ÖZYEĞİN UNIVERSITY

CS105
Introduction to Object-Oriented Programming

Prof. Dr. Nizamettin AYDIN

naydin@itu.edu.tr

nizamettin.aydin@ozyegin.edu.tr

Procedural Programming

Outline

- Procedural Programming Paradigm
- Sequence
- Alternation
- Alternation - switch statement
- Repetition
- for loop
- while loop
- do..while loop
- Nested loop
- Examples

Procedural Programming Paradigm

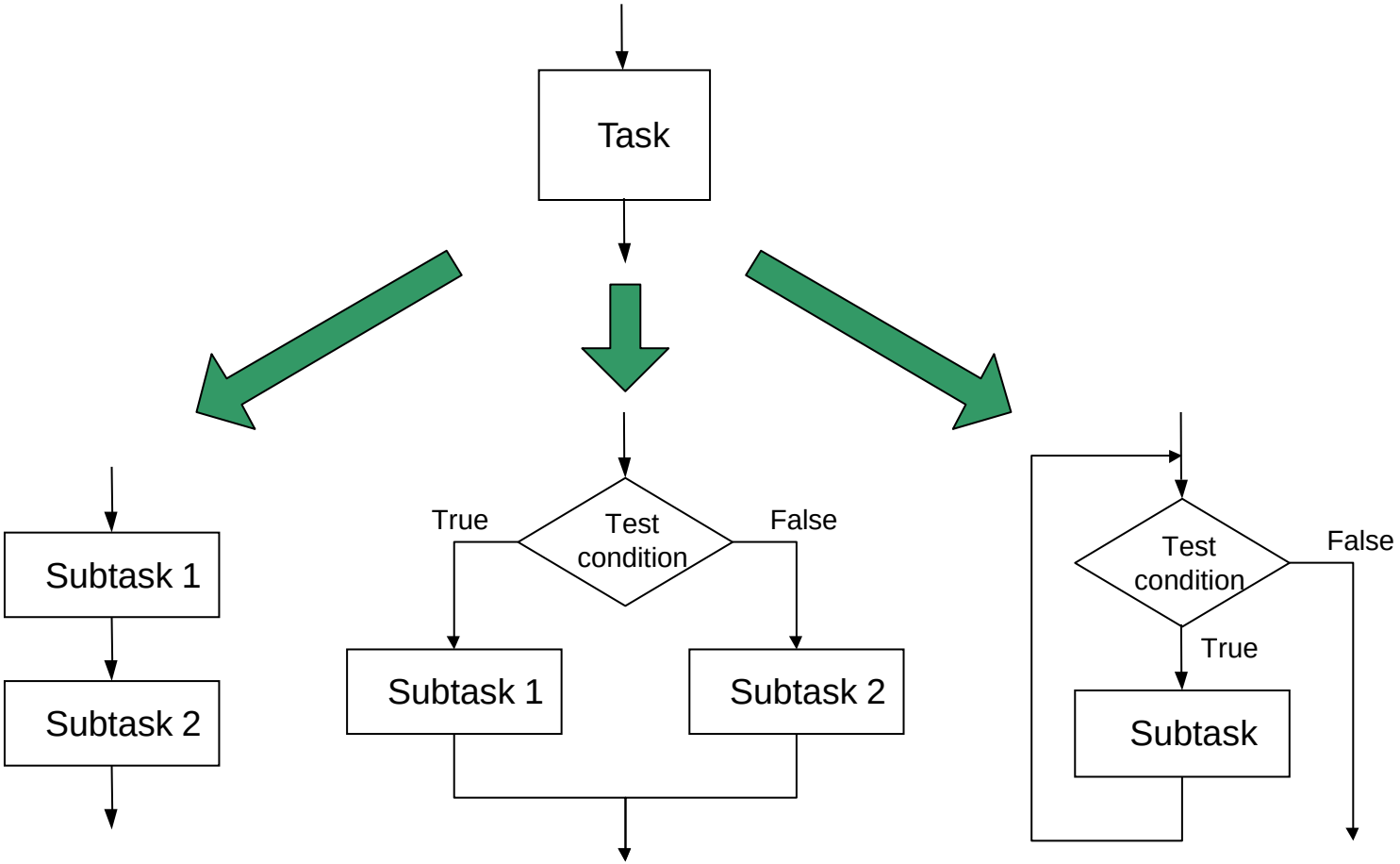
- Java is an Object-Oriented language
  - –but it also has the procedural programming concepts as its core
- procedural programming:
  - –a style of programming where a problem is broken down in a set of smaller procedures,
    - also called functions and, in Java's case, methods
- the term is also used to include a set of programming code constructions (structured programming) that deliver the minimal requirements of a general-purpose programming language.
  - –These constructions themselves arise from the pioneering work of Alan Turing and his abstract Turing machine

