Kaitlin Hoffmann

Office Hours:

SH 243 MR 11:00 - 12:30 PM via appointment https://calendly.com/hoffmank4/15min

Email: hoffmank4@newpaltz.edu

For TA Office Hours and Email – Please see syllabus

NEW! Supplemental Instruction: Sign up at my.newpaltz.edu

LOOPS CONTINUED

COMPUTER SCIENCE I

OBJECTIVES

- while Loops
- Scope of a variable
- do while loops

LOOPS

- So we've seen a for loop as shown below:

```
for(int i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
```

- Now we'll learn about the while loop:

```
int i = 0;
while(i < 100) {
    System.out.println("Welcome to Java!");
    i++;
}
```

WHILE LOOPS

```
int i = 0;
while(i < 100) {
    System.out.println("Welcome to Java!");
    i++;
}
```

- The above is an example of a **while** loop. The parts it's made up of:

1. **initializing statement**: int i = 0
2. **testing condition**: i < 100
3. **increment/decrement**: i++
4. **Code to be iterated/repeated:**

   System.out.println("Welcome to Java!");

- If you notice, the parts are the same as a for loop; they're just on separate lines! I like to think of while loops as a **deconstructed for loop**.

WHILE LOOPS VS FOR LOOP

```
for(int i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
```

```
int i = 0;
while(i < 100) {
    System.out.println("Welcome to Java!");
    i++;
}
```

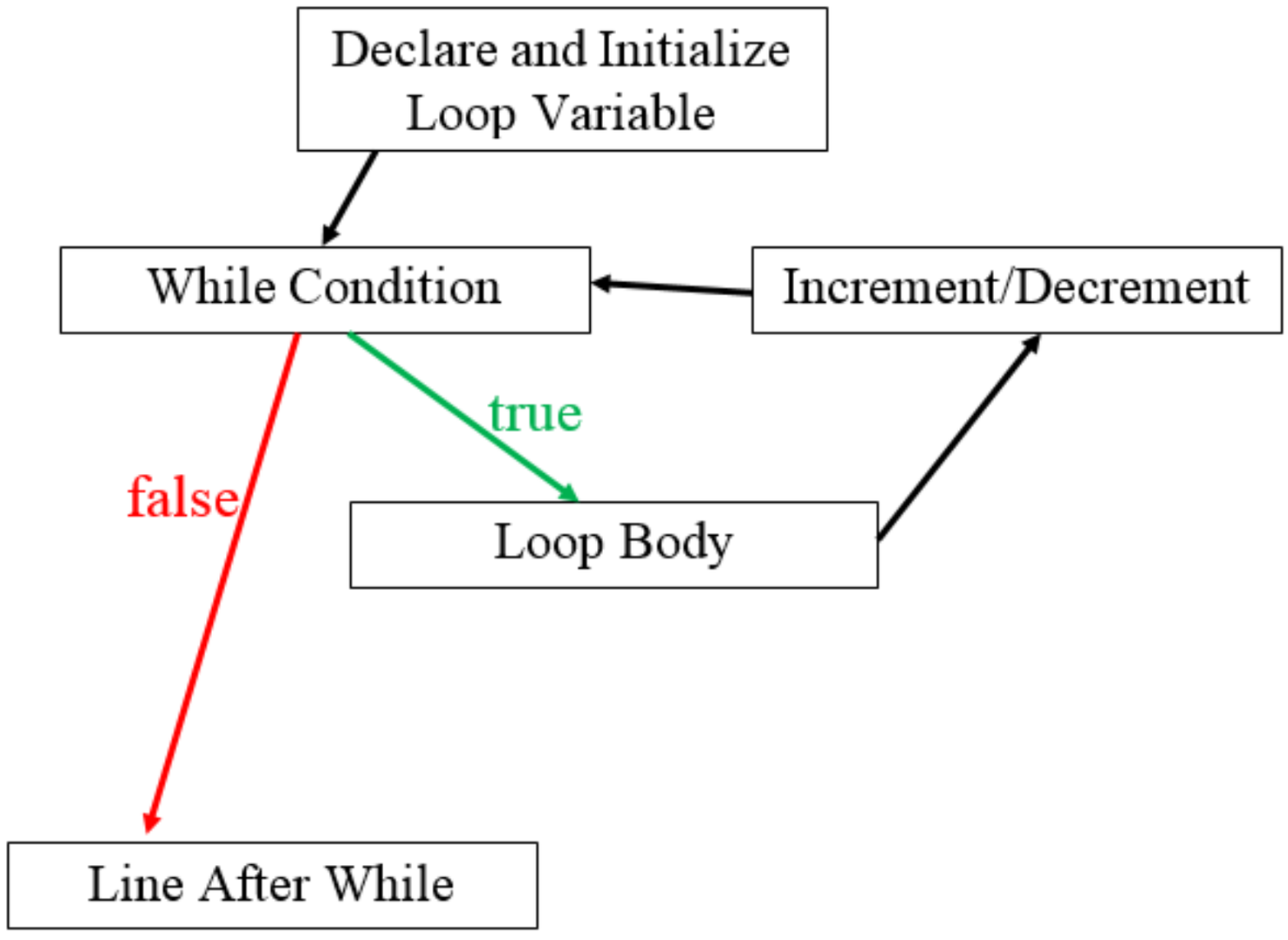
What would happen
if we forgot the i++?

