

do {
    System.out.print("Enter a positive number: ");
    input_value = input.nextDouble();
} while (input_value <= 0);

System.out.printf("The square root is %.3f%n", Math.sqrt(input_value));
```

String Example

```
String input_value;

do {
    System.out.print("Enter y to print this message again: ");
    input_value = input.next();
} while (input_value.equals("y"));
```

Exercise

- Write a program that uses a **`do-while`** loop to read integer values from the user until a value between 1 and 100 (inclusive) is entered

Answer

```
int input_value;

do {
    System.out.print("Enter a number between 1 and 100 (inclusive): ");
    input_value = input.nextInt();
} while (input_value < 1 || input_value > 100);
```

Take Home Points

- Use **`while`** loops to repeat a series of statements so long as some condition is true
- Use **`do-while`** loops if you need to guarantee that the loop body executes at least once
- Be wary of infinite loops
- Use increment/decrement operators as shortcuts to add or subtract one from a variable