Procedural Programming Paradigm

- It can be demonstrated that a programming language is general purpose (i.e.can perform any computable calculation) provided it exhibits 3 characteristics:

  - Sequence:
    - processes one instruction after another, until all instructions have been executed.
      - linear statements
  - Alternation (also called selection):
    - selects one execution path from a set of alternatives.
      - conditionals
  - Repetition (also called iteration):
    - repeatedly executes some code whilst some condition persists.
      - loops

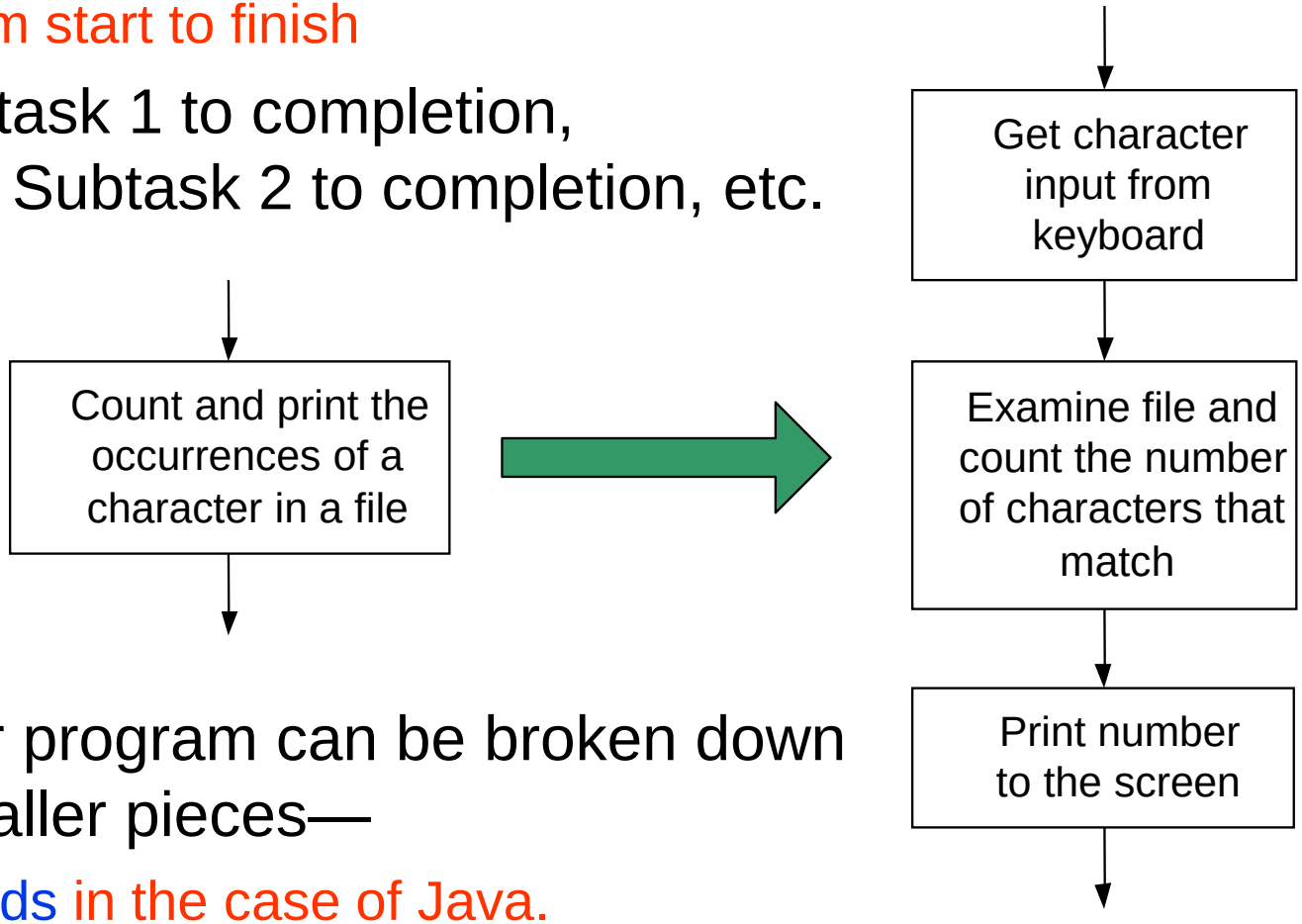
Three Basic Constructs



Sequential ***Conditional*** ***Iterative***

Sequence

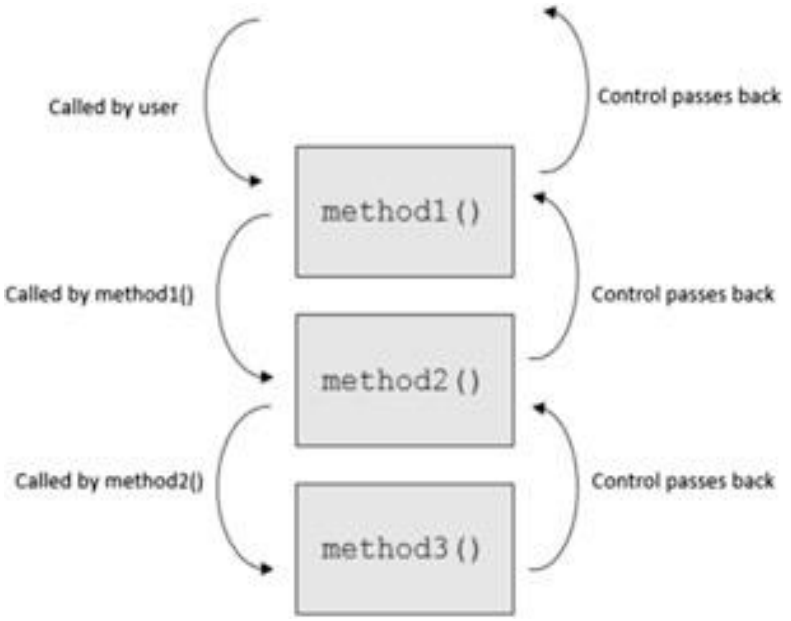
- Instructions are executed in a given reliable order
  - –i.e. from start to finish
- Do Subtask 1 to completion, then do Subtask 2 to completion, etc.



- A larger program can be broken down into smaller pieces—
  - – methods in the case of Java.
    - One method can then call upon another

Sequence

- When a method calls upon another method, execution of the calling method is parked whilst the called method is executed.
- Once the called method has completed execution, control passes back to the calling method.
- At any stage of the execution of the program, there is a stack of calling methods
  - –where the order of the stack is determined by the sequence in which the method calls took place.
- The example in the next slide will help you understand this point:



