Programming 1

Lecture 4 – Iteration

Faculty of Information Technology
Hanoi University

The power of computer

- Comes from the fact that it can repeat things efficiently.
  - They are extremely fast!
- Statements can be made to repeat for a great number of times.
  - Most problems require repeating a lot of calculations or actions.
- All programming languages support repetition with a feature called **Loop**.

The `for` loop

- Used to repeat a block of code many times.

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

block of code to repeat

the starting point

stopping condition
(loop stops when it's false)

how `i` changes
(after a step/iteration)

```
for (i = 0; i < 10; i++)
```

The `for` loop explained

```
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Iteration " + i);
}
```

- Let **`i = 0`**
- Now **`i < 2`** is **`true`**, let's display the text:

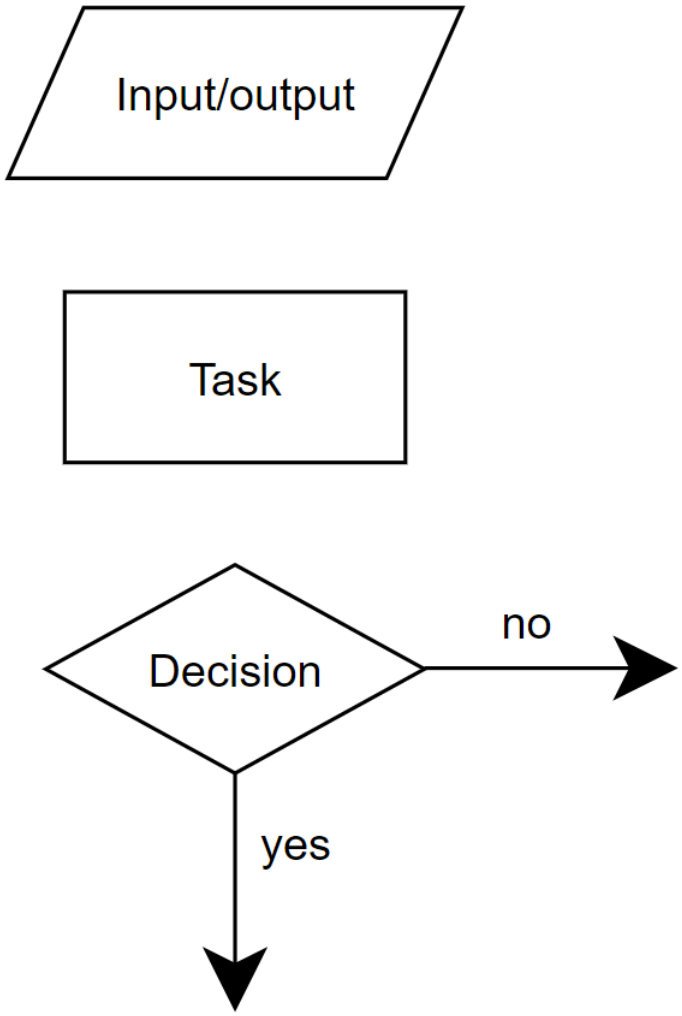
  `Iteration 0`
- Execute **`i++`**, and now **`i`** becomes **`1`**
- The condition **`i < 2`** is still **`true`**, let's display the text:

  `Iteration 1`
- Execute **`i++`**, and now **`i`** becomes **`2`**
- Finally **`i < 2`** is **`false`**, we won't display another line.Loop ends.

How many lines have we displayed? What numbers were shown?

Flowcharts

- Flow charts are made up of elements for tasks, input/output, and decisions.
- **The basic idea:** link tasks and inputs, outputs in the sequence in which they should be executed.



