

# Unit 4: Iteration For Loops

Adapted from:

- 1) Building Java Programs: A Back to Basics Approach  
by Stuart Reges and Marty Stepp
- 2) Runestone CSAwesome Curriculum

# Categories of loops

**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 types "q" to quit.

The **while loop** is usually used for indefinite loops.

# Categories of loops

**definite loop:** Executes a known number of times.

- Print "hello" 10 times.
- Find all the prime numbers up to an integer  $n$ .
- Print each odd number between 5 and 127.

In this lecture, we will see that a for loop is often used to implement a definite loop.

# Repetition with for loops

- So far, repeating a statement is redundant:

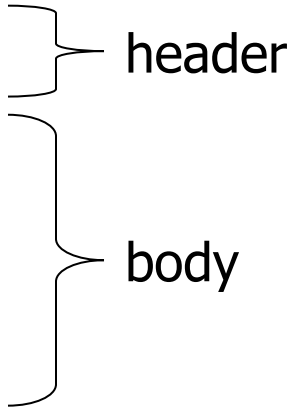
```
System.out.println("Homer says:");  
System.out.println("I am so smart");  
System.out.println("I am so smart");  
System.out.println("I am so smart");  
System.out.println("I am so smart");  
System.out.println("S-M-R-T... I mean S-M-A-R-T");
```

- Java's **for loop** statement performs a task many times.

```
System.out.println("Homer says:");  
for (int i = 1; i <= 4; i++) {    // repeat 4 times  
    System.out.println("I am so smart");  
}  
System.out.println("S-M-R-T... I mean S-M-A-R-T");
```

# for loop syntax

```
for (initialization; test; update) {  
    statement;  
    statement;  
    ...  
    statement;  
}
```



header

body

- Perform **initialization** once.
- Repeat the following:
  - Check if the **test** is true. If not, stop.
  - Execute the **statements**.
  - Perform the **update**.

# Initialization

```
for (int i = 1; i <= 6; i++) {  
    System.out.println("I am so smart");  
}
```

- Tells Java what variable to use in the loop
  - Performed once as the loop begins
  - The variable is called a *loop counter*
    - can use any name, not just `i`
    - can start at any value, not just `1`

# Test

```
for (int i = 1; i <= 6; i++) {  
    System.out.println("I am so smart");  
}
```

- Tests the loop counter variable against a limit
  - Uses comparison operators:
    - < less than
    - <= less than or equal to
    - > greater than
    - >= greater than or equal to

# Repetition over a range

Use a for loop to print:

```
System.out.println("1 squared = " + 1 * 1);  
System.out.println("2 squared = " + 2 * 2);  
System.out.println("3 squared = " + 3 * 3);  
System.out.println("4 squared = " + 4 * 4);  
System.out.println("5 squared = " + 5 * 5);  
System.out.println("6 squared = " + 6 * 6);
```

– Intuition: "I want to print a line for each number from 1 to 6"



# Answer

```
for (int i = 1; i <= 6; i++) {  
    System.out.println(i + " squared = " + i * i);  
}  
System.out.println("Whoo!");
```

## Output:

```
1 squared = 1  
2 squared = 4  
3 squared = 9  
4 squared = 16  
5 squared = 25  
6 squared = 36
```

Whoo!

# Example

Write a for loop to print: 5 9 13 17 21 25

Answer:

```
for (int i = 5; i <= 25; i += 4)
    System.out.print(i + " ");
}
```

# Counting down

- The **update** can use `--` to make the loop count down.
  - The **test** must say `>` instead of `<`

```
System.out.print("T-minus ");
for (int i = 10; i >= 1; i--) {
    System.out.print(i + ", ");
}
System.out.println("blastoff!");
System.out.println("The end.");
```

## – Output:

```
T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
The end.
```

# For Loop in Movies

Groundhog Day (1993); Bill Murray.

Looper (2010); Bruce Willis and Joseph Gordon-Levitt, Emily Blunt.

Edge of Tomorrow (2014); Tom Cruise, Emily Blunt.

# Primes

A prime number is a number that is only divisible by 1 and itself.

For loops can be used to determine if a number is prime. To determine if  $n$  is prime:

- 1) Loop from 1 to  $n$ .
- 2) Count the number of factors of  $n$ . A number  $m$  is a factor of  $n$  if it divides evenly into  $n$ , that is, if  $n \% m == 0$ .
- 3)  $n$  is prime if the number of factors is exactly 2. Otherwise, it is composite.