Sequence

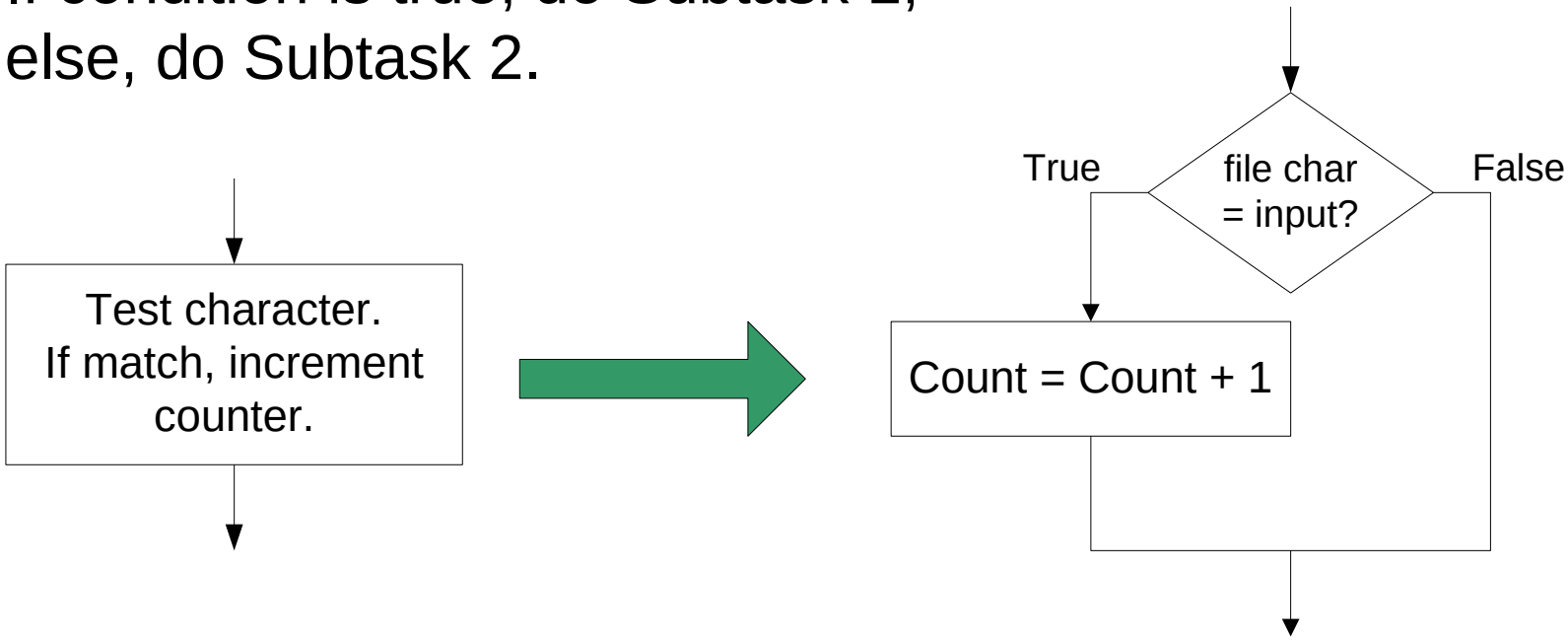
```
public void method1()
{
      int x=2;
      int y=3;
      method2();
      System.out.println("Value of y in method1 is: " + y);
}
public void method2()
{
      boolean x = true;
      int y=6;
      System.out.println("x is: " + x);
      System.out.println("Value of y in method2 is: " + y);
      method3();
}
public void method3()
{
      int y = 5;
      System.out.println("Value of y in method3 is: " + y);
}
```

y is a method variable, so the version in method1() is unique to that method, and distinct from the y used in method2() and so on

what goes inside the method, stays in the method.

Alternation

- AKA selection, or conditionals
- If condition is true, do Subtask 1; else, do Subtask 2.



- In Java, it is delivered using if and switch statements.
  - They are both equally expressive in that anything that is written using switch can be re-written using if.

Alternation - if statement

- The if statement takes the general form:

```
if (<test-condition-1>)
    {
        // Body of statement
    }
else if (<test-condition-2>)
    {
        // Body of statement
    }
else
{
    // Body of statement
}
```

  - For the if statement, operators that return values of true or false are used.

Alternation

- Common logical operators:

Operator	Meaning
==	Is equal to
!=	Is not equal to
>	Is greater than
>=	Is greater than or equal to
<	Is less than
<=	Is less than or equal to
&&	Logical AND
||	Logical OR

– Do not mix up == (equality operator) with = (assignment operator)

- The equality operator tests to see whether two values are the same or not.
- The assignment operator sets a variable to a specific value.