`for` loop flowchart

```
for (i = 2; i <= 6; i = i + 2) {
    print (i + 1)
}
```

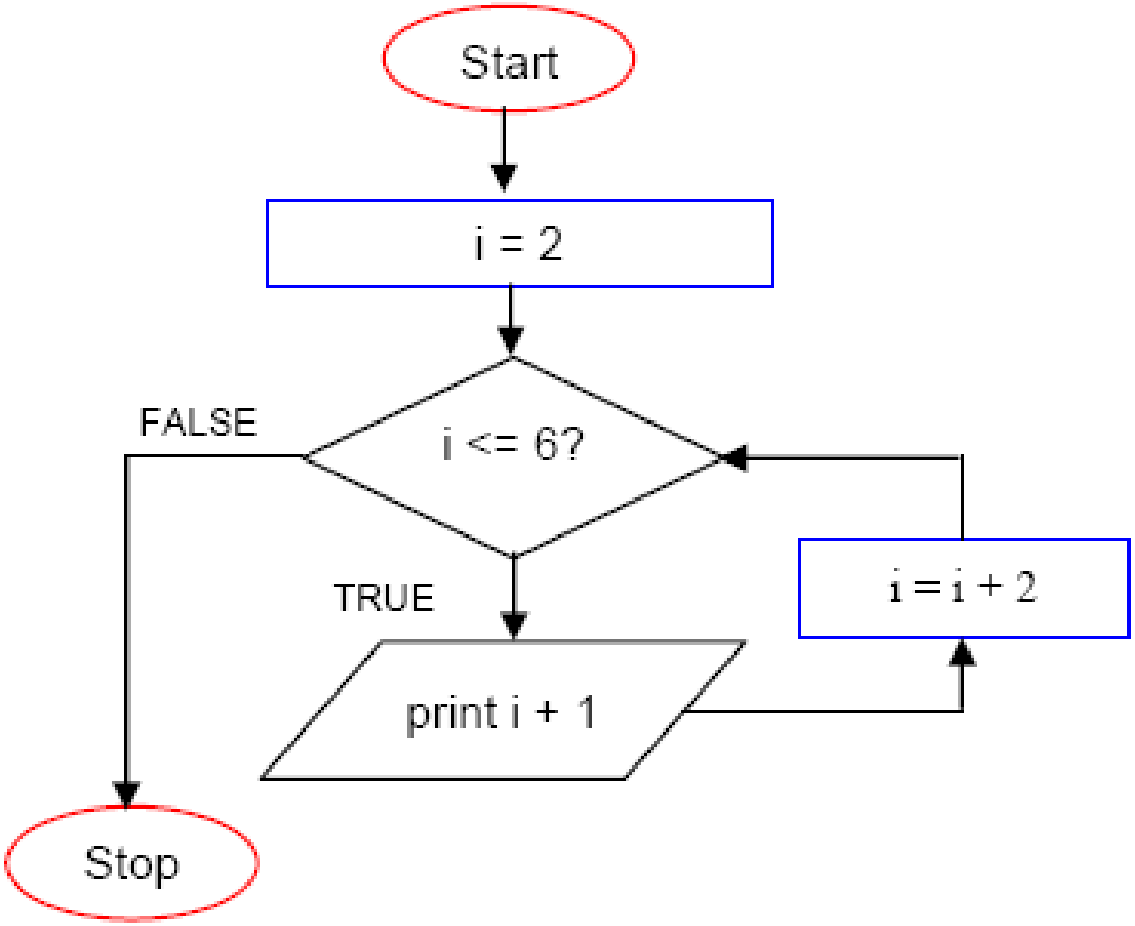


Image credit: stackoverflow.com

Example 1

```
public class LoopDemo {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            if (i % 2 == 0) {
                System.out.println(i);
            }
        }
    }
}
```

For loop examples

Loop	Values of i	Comment
for (i = 0; i <= 5; i++)	0 1 2 3 4 5	The loop is executed 6 times
for (i = 5; i >= 0; i--)	5 4 3 2 1 0	Use i-- for decreasing values
for (i = 0; i < 9; i = i + 2)	0 2 4 6 8	Use i = i + 2 for a step size of 2
for (i = 0; i != 9; i = i + 2)	0 2 4 6 8 10 12 … (infinite loop)	Use < or <= instead of != to avoid this problem
for (i = 1; i <= 20; i = i * 2)	1 2 4 8 16	You can modify i any way you want, such as doubling it in every step
for (i = 0; i < str.length(); i++)	0 1 2 … str.length() – 1	In the loop body, use str.charAt(i) to get the i^{th} char

