Building Java Programs

Chapter 5
Lecture 5-1: `while` Loops, Fencepost Loops, and Sentinel Loops

reading: 4.1, 5.1

self-check: Ch. 4 #2; Ch. 5 # 1-10

exercises: Ch. 4 #2, 4, 5, 8; Ch. 5 # 1-2

A deceptive problem...

- Write a method `printNumbers` that prints each number from 1 to a given maximum, separated by commas.

  For example, the call:

  ```
  printNumbers(5)
  ```

  should print:

  ```
  1, 2, 3, 4, 5
  ```

Flawed solutions

-
  ```
  public static void printNumbers(int max) {
      for (int i = 1; i <= max; i++) {
          System.out.print(i + ", ");
      }
      System.out.println();   // to end the line of output
  }
  ```

  - Output from `printNumbers(5)`: `1, 2, 3, 4, 5,`

-
  ```
  public static void printNumbers(int max) {
      for (int i = 1; i <= max; i++) {
          System.out.print(", " + i);
      }
      System.out.println();   // to end the line of output
  }
  ```

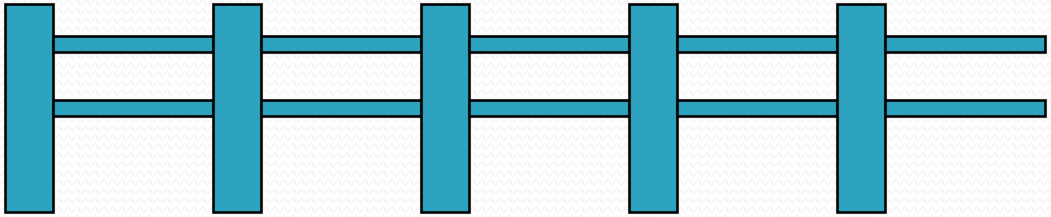
  - Output from `printNumbers(5)`: `, 1, 2, 3, 4, 5`

Fence post analogy

- We print *n* numbers but need only *n* - 1 commas.
- Similar to building a fence with wires separated by posts:
  - If we repeatedly place a post + wire,
    the last post will have an extra dangling wire.

  - A flawed algorithm:

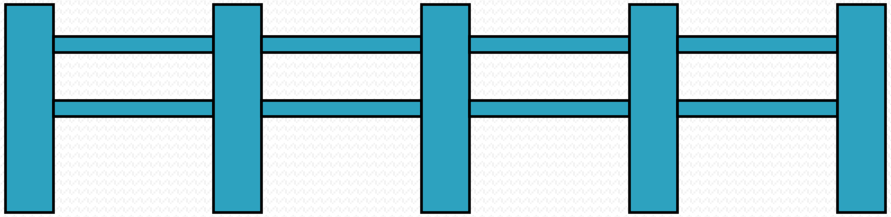
```
for (length of fence) {
    place a post.
    place some wire.
}
```



Fencepost loop

- Add a statement outside the loop to place the initial "post."
  - Also called a *fencepost loop* or a "loop-and-a-half" solution.
  - The revised algorithm:

```
place a post.
for (length of fence - 1) {
    place some wire.
    place a post.
}
```



Fencepost method solution

```
public static void printNumbers(int max) {
    System.out.print(1);
    for (int i = 2; i <= max; i++) {
        System.out.print(", " + i);
    }
    System.out.println();      // to end the line
}
```

- Alternate solution: Either first or last "post" can be taken out:

```
public static void printNumbers(int max) {
    for (int i = 1; i <= max - 1; i++) {
        System.out.print(i + ", ");
    }
    System.out.println(max);  // to end the line
}
```

Fencepost mini-exercises

- Write a method `printRange` that prints all the integers up to a given maximum in the following format:
  - Examples: `printRange(5)` prints

    ```
    [1 2 3 4 5]
    ```

  You can assume that the argument is positive.

- Modify `printRange` so that the argument can be any integer. If the integer is negative or zero just print the brackets:

  `printRange(0)` prints

  ```
  []
  ```

Fencepost mini-exercise solution 1

```
public static void printRange(int max) {
    System.out.print("[1");
    for (int i = 2; i <= max; i++) {
        System.out.print(" " + i);
    }
    System.out.println("]");
}
```

Fencepost mini-exercise solution 2

```
// also support zero and negative arguments
public static void printRange(int max) {
    System.out.print("[");
    if (max>0) {
        System.out.print(1);
    }
    for (int i = 2; i <= max; i++) {
        System.out.print(" " + i);
    }
    System.out.println("]");
}
```

More fencepost questions

- Write a method `printPrimes` that prints all prime numbers up to a given maximum in the following format.
  - Example: `printPrimes(50)` prints

    ```
    [2 3 5 7 11 13 17 19 23 29 31 37 41 43 47]
    ```

- To find primes, write a method `countFactors` which returns the number of factors of an integer.
  - `countFactors(60)` returns `12` because
    1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, and 60 are factors of 60.