# isPrime

```
public static boolean isPrime(int n) {  
    int factors = 0;  
    for (int i = 1; i <= n; i++) {  
        if (n % i == 0) {  
            factors++;  
        }  
    }  
  
    if (factors == 2) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

- Calls to methods returning `boolean` can be used as tests:

```
if (isPrime(57)) {  
    ...  
}
```

# isPrime method

The condition below is verbose. Can we simplify it? Replace it with one statement!

```
public static boolean isPrime(int n) {  
    int factors = 0;  
    for (int i = 1; i <= n; i++) {  
        if (n % i == 0) {  
            factors++;  
        }  
    }  
    if(factors==2)  
        return true;  
    else  
        return false;  
}
```

# Improved isPrime method

The following version utilizes Boolean Zen:

```
public static boolean isPrime(int n) {  
    int factors = 0;  
    for (int i = 1; i <= n; i++) {  
        if (n % i == 0) {  
            factors++;  
        }  
    }  
    return factors == 2; // if n has 2 factors, true  
}
```



# For vs While

Write a loop to compute the sum:

$$1 + 2 + 3 + \dots + 99 + 100$$

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

**Both for and while loops can be used to solve this problem.**

```
int sum = 0;
for(int i = 1; i <= 100; i++) {
    sum += i;
}
```

# For vs. While

Although for and while loop can generally be interchangeable. It is best practice to use a for loop as a definite loop where the beginning and termination of the loop is well defined.

```
for(int i = 1; i <= 100; i++) {  
    System.out.println(i);  
}
```

This for loop executes 100 times. It is a definite loop. It is usually better to use a for loop as a definite loop.

# For vs. While

If the termination condition of a loop is less well defined, use a while loop.

```
int x=3;
while(x == 3)
{
    System.out.println(x);
    x=(int) (4*Math.random()); // x is 0,1,2, or 3
}
```

This while loop executes an unknown number of times. It is a indefinite loop. It is better to use a while loop here.

# Lab 1

Create a new repl on repl.it. You can use this repl for BOTH Lab 1 and Lab 2.

Write 3 static methods: `multipleOf3`, `multipleOf4Not5` and `printList` as explained below.

```
//print multiples of 3 from m to n inclusive.
```

```
// MUST USE %
```

```
public static void multipleOf3(int m, int n)
{...}
```

```
// print multiples of 4 and not of 5 from 4
```

```
// to m inclusive. Assume m>=4 Must use %
```

```
public static void multiple4Not5(int m){...}
```

# Lab 1

Write a method `printList` that accepts two integer as arguments. You can assume that these integers are different.

If the first argument is smaller than the second, print the even integers in natural order between the arguments including, if necessary, the endpoints. If the first argument is larger than the second, print the odd numbers between them. Write out the complete method header.

The following calls from the main method should have the output below.

```
printList(2, 7); // 2 4 6
printList(9, 3); // 3 5 7 9
```

# Lab 2: For or While?

Write a static method named `fourHeads()` that repeatedly flips a coin until four heads *in a row* are seen. You should use `Math.random()` to give an equal chance to a head or a tail appearing.

Each time the coin is flipped, what is seen is displayed (H for heads, T for tails). When four heads in a row are flipped a congratulatory message is printed. **Should you use a for or a while loop?**

Here are possible outputs of two calls to `fourHeads`:

```
fourHeads();
```

```
T T T H T H H H H
```

```
Four heads in a row!
```

```
fourHeads();
```

```
T H T H T T T T H H T H H H H
```

```
Four heads in a row!
```

# Lab 2: For or While?

Write a static method named `printTwoDigit` that accepts an integer  $n$  as a parameter and that prints a series of  $n$  randomly generated numbers. The method should use `Math.random()` to select numbers in the range of 10 to 19 inclusive where each number is equally likely to be chosen.

After displaying each number that was produced, the method should indicate whether the number 13 was ever selected ("we saw a 13!") or not ("no 13 was seen."). You may assume that the value of  $n$  passed is at least 0. **Should you use a for or a while loop?**

You should an output similar to below. (see next slide)

# Lab 2

Call	<u>printTwoDigit(4);</u>	<u>printTwoDigit(7);</u>
Output	<u>next</u> = 12 <u>next</u> = 10 <u>next</u> = 16 <u>next</u> = 11 <u>no</u> 13 was seen.	<u>next</u> = 12 <u>next</u> = 19 <u>next</u> = 12 <u>next</u> = 13 <u>next</u> = 11 <u>next</u> = 16 <u>next</u> = 13 we saw a 13!

□



# References

1) Building Java Programs: A Back to Basics Approach by Stuart Reges and Marty Stepp

2) Runestone CSAwesome Curriculum:

<https://runestone.academy/runestone/books/published/csawesome/index.html>

For more tutorials/lecture notes in Java, Python, game programming, artificial intelligence with neural networks:

<https://longbaonguyen.github.io>