Some example if statements

```
int a = 1;
int b = 2;
int c = 3;
boolean x = true;
// Example 1
if (a == 1)
{
    System.out.println("Get here if a has the value
1");
}
// Example 2
if (a != 1)
{
    System.out.println("a does not have the value 1");
}
```

Some example if statements

```
// Example 3
if (a > 2)
{
    System.out.println("a has a value greater than 2");
}
else
{
    System.out.println("Otherwise we get here!");
}
// Example 4
if ((a==1) && (x == true))
{
    System.out.println("a is 1 AND x is true");
}
```

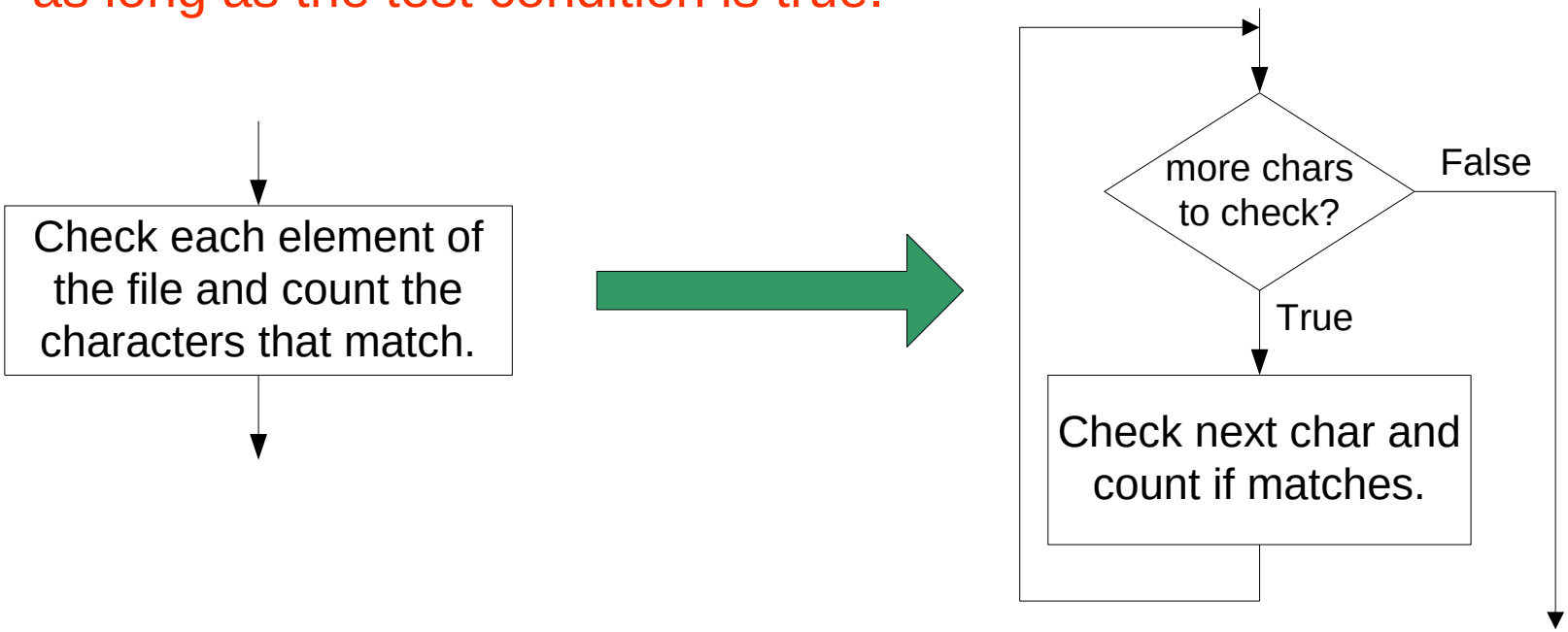
Alternation - switch statement

- A switch statement makes use of a single variable to determine which of the options is to be executed
  - Each option is characterized by a case statement, and each case should end with a break statement
- example of a switch statement:

```
int x = 3;
switch(x)
{
     case 1:
             System.out.println("Option if x has the value 1");
             break;
     case 2:
             System.out.println("Option if x has the value 2");
             break;
     case 3:
             System.out.println("Option if x has the value 3");
             break;
     default:
             System.out.println("x has some other value");
             break;
}
```

Repetition

- AKA iteration, or loop
- Do Subtask over and over,
  - –as long as the test condition is true.



- In the Java language, it is delivered using for, while and do .. while loops.

