Fencepost loops

“How do you build a fence?”
The fencepost problem

Problem: Write a class named `PrintNumbers` that reads in an integer called `max` and prints each number from 1 to `max`, separated by commas.

Example:
```
java PrintNumbers
Please enter a maximum integer: 5
```

should print:
```
1, 2, 3, 4, 5
```
A solution?

```java
import java.util.Scanner;
public class PrintNumbers {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int max = keyboard.nextInt();
        for (int i = 1; i <= max; i++) {
            System.out.print(i + " , ");
        }
        System.out.println();  // to end the line
    }
}
```

- **Output when user enters 5:**
  
  1, 2, 3, 4, 5,       // notice extra comma at end!
import java.util.Scanner;
public class PrintNumbers

public static void main(String [] args) {
    Scanner keyboard = new Scanner(System.in);
    int max = keyboard.nextInt();
    for (int i = 1; i <= max; i++) {
        System.out.print(", " + i);
    }
    System.out.println();  // to end the line
}
}

■ Output when user enters 5:
, 1, 2, 3, 4, 5       // comma at beginning
The fencepost problem

- We want to print $n$ numbers but need only $n - 1$ commas.
- Similar to the task of building a fence
  - If we repeatedly place a post and wire, the last post has an extra dangling wire.

- A flawed algorithm:
  ```
  for (length of fence) {
    plant a post.
    attach some wire.
  }
  ```
The solution is to add an extra statement outside the loop that places the initial "post."

This is called a fencepost loop.

The revised algorithm:

plant a post.
for (length of fence - 1) {
    attach some wire.
    plant a post.
}
import java.util.Scanner;
public class PrintNumbers
    public static void main(String [] args) {
        Scanner keyboard = new Scanner(System.in);
        int max = keyboard.nextInt();
        System.out.print(1);
        for (int i = 2; i <= max; i++) {
            System.out.print("", " + i);
        }
        System.out.println();  // to end the line
    }
}

Output when user enters 5:
1, 2, 3, 4, 5  // no extra comma!
Fencepost loop: Exercise

- Write a program that reads a base and a maximum power and prints all of the powers of the given base up to that max, separated by commas.

Base: 2
Max exponent: 9

The first 9 powers of 2 are:
2, 4, 8, 16, 32, 64, 128, 256, 512
Debugging 101
Why won’t it toast?

- You arrive at your dorm after a thought-provoking lecture of CIS 1068. To feed your brain, you put some bread into your toaster oven and set the dial for 5 minutes. The toaster oven ticks away. After five minutes, the toaster oven dings. You take the bread out, but it’s not even toasted. What do you do?
import java.util.*;

public class Buggy {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("How many numbers to average? ");
        int count = console.nextInt();

        int sum = 0;
        for (int i = 1; i <= count; i++) {
            System.out.print("#" + i + ": ");
            sum = console.nextInt();
        }

        System.out.println("The average is: " + (sum / count));
    }
}
Always remember

- `System.out.println` is your friend

- Print out the variables in your program at various places to pinpoint the problem.

Example:

- `System.out.println("x = " + x);`
Sentinel loops
Sentinel values

- **sentinel**: A special value that signals the end of the user's input.
- **sentinel loop**: A loop that repeats until a sentinel value is seen.

Example: Write a program that repeatedly prompts the user for numbers to add until the user types 0, then outputs the sum of the numbers. (In this case, 0 is our sentinel value.)

**Sample run:**

Enter a number (0 to quit): 95
Enter a number (0 to quit): 87
Enter a number (0 to quit): 42
Enter a number (0 to quit): 26
Enter a number (0 to quit): 0
The total was 250
A solution?

```java
Scanner console = new Scanner(System.in);
int sum = 0;
int inputNumber = 1;  // "dummy value", anything but 0

while (inputNumber != 0) {
    System.out.print("Enter a number (0 to quit): ");
    inputNumber = console.nextInt();
    sum += inputNumber;
}

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

- Will this work? Why or why not?
Using a different sentinel value

- Modify your program to use a sentinel value of -1.

**Sample run:**

Enter a number (-1 to quit): 95
Enter a number (-1 to quit): 87
Enter a number (-1 to quit): 42
Enter a number (-1 to quit): 26
Enter a number (-1 to quit): -1
The total was 250
Changing the sentinel value

- **Just change the test value to -1?**
  
  ```java
  Scanner console = new Scanner(System.in);
  int sum = 0;
  int inputNumber = 1;  // "dummy value", anything but -1

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

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

- **Now the solution produces the wrong output! Why?**
  The total was 249
The problem

- The current algorithm:
  \[ \text{sum} = 0. \]
  \[ \text{while input is not the sentinel:} \]
  \[ \quad \text{prompt for input; read input.} \]
  \[ \quad \text{add input to the sum.} \]

- On the last pass through the loop, the sentinel value -1 is added to the sum:
  \[ \text{prompt for input; read input (-1).} \]
  \[ \text{add input (-1) to the sum.} \]

- What kind of problem is this?
  - This is a fencepost problem. We want to read \( N \) numbers (\( N \) is not known ahead of time), but only sum the first \( N - 1 \) of them.
Here is a correct algorithm:

\[ \text{sum} = 0. \]

prompt for input; read input.

while input is not the sentinel:

add input to the sum.

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

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

System.out.println("The total was " + sum);
I hope you did not forget constants…

- An even better solution creates a constant for the sentinel. Why?
  
  ```java
  public static final int SENTINEL = -1;
  ```

- Using the constant
  
  ```java
  Scanner console = new Scanner(System.in);
  int sum = 0;
  System.out.print("Enter a number (" + SENTINEL + " to quit): ");
  int inputNumber = console.nextInt();

  while (inputNumber != SENTINEL) {
    sum += inputNumber;
    System.out.print("Enter a number (" + SENTINEL + " to quit): ");
    inputNumber = console.nextInt();
  }

  System.out.println("The total was " + sum);
  ```
Type casting
Type casting

- **type cast**: A conversion from one type to another.
  - Common uses:
    - To promote an `int` into a `double` to achieve exact division.
    - To truncate a `double` from a real number to an integer.

- General syntax:
  
  `( <type> ) <expression>`

- Examples:
  
  ```java
  double result = (double)19 / 5; // 3.8
  int result2 = (int)result; // 3
  ```
Type casting

- Type casting has high precedence and only casts the item immediately next to it.
  
  ```c
  double x = (double)1 + 1 / 2;       // 1.0
  double y = 1 + (double)1 / 2;       // 1.5
  ```

- You can use parentheses to force evaluation order.
  
  ```c
  double average = (double)(a + b + c) / 3;
  ```

- A conversion to `double` can be achieved in other ways.
  
  ```c
  double average = 1.0 * (a + b + c) / 3;
  ```