Short-hand operators

Short-hand	Equivalent	Comment
a++	a = a + 1	a++ has the value of a
++a	a = a + 1	++a has the value of (a + 1)
a--	a = a - 1	a-- has the value of a
--a	a = a - 1	--a has the value of (a - 1)
a += 3	a = a + 3	Increments then assigns
a -= 4	a = a - 4	Decrements then assigns
a *= 5	a = a * 5	Multiplies then assigns
a /= 6	a = a / 6	Divides then assigns
a %= 2	a = a % 2	Modulus then assigns

The trace table technique

```
int t;
int x = 3;
for (t = 0; t < 16; t += 3) {
    x *= 3;
}
System.out.println(x);
System.out.println(t);
```

t	x
-	-
-	3
0	3
0	9
3	9
3	27
6	27
6	81
9	81
9	243
12	243
…	…

Example 2

- Let the user enter 5 real values and calculate their average.

Answer

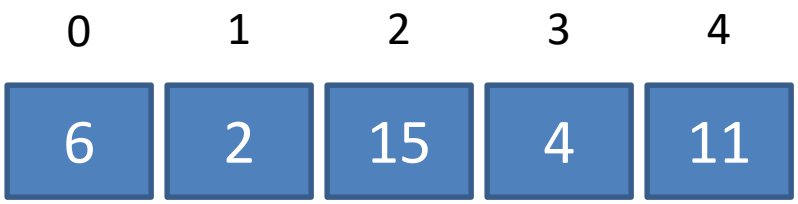
```
Scanner sc = new Scanner(System.in);
int n = 5;
double sum = 0, average;

for (int i = 0; i < n; i++) {
    System.out.print("Enter number #" + (i + 1));
    sum = sum + sc.nextDouble();
}
average = sum / n;
System.out.println("Average is: " + average);
```

The `array` structure

- At times, we have to handle a lot of values and declaring too many variables is not a good option
- So they gave programming languages a tool to group many values into one variable called array
- We can do something like this:

```
int[] a = {6, 2, 15, 4, 11};
System.out.println(a[0] + a[2]); // 6 + 15
```

- We call the values by their **position** in the array
- Position starts from **0**



Declare arrays

```
int[] a;
double[] b;
String[] names;
```

- If you declare them like this, they will be **null**
- **null** is the value of an object which hasn't been initialized yet

```
String s; // s is null
Scanner sc; // sc is also null
Scanner sc2 = new Scanner(System.in);
// sc2 got initialized and isn't null
sc = new Scanner(System.in);
s = "Hello";
// sc and s are initialized
```

Initialize `arrays`

```
int[] a = new int[10];
```

- An array of 10 zeros

```
double[] b = new double[5];
```

- A double array of 5 zeros

```
String[] names = new String[3];
```

- An array of 3 `null` values

→ **Reason:** The default value for `int` and `double` is `0` and for `String` is `null`

Initialize `arrays` with values

```
int[] a = {2, 4, 6};
```

- An array of 3 numbers

```
double[] b = {0.2, 0.4, 0.1, -0.13, 0.9};
```

- An array of 5 real numbers

```
String[] names = {"Ha", "Tu", "Hoa"};
```

- An array of 3 strings

Getting `array` length

```
int[] a = {2, 4, 6};
System.out.println(a.length); // 3
```

```
double[] b = {0.2, 0.4, 0.1, -0.13, 0.9};
System.out.println(b.length); // 5
```

```
String[] names = new String[10];
System.out.println(names.length); // 10
```

→ Array length can be automatically determined (based on initialized values) or specified on declaration

arrays and the for loop

- arrays are most useful when combined with the for loop

```
double[] b = {0.2, 0.4, 0.1, -0.13, 0.9};
for (int i = 0; i < b.length; i++) {
    System.out.println("#" + i + ": " + b[i]);
}
```

Result

```
#0: 0.2
#1: 0.4
#2: 0.1
#3: -0.13
#4: 0.9
```

The previous example

- Let the user enter 5 real values and calculate their average.