Repetition – for loop

- Output:

```
Homer says:
I am so smart
I am so smart
I am so smart
I am so smart
S-M-R-T... I mean S-M-A-R-T
```

- Code:

```
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");
```

- Repeating a statement is redundant

Repetition – for loop

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

```
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");
```

- Output

```
Homer says:
I am so smart
I am so smart
I am so smart
I am so smart
S-M-R-T... I mean S-M-A-R-T
```

Repetition – for loop

- The for loop takes the following general form:

```
for (<initial-state>; <test-condition>; <action>)
{
    // Body of loop
}
```

- Three things need to be specific in the round brackets:
  - The <initial-state>:
    - the initial value of some controlling variable that will be set one-time prior to the first evaluation of the test condition to some initial value.
  - The <test-condition>:
    - a Boolean test that will determine whether the loop continues to execute.
  - The <action>:
    - a short section of code that is executed each time the body of the loop has finished executing.
- The body of the loop is any amount of valid Java code
  - (including, although not limited to, sequence, alternation and other examples of repetition)

Repetition – for loop

- 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**

Repetition – for loop

- 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 – for loop

- Action:

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

- Increment and decrement
  - –shortcuts to increase or decrease a variable's value by 1

Shorthand	Equivalent longer version
variable++;	**variable = variable + 1;**
variable--;	**variable = variable - 1;**

```
int x = 2;
x++;                    // x = x + 1;        x now stores 3
double gpa = 2.5;
gpa--;                  // gpa = gpa - 1;    gpa now stores 1.5
```

Repetition – for loop

- Modify and assign:
  - –shortcuts to modify a variable's value

Shorthand	Equivalent longer version
variable += value;	**variable = variable + value;**
variable -= value;	**variable = variable - value;**
variable *= value;	**variable = variable * value;**
variable /= value;	**variable = variable / value;**
variable %= value;	**variable = variable % value;**

```
x += 3;                  // x = x + 3;

gpa -= 0.5;              // gpa = gpa - 0.5;

number *= 2;             // number = number * 2;
```

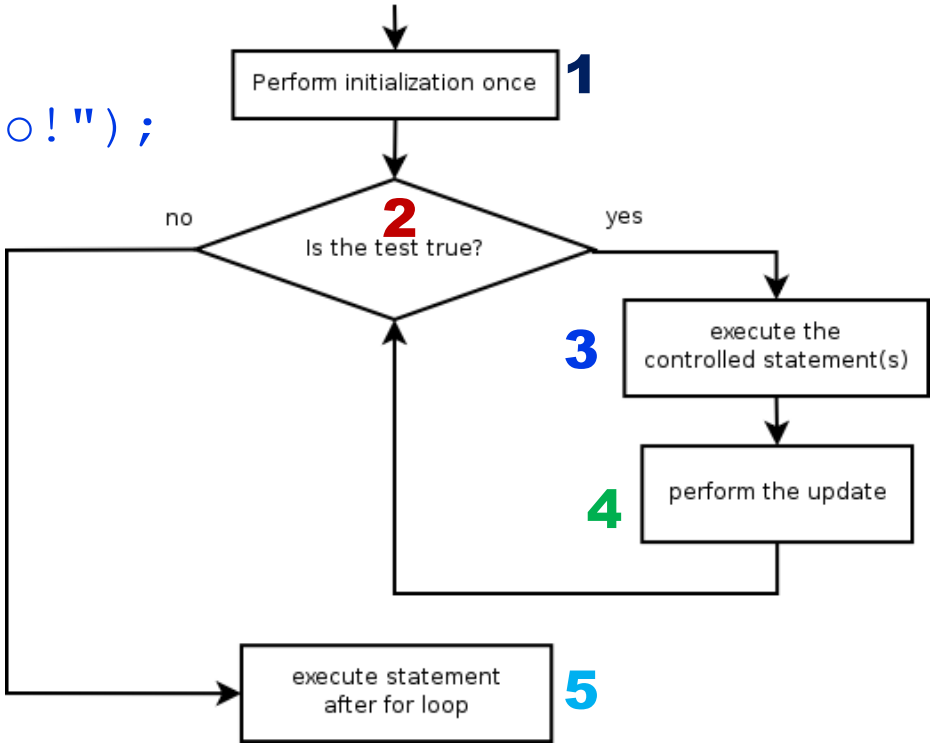
Repetition – for loop

- Loop walkthrough:

```
for (int i = 1; i <= 4; i++)
{
  3 System.out.println(i+" squared="+(i * i));
}
5 System.out.println("Whoo!");
```