Fencepost answer

```
public class Primes {
    public static void main(String[] args) {
        printPrimes(50);
        printPrimes(1000);
    }

    // Prints all prime numbers up to the given max.
    public static void printPrimes(int max) {
        System.out.print("[2");
        for (int i = 3; i <= max; i++) {
            if (countFactors(i) == 2) {
                System.out.print(" " + i);
            }
        }
        System.out.println("]");
    }
```

Fencepost answer, continued

```
    // Returns how many factors the given number has.
    // Note: this is also in ch04-1 slides
    public static int countFactors(int number) {
        int count = 0;
        for (int i = 1; i <= number; i++) {
            if (number % i == 0) {
                count++;  // i is a factor of number
            }
        }
        return count;
    }
}
```

`while` loops

reading: 5.1

self-check: 1 - 10

exercises: 1 - 2

Categories of loops

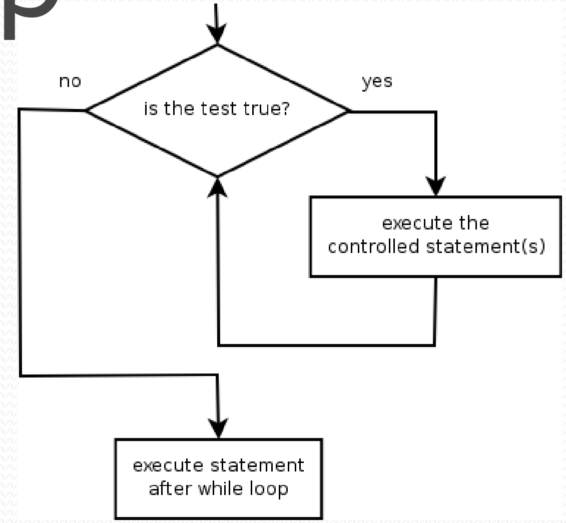
- **definite loop**: Executes a known number of times.
  - The `for` loops we have seen are definite loops.
  - Examples:
    - Print "hello" 10 times.
    - Find all the prime numbers up to an integer *n*.
    - Print each odd number between 5 and 127.

- **indefinite loop**: One where the number of times its body repeats is not known in advance.
  - Examples:
    - Prompt the user until they type a non-negative number.
    - Print random numbers until a prime number is printed.
    - Repeat until the user has types "q" to quit.

The `while` loop

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

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


no
is the test true?
yes
execute the controlled statement(s)
execute statement after while loop

- 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 a number's first factor other than 1
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int number = console.nextInt();
int factor = 2;
while (number % factor != 0) {
    factor++;
}
System.out.println("First factor: " + factor);
```

- Example log of execution:

```
Type a number: 91
First factor: 7
```

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

for vs. while loops

- The for loop is just a specialized form of the while loop.
  - The following loops are equivalent (more or less):

```
for (int num = 1; num <= 200; num = num * 2) {
      System.out.print(num + " ");
}

// actually, not a very compelling use of a while loop
// (a for loop is better because the # of reps is definite)
int num = 1;
while (num <= 200) {
    System.out.print(num + " ");
    num = num * 2;
}
```

Mini-exercise

- Convert the following `for` loop to an almost-equivalent `while` loop:

```
for (int i = 0; i < 10; i++) {
      System.out.println(i);
}
```

Mini-exercise - solution

- Convert the following loop to an equivalent `while` loop:

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

```
int i = 0;
while (i < 10) {
    System.out.println(i);
    i++;
}
```

Mini-exercise part 2

- Puzzler: when we converted this `for` loop to a `while` loop:

```
for (int i = 0; i < 10; i++) {
      System.out.println(i);
}
```

why might the `for` loop not be precisely equivalent to the `while` loop?

Mini-exercise 2 - solution

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}


int i = 0;
while (i < 10) {
    System.out.println(i);
    i++;
}
```

These might not totally equivalent, since the integer `i` is only within the scope of the `for` loop body; but in the `while` loop it is outside the scope of the `while`.

Possible fix: rename `i` to a variable used noplace else.

`while` and `Scanner`

- `while` loops are often used with `Scanner` input.
  - You don't know many times you'll need to re-prompt the user if they type bad data. (an indefinite loop!)

