Looping/Iterative Structures

Ryan Nixon

Repetition

- While the branching statements allow you to skip code based on a condition, looping statements allow you to repeat code based on a condition
- These constructs are the `while`, `do-while`, `for`, and `foreach` statements.
- We’ll cover everything except `foreach` today and discuss `foreach` when we get to Arrays later in the semester

The Loop

- Looping is not that hard of a concept to learn, but the potential of loops doesn't always come as easily
- Just remember:
    - Loops are for if you need to repeat code until something happens such as user input, a certain point in time, or a certain state of the program occurs
    - Loops are also for if you have a list or other iterable variable and need to execute code against each item in the list
    - Loops always need a reachable end condition. If the condition is not reachable then the loop will never end and your program will execute indefinitely (freeze)

While

-
  ```
  while (condition) {
      action;
  }
  ```

- This is the `while` loop. It will run anything within its scope (the braces) until the boolean condition becomes false

- There is nothing special about the condition. It is exactly like you would see in an `if` statement.

- The same rules apply to if statements as well. You may run anything you want between the braces, including additional nested loops

While

-
  ```
  while (condition) {
       action;
  }
  ```
- A `while` loop will always check the condition first, similar to an `if` statement. If the condition is already false then the entire loop will be skipped
- After all actions in the braces have executed, the `while` loop will return to the condition and check it again. If it has become false then the loop will exit. Otherwise it will run the action again
- Always remember -- In your action code you must ensure that something will cause the condition to become false. Otherwise the loop will never exit.

While

- Let’s jump to an example of this

Do-While

-
  ```
  do {
      action;
  } while (condition);
  ```

- The `do-while` loop is very similar to the `while` loop with one exception: the `do-while` loop always executes at least once

- Java hints at this by placing the condition check at the end of the code. At runtime the program will jump straight to the actions in the braces without caring about the `while` condition. When it reaches the end of the loop it checks the condition and, if the condition is true, returns to the top and executes again

Do-While

- The Latitude/Longitude class shouldn’t require some of the code that we had to write to get the user input. Lets look at how we can improve it with a `do-while`.

For

- For loops are formatted a bit differently than the prior two loops, but they are substantially more powerful
- Think about a for loop like a while loop, because that’s actually exactly what it is. However the for loop makes it easier to define your start and end conditions
- The for loop (and similar construct, the foreach loop) are the most used loops in programming, so make sure you are comfortable with their use.

For

-
  ```
  for(iter-init; condition; iter-action) {
        action;
  }
  ```

- There is much more to set in a for loop to get it working, but what you set is always something you'd have to do anyways with the other loops

- iter-init represents a single line of code to declare a variable. This variable will be used to cause the end condition to become false. Typically this is a simple integer counter:
  ```
  for(int cnt = 0; ...
  ```

- condition is exactly like a while loop's condition:
  ```
  for(int cnt = 0; cnt < 10; ...
  ```

- And iter-action is the code that is run to change the iter-init variable, such as increment or decrement code:
  ```
  for(int cnt = 0; cnt < 10; cnt++) { }
  ```

For

- Back to the examples...

Controlling Loops

- Sometimes you may need exceptions in your loop logic. Perhaps you’re looking for something and don’t need to continue looping once you find it
- Java provides two keywords to control this logic, `continue` & `break`
- `Continue` - Skips the rest of the loop and begins the loop at the next iteration
- `Break` - Exits the loop early regardless of the loop condition

Controlling Loops

-
  ```
  row = 0;
  while(true) {
      row++;
      if (row == 1) continue;
      else if (row >= 10) break;
      else System.out.println(row);
  }
  ```

- The above code would print “23456789”. The if statement skips the first row using continue, exits the loop entirely once the 10th row is reached, and otherwise prints out the current row

- Note that the row++ line here is important. If it was excluded or placed after the if statements then the loop would run indefinitely as it would never increment

Controlling Loops

- When possible, try to avoid using loop control statements. While very useful they are more difficult to read than, say, this:

-
  ```
  row = 0;
  while(row < 10) {
      row++;
      if (row > 1) {
         System.out.println(row);
      }
  }
  ```

Questions?