- Output:

```
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!
```



Repetition – for loop

- Multi-line Loop Body:

```
System.out.println("+----+");
for (int i = 1; i <= 3; i++) {
    System.out.println("\\    /");
    System.out.println("/    \\");
}
System.out.println("+----+");
```

- Backslash(\)is the 'escape' character:
  - you write one backslash, and then one other character which together represents a single character

- Output:

```
+----+
\    /
/    \
\    /
/    \
\    /
/    \
+----+
```

Repetition – for loop

- Expression for counter:

```
int highTemp = 5;
for (int i = -3; i <= highTemp / 2; i++) {
    System.out.println(i * 2 + 32);
};
```

- Output:

```
26
28
30
32
34
36
```

Repetition – for loop

- System.out.print:
  - –Prints without moving to a new line
    - allows you to print partial messages on the same line

```
int highTemp = 5;
for (int i = -3; i <= highTemp / 2; i++) {
    System.out.print((i * 2 + 32) + "   ");
};
```

- Output:

```
26   28   30   32   34   36
```

  - Concatenate "   " to separate the numbers

Repetition – for loop

- 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.");
```

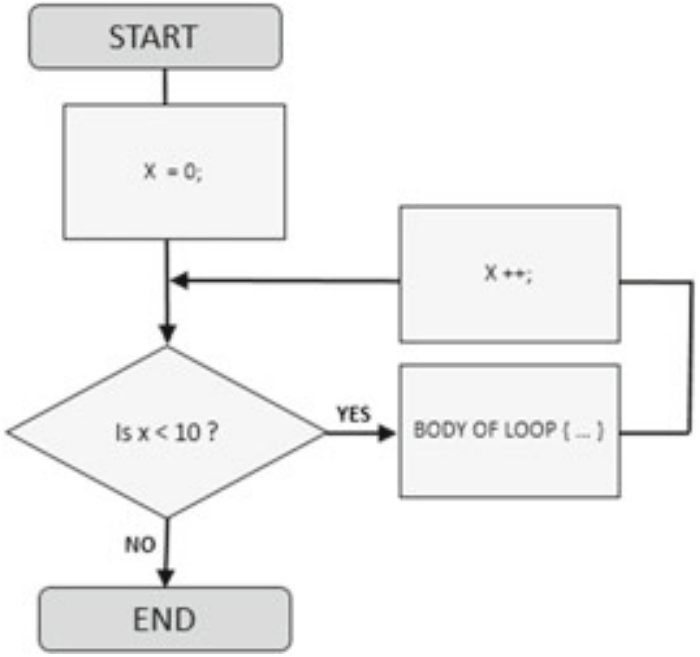
- Output:

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

For loop example

- a code example and corresponding flowchart

```
public void forLoop()
        {
        // Note that x has scope
        // of the for loop
        for (int x = 0; x < 10; x++)
                {
                System.out.println(x);
                }
        }
```



- Note that the loop variable x was actually declared within the scope of the for loop.
  - It therefore has the scope of just that loop and does not exist outside the scope of the loop.

Repetition – while loop

- The while loop takes the following general form:

```
while (<test-condition>)
{
    // Body of loop
}
```

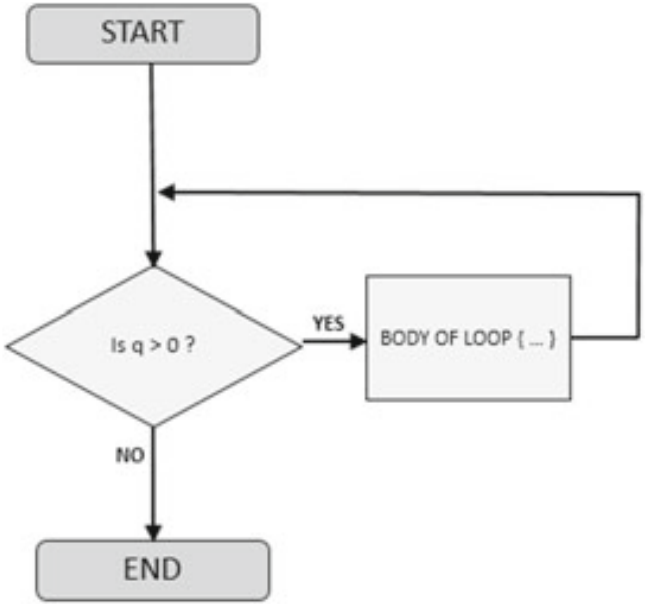
With the while loop, you can incorporate as much action style code into the body of the loop as is required to solve a problem.

- a code example and corresponding flowchart