Answer with array

```
double[] numbers = new double[5];
Scanner sc = new Scanner(System.in);
double sum = 0, average;

for (int i = 0; i < numbers.length; i++) {
    System.out.print("Enter number #" + (i + 1));
    numbers[i] = sc.nextDouble();
    sum = sum + numbers[i];
}
average = sum / numbers.length;
System.out.println("Average is: " + average);
```

How to stop a `for` loop

- When we search for something with a for loop, we may want to stop looking as soon as it is found.
- E.g. Find one negative number from an array such as: `int[] a = {6,4,-2,6,5,9,15,-6,2};`

```
for (int i = 0; i < a.length; i++) {
    if (a[i] < 0) {
        System.out.println("Found: " + a[i]);
    }
}
```

What is the output of the above piece of code?

How to stop a `for` loop

```
int[] a = {6, 4, -2, 6, 5, 9, 15, -6, 2};
for (int i = 0; i < a.length; i++) {
    if (a[i] < 0) {
        System.out.println("Found: " + a[i]);
    }
}
```

Output:

```
Found: -2
Found: -6
```

- This piece of code found 2 negative numbers but only one is required.
- After -2 is found at the 3rd iteration, the loop continues to run until it finishes after 9 iterations.
- It should've stopped at the 3rd iteration.

How to stop a `for` loop

```
int[] a = {6, 4, -2, 6, 5, 9, 15, -6, 2};
for (int i = 0; i < a.length; i++) {
    if (a[i] < 0) {
        System.out.println("Found: " + a[i]);
        break;
    }
}
```

Output:

```
Found: -2
```

- The **`break`** statement terminates an on-going **`for`** loop.
- **`break`** affects the loop which immediately contains it.

How to stop a `for` loop

```
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        if (i * j > 30) {
            System.out.println(i + "," + j);
            break; // out of j loop
        }
    }
}
```

Output:

```
4,8
5,7
6,6
7,5
8,4
9,4
```

How to stop a `for` loop

```
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        if (i * j > 30) {
            System.out.println(i + "," + j);
        }
    }
    if (i == 4) break; // out of i loop
}
```

Output:

```
4,8
4,9
```

The `while` loop

- Repeat a block of code as long as a condition holds true
- The number of iterations is not specific and can be zero

The loop stops when **loop condition** is false

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

block of code to repeat

`while` loop explained

```
int n = 1, e = 0;
while (n < 10) {
    n = n * 2;
    e++;
}
System.out.println("2^" + e + " = " + n);
```

What is the output?

- Let **`n = 1, e = 0`**
- Now **`n < 10`** is **`true`**, let's continue the loop.
- Execute **`n = n * 2`** and **`e++`** → **`n`** becomes **`2`**, **`e`** becomes **`1`**
- The condition **`n < 10`** is still **`true`**, let's continue the loop.
- Execute **`n = n * 2`** and **`e++`** → **`n`** becomes **`4`**, **`e`** becomes **`2`**
- The condition **`n < 10`** is still **`true`**, let's continue the loop.
- Execute **`n = n * 2`** and **`e++`** → **`n`** becomes **`8`**, **`e`** becomes **`3`**
- The condition **`n < 10`** is still **`true`**, let's continue the loop.
- Execute **`n = n * 2`** and **`e++`** → **`n`** becomes **`16`**, **`e`** becomes **`4`**
- Finally **`n < 10`** is **`false`**, the loop ends.