WHEN TO USE A WHILE LOOP VS A FOR LOOP

- In general, you should use a for loop when you know how many times the loop should run.
  - **Ex.** Print the integers between 0 and 100 inclusive
- If you want the loop to break based on a condition other than the number of times it runs, you should use a while loop.
  - **Ex.** Asking a user to try again when creating a password if they didn't meet the conditions.

WHILE LOOP FLOW

```
int i = 0;
while(i < 100) {
    System.out.println("Welcome to Java!");
    i++;
}
```



EXAMPLE 1

- Print 15 to 21 using a **while** loop.

EXAMPLE 1

- Print 15 to 21 using a **while** loop.

```
int i = 15;
while(i <= 21) {
    System.out.print(i + " ");
    i++;
}
```

Output

```
15 16 17 18 19 20 21
```

EXAMPLE 2

- Print the positive numbers that are odd and less than or equal to *n* using a **while** loop.
- When you see *n*, that means that the program should work for any integer, meaning, use Scanner!
- If a user tries entering a number that is negative or zero, we should print out, “Invalid input” and end the program.

EXAMPLE 2 – SOLUTION


```java
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a positive integer: ");

        int n = sc.nextInt();

        if(n <= 0) {
            System.out.println("Invalid input.");
        } else {
            int i = 1;
            while(i <= n) {
                System.out.print(i + " ");
                i+=2;
            }
        }

    }
}
```

Output

```
Enter a positive integer: 12
1 3 5 7 9 11
```

EXAMPLE 3

a. Find the sum of all the integers between 1 and *n* inclusive using a loop. *n* must be greater than 1. Display the sum.

b. Let's make it so if a user tries entering a number less than or equal to 1, they have to try again until they enter a valid number.

- This is an example of where a while loop and for loop are best for different scenarios. We don't know how many times a user will enter an invalid number, so a while loop is best for input validation (part b). However, a for loop is best for finding the sum since we know how many times it will run (*n* times – part a).

EXAMPLE 3 – SOLUTION


```java
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter an integer greater than 1: ");
        int n = sc.nextInt();

        while(n <= 1) {
            System.out.print("Invalid. Try again: ");
            n = sc.nextInt(); // this re-assigns n every loop
        }

        int sum = 0;
        for(int i = 0; i <= n; i++) {
            sum+=i;
        }
        System.out.println("Sum = " + sum);

    }
}
```

Output

```
Enter an integer greater than 1: 1
Invalid. Try again: 0
Invalid. Try again: -2
Invalid. Try again: 3
Sum = 6
```

YOUR TURN — EXERCISE 1

- Print 5 to -5 using a **while** loop.