```
public void whileLoop()
    {
    // q has scope of the method
    int q = 5;
    while (q > 0)
            {
            System.out.println(q);
            q--;
            }
    }
```

- Output:

```
5
4
3
2
1
```



Repetition – do .. while loop

- The while loop takes the following general form:

```
do
{
        // Body of loop
}
while (<test-condition>);
```

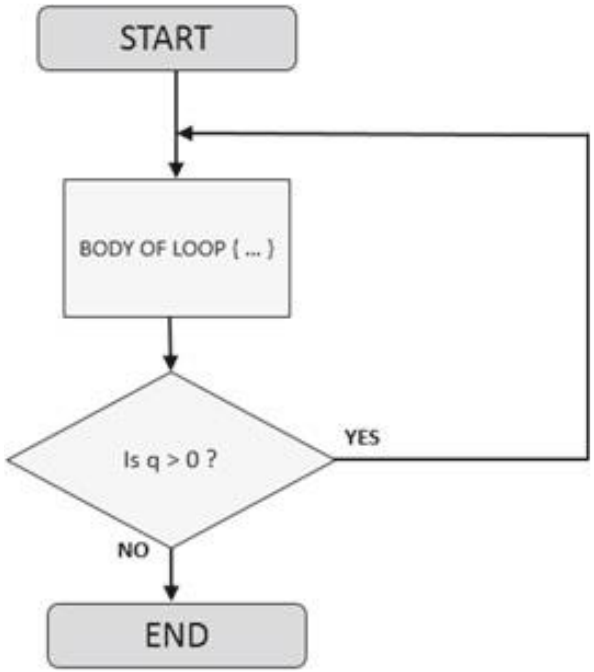
This loop is often used in the context of user interfaces and user interaction, where a user needs to input some value before the value can be determined as acceptable or not. If the value is not within some acceptable bounds, the loop runs again to invite the user to try again.

- a code example and corresponding flowchart

```
public void doWhileLoop()
        {
        // q has scope of the method
        int q = 5;
        do
                {
                System.out.println(q);
                q--;
                }
        while (q > 0);
        }
```

- Output:

```
5
4
3
2
1
```



Nested For Loops

- Nested loop:
  - A loop placed inside another loop.

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.print("*");
    }
    System.out.println();     // to end the line
}
```

- Output:

```
**********
**********
**********
**********
**********
```

- The outer loop repeats 5 times; the inner one 10 times.

Nested for loop exercise

- What is the output of the following nested for loops?

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

- Output:

```
*                   i=1, j=1
**                  i=2, j=1,2
***                 i=3, j=1,2,3
****                i=4, j=1,2,3,4
*****               i=5, j=1,2,3,4,5
```

Nested for loop exercise

- What is the output of the following nested for loops?

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(i);
    }
    System.out.println();
}
```

- Output:

```
1                       i=1, j=1
22                      i=2, j=1,2
333                     i=3, j=1,2,3
4444                    i=4, j=1,2,3,4
55555                   i=5, j=1,2,3,4,5
```

Common Errors

- Both of the following sets of code produce infinite loops:

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; i <= 10; j++) {
          System.out.print(i);
    }
    System.out.println();
}
```

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; i <= 10; i++) {
          System.out.print(i);
    }
    System.out.println();
}
```

Loop Pattern Exercise

- What statement in the body would cause the loop to print:

  `2  7  12  17  22`

- To see patterns, make a table of `count` and the numbers.

Count	Numbers
1	2
2	7
3	12
4	17
5	22

ax+b=y → