`while` loops

reading: 5.1

Slides by Chand John.
Used with permission.
https://www.cs.utexas.edu/~chand/cs312/

- In Java, **while** statements follow a general format:

```
while ( <boolean expression> ){
   <statement>
}
```

- For example:

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

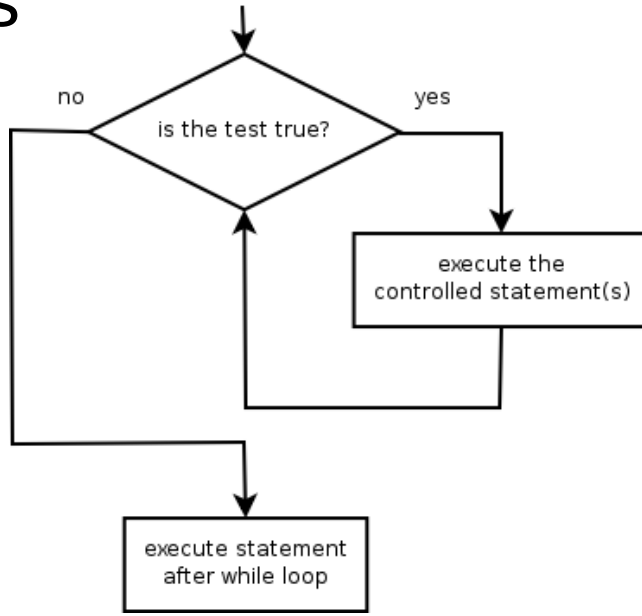
Categories of loops

- **definite loop**: Executes a known number of times.
  - The `for` loops we have seen are definite loops.
    - Print "hello" 10 times.
    - Find all the prime numbers up to an integer *n*.
    - Print each odd number between 5 and 127.
- **indefinite loop**: One where the number of times its body repeats is not known in advance.
    - Prompt the user until they type a non-negative number.
    - Print random numbers until a prime number is printed.
    - Repeat until the user has typed "q" to quit.

The `while` loop

- **`while` loop**: Repeatedly executes its body as long as a logical test is true.

```
while (<test>) {
    <statement(s)>;
}
```



- Example:

```
int num = 1;                    // initialization
while (num <= 200) {              // test
    System.out.print(num + " ");
    num = num * 2;                // update
}

// output:  1 2 4 8 16 32 64 128
```

Example `while` loop

```
// finds the first factor of 91, other than 1
int n = 91;
int factor = 2;
while (n % factor != 0) {
    factor++;
}
System.out.println("First factor is " +
                                   factor);

// output:  First factor is 7
```

- `while` is better than `for` because we don't know how many times we will need to increment to find the factor.

clicker

- What is output by the following code?

```
int x = 1;
int limit = 60;
int val = 1;
while(val < limit) {
    x *= 2;
}
System.out.print(x);
```

A. 1 B. 32 C. 64

D. No output due to syntax error

E. No output due to some other reason

Sentinel values

- **sentinel**: A value that signals the end of user input.
  - **sentinel loop**: Repeats until a sentinel value is seen.
- Example: Write a program that prompts the user for text until the user types nothing, then output the total number of characters typed.
  - (In this case, the *empty* string is the sentinel value.)

```
Type a line (or nothing to exit): hello
Type a line (or nothing to exit): this is a line
Type a line (or nothing to exit):
You typed a total of 19 characters.
```

Solution?

```
Scanner console = new Scanner(System.in);
int sum = 0;
String response = "dummy"; // "dummy" value, anything but ""

while (!response.equals("")) {
    System.out.print("Type a line (or nothing to exit): ");
    response = console.nextLine();
    sum += response.length();
}

System.out.println("You typed a total of " + sum + "
  characters.");
```

Changing the sentinel value

- Modify your program to use `"quit"` as the sentinel value.
  - Example log of execution:

```
Type a line (or "quit" to exit): hello
Type a line (or "quit" to exit): this is a line
Type a line (or "quit" to exit): quit
You typed a total of 19 characters.
```

Changing the sentinel value

- Changing the sentinel's value to `"quit"` does not work!

```
Scanner console = new Scanner(System.in);
int sum = 0;
String response = "dummy"; // "dummy" value, anything but "quit"

while (!response.equals("quit")) {
    System.out.print("Type a line (or \"quit\" to exit): ");
    response = console.nextLine();
    sum += response.length();
}
System.out.println("You typed a total of " + sum + "
  characters.");
```

- This solution produces the wrong output. Why?

```
You typed a total of 23 characters.
```

The problem with the code

- The code uses a pattern like this:

sum = 0.

while (input is not the sentinel) {

prompt for input; read input.

add input length to the sum.

}

problem with code

- On the last pass, the sentinel's length (4) is added to the sum:

  prompt for input; read input (`"quit"`*).*

  add input length (4) to the sum.

- This is a fencepost problem.
  - Must read *N* lines, but only sum the lengths of the first *N*-1.

A fencepost solution

```
sum = 0.
prompt for input; read input.      // place a "post"

while (input is not the sentinel) {
    add input length to the sum.   // place a "wire"
    prompt for input; read input.  // place a "post"
}
```

- Sentinel loops often utilize a fencepost "loop-and-a-half" style solution by pulling some code out of the loop.

Correct code

```
Scanner console = new Scanner(System.in);
int sum = 0;

// pull one prompt/read ("post") out of the loop
System.out.print("Type a line (or \"quit\" to exit): ");
String response = console.nextLine();

while (!response.equals("quit")) {
    sum += response.length();    // moved to top of loop
    System.out.print("Type a line (or \"quit\" to exit):
  ");
    response = console.nextLine();
}

System.out.println("You typed a total of " + sum + "
  characters.");
```

Sentinel as a constant

```
public static final String SENTINEL = "quit";
...

Scanner console = new Scanner(System.in);
int sum = 0;

// pull one prompt/read ("post") out of the loop
System.out.print("Type a line (or \"" + SENTINEL + "\" to exit): ");
String response = console.nextLine();

while (!response.equals(SENTINEL)) {
    sum += response.length();    // moved to top of loop
    System.out.print("Type a line (or \"" + SENTINEL + "\" to exit): ");
    response = console.nextLine();
}

System.out.println("You typed a total of " + sum + " characters.");
```

examples

- write a method to improve checking if a number is prime or not
  - when can we stop?
- Write a program that flips a coin until there is a run of 10 flips of the same side in a row
  - how many flips were there before 10 in a row?
  - repeat the experiment 1000 times, what is the average number of flips
- Flip a coin 100 times. What is the longest run in the 100 flips?