`while` loop flowchart

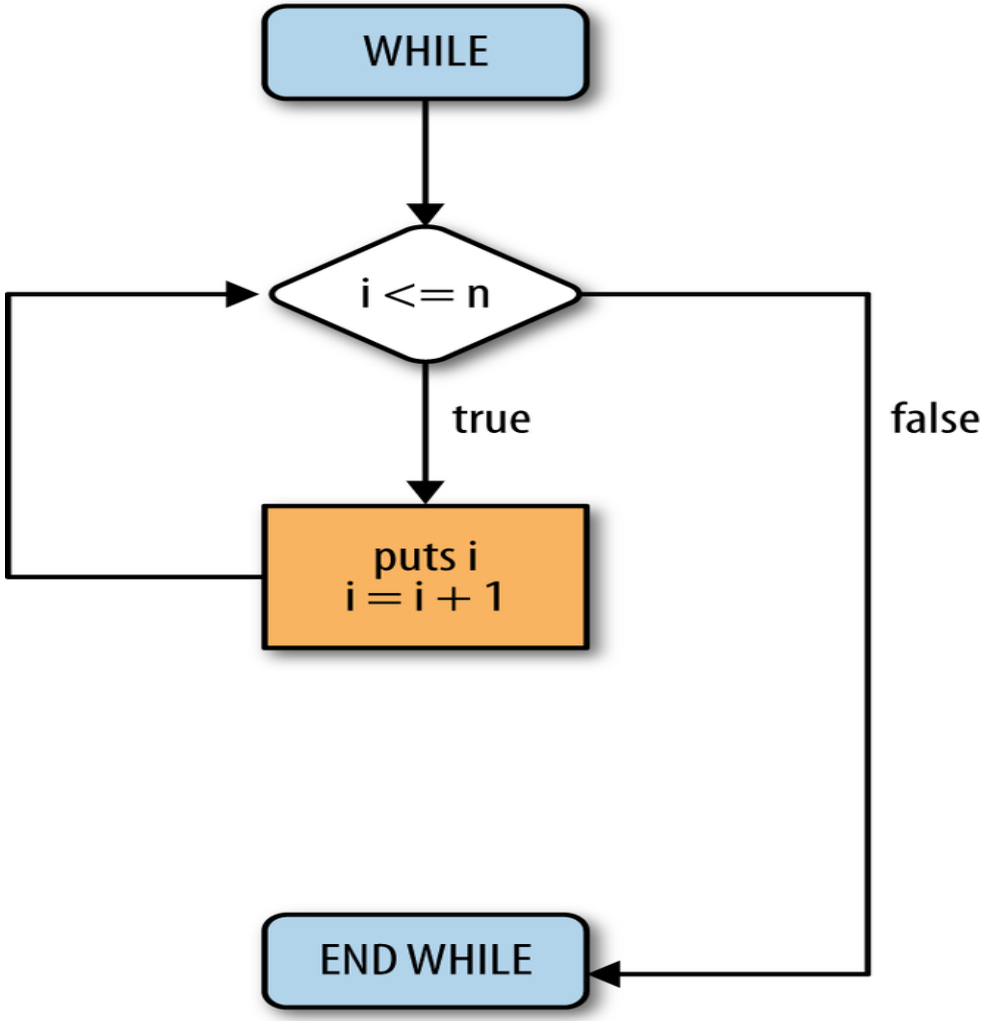


Image credit: oreilly.com

The do…while loop

- Repeat a block of code once, and then continues as long as a condition holds true
- The number of iterations is not specific but always >= 1

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

block of code to repeat

The loop stops when **loop condition** is false

do...while loop explained

```
int n;
do {
   System.out.print("Enter a positive integer: ");
   n = sc.nextInt();
} while (n <= 0);
System.out.println("Thank you!");
```

- Let **n** be uninitialized
- Print a text message to ask user to enter a positive integer.
- Get **n**'s value from the keyboard with **sc.nextInt()** method.
- Repeat if the user does not obey you.

`do...while` loop flowchart

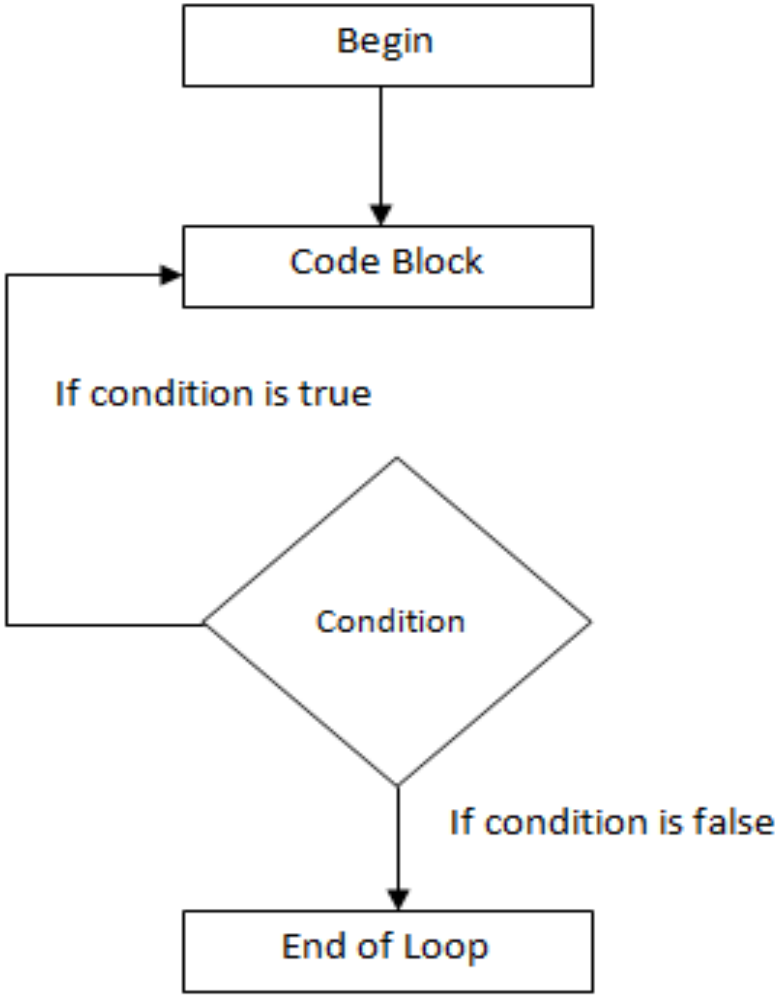


Image credit: tutorialspanel.com

Stop a `while` loop with `break`

- Similar to the `for` loop, the `while` loop can be terminated with the `break` statement.

```
while (n < 10) {
    if (sc.nextLine().equals("q")) {
        System.out.println("Goodbye!");
        break;
    }
    n++;
}
```

Skip the rest of an iteration with `continue`

- Similar to the `for` loop, an iteration of a `while` loop and do…while loop can be interrupted with `continue`

```
int n = 0;
while (n < 3) {
    n++;
    System.out.println(n);
    if (n == 2) continue;
    System.out.println("...checked!");
}
```

Output:

```
1
...checked!
2
3
...checked!
```

Example 3

- Replace all spaces in a string with underscores.

Answer 1

```
String s = "To infinity and beyond!";

for (int i = 0; i < s.length(); i++) {
    if (s.charAt(i) == ' ') {
        System.out.print("_");
    } else {
        System.out.print(s.charAt(i));
    }
}

System.out.println(); // add a new line at the end
```

Comment: This solution uses a lot of print statements.

Answer 2

```
String s = "There's a snake in my boot!";
String s2 = "";

for (int i = 0; i < s.length(); i++) {
    if (s.charAt(i) == ' ') {
        s2 = s2 + "_"; // this creates a new String
    } else {
        s2 = s2 + s.charAt(i); // same as above
    }
}

System.out.println(s2);
```

Comment: **This solution creates a lot of String objects, which is computationally expensive.**

Answer 3

```
String s = "You've got a friend in me.";
char[] c = new char[s.length()];

for (int i = 0; i < s.length(); i++) {
    if (s.charAt(i) == ' ') {
        c[i] = '_';
    } else {
        c[i] = s.charAt(i);
    }
}

// we can create a String from a char array
System.out.println(new String(c));
```

Comment: This solution makes use of an array to improve performance.