- Write code that repeatedly prompts until the user types a non-negative number, then computes its square root.
  - Example log of execution:

```
Type a non-negative integer: -5
Invalid number, try again: -1
Invalid number, try again: -235
Invalid number, try again: -87
Invalid number, try again: 121
The square root of 121 is 11.0
```

`while` loop answer

```
System.out.print("Type a non-negative integer: ");
int number = console.nextInt();

while (number < 0) {
    System.out.print("Invalid number, try again: ");
    number = console.nextInt();
}

System.out.println("The square root of " + number +
                   " is " + Math.sqrt(number));
```

- Notice that `number` has to be declared outside the loop.

Sentinel loops

reading: 5.1

self-check: 5

exercises: 1, 2

videos: Ch. 5 #4

Sentinel values

- **sentinel**: A value that signals the end of user input.
  - **sentinel loop**: Repeats until a sentinel value is seen.

- Example: A program that repeatedly prompts the user for numbers until the user types -1, then outputs their sum.
  - (In this case, -1 is the sentinel value.)

```
Enter a number (-1 to quit): 10
Enter a number (-1 to quit): 25
Enter a number (-1 to quit): 35
Enter a number (-1 to quit): -1
The sum is 70
```

A second sentinel problem

- Exercise: Write a program that repeatedly prompts the user for words until the user types "goodbye", then outputs the longest word that was typed.
  - (In this case, "goodbye" is the sentinel value.)

```
Type a word (or "goodbye" to quit): Obama
Type a word (or "goodbye" to quit): McCain
Type a word (or "goodbye" to quit): Biden
Type a word (or "goodbye" to quit): Palin
Type a word (or "goodbye" to quit): goodbye
The longest word you typed was "McCain" (6 letters)
```

Flawed sentinel solution

- What's wrong with this solution?

```
Scanner console = new Scanner(System.in);
String longest = "";
String word = "";    // "dummy value"; anything but "goodbye"

while (!word.equals("goodbye")) {
    System.out.print("Type a word (or \"goodbye\" to quit): ");
    word = console.next();
    if (word.length() > longest.length()) {
        longest = word;
    }
}

System.out.println("The longest word you typed was \"" +
        longest + "\" (" + longest.length() + " letters)");
```

- The solution produces the wrong output!

```
The longest word you typed was "goodbye" (7 letters)
```

The problem

- Our code uses a pattern like this:

```
longest = empty string.
while (input is not the sentinel) {
    prompt for input; read input.
    check if input is longest; if so, store it.
}
```

- On the last pass, the sentinel is added to the sum:

```
    prompt for input; read input ("goodbye").
    check if input is longest; if so, store it.
```

- This is a fencepost problem.
  - We must read *N* words, but only process the first *N*-1 of them.

A fencepost solution

- We need to use a pattern like this:

```
longest = empty string.
prompt for input; read input.                    // place 1st "post"

while (input is not the sentinel) {
    check if input is longest; if so, store it.  // place a "wire"
    prompt for input; read input.                // place a "post"
}
```

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

Correct code

- This solution produces the correct output:

```
Scanner console = new Scanner(System.in);
String longest = "";

// moved one "post" out of loop
System.out.print("Type a word (or \"goodbye\" to quit): ");
String word = console.next();

while (!word.equals("goodbye")) {
    if (word.length() > longest.length()) {
        longest = word;     // moved to top of loop
    }
    System.out.print("Type a word (or \"goodbye\" to quit): ");
    word = console.next();
}

System.out.println("The longest word you typed was \"" +
        longest + "\" (" + longest.length() + " letters)");
```

Constant with sentinel

- A better solution uses a constant for the sentinel:

```
public static final String SENTINEL = "goodbye";
```

- This solution uses the constant:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a word (or \"" + SENTINEL + "\" to quit): ");
String word = console.next();
String longest = "";

while (!word.equals(SENTINEL)) {
    if (word.length() > longest.length()) {
        longest = word;      // moved to top of loop
    }
    System.out.print("Type a word (or \"" + SENTINEL + "\" to quit): ");
    word = console.next();
}

System.out.println("The longest word you typed was \"" +
        longest + "\" (" + longest.length() + " letters)");
```

Sentinel number problem

- Solution to the "sum numbers until -1 is typed" problem:

```
Scanner console = new Scanner(System.in);
int sum = 0;
System.out.print("Enter a number (-1 to quit): ");
int number = console.nextInt();

while (number != -1) {
    sum = sum + number;     // moved to top of loop
    System.out.print("Enter a number (-1 to quit): ");
    number = console.nextInt();
}

System.out.println("The sum is " + sum);
```