YOUR TURN — EXERCISE 1

- Print 5 to -5 using a **while** loop.

```
int i = 5;
while(i >= -5) {
    System.out.print(i + " ");
    i--;
}
```

Output

```
5 4 3 2 1 0 -1 -2 -3 -4 -5
```

YOUR TURN — EXERCISE 2

- Print the even numbers between 25 and *n* using a **while** loop. *n* must be greater than 25.
- If n is 25 or less, print, “Invalid input” and end the program. Else, continue with the program.

YOUR TURN — EXERCISE 2 SOLUTION


```java
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter an integer greater than 25: ");

        int n = sc.nextInt();

        if(n <= 25) {
            System.out.println("Invalid input.");
        } else {
            int i = 25;
            while(i <= n) {
                if(i % 2 == 0) {
                    System.out.print(i + " ");
                }
                i++;
            }
        }

    }
}
```

Output

```
Enter an integer greater than 25: 34
26 28 30 32 34
```

SCOPE AND FOR LOOPS

- **Scope** is the part of the program that a variable exists in.
- In a **for** loop, the loop variable does ***not*** exist before or after the loop.
- Loop variables in for loops only exist in the loop.
  - ✦ Example below, variable **i** only exists within the loop

```
for(int i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
```

SCOPE AND FOR LOOPS

- Because the variable exists only with the for loop, we can reuse int i again in the same program.
- They are two unrelated variables that exist at two different times. **Example:**

```
public class Main {

    public static void main(String[] args) {

        for(int i = 0; i<10; i++) {
            System.out.println("Hello World!");
        }

        for(int i = 5; i > -10; i--) {
            System.out.print(i + " ");
        }
    }
}
```

SCOPE AND FOR LOOPS

- **Scope** is the part of the program that a variable exists in.
- In a **while** loop, the loop variable ***does*** exist before and after the loop.
- Loop variables in while loops only exist before, after and inside the loop. **Example**: i exists before, after, and inside

```
int i = 0;
while(i < 100) {
    System.out.println("Welcome to Java!");
    i++;
}
```

SCOPE AND FOR LOOPS

- Because the variable exists throughout the whole program, we ***can't*** reuse int i again in the same program. However, we can re-assign it instead.

- **Example:**

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

i = 5;
while(i>-10) {
    System.out.print(i + " ");
    i--;
}
```

- So we've seen a for loop as shown below:

```
for(int i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
```

- And the while loop:

```
int i = 0;
while(i < 100) {
    System.out.println("Welcome to Java!");
    i++;
}
```

- Now we'll learn about the do while loop briefly.

```
int i = 0;
do {
    System.out.println("Welcome to Java!");
    i++;
} while(i<10);
```

DO WHILE LOOPS

```
int i = 0;
do {
    System.out.println("Welcome to Java!");
    i++;
} while(i<10);
```

- The above is an example of a **while** loop. The parts it's made up of:

1. **initializing statement**: int i = 0
2. **testing condition**: i < 100
3. **increment/decrement**: i++
4. **Code to be iterated/repeated:**

   System.out.println("Welcome to Java!");

- A do while loop looks like a reversed while loop with the code to be iterated coming before the increment/decrement.

WHY USE A DO WHILE LOOP

- In do while loops, the loop body is always executed ***at least*** once.
- In a for and while loop, it's possible to not execute at all if the condition is never met. **Example:**

```java
//this will execute at least once
int i = -1;
do {
    System.out.println("Welcome to Java!");
    i--;
} while(i >= 0);


//this will never execute since -1 never meets the conditon, i >= 0
i = -1;
while(i >= 0) {
    System.out.println("Welcome to Java!");
    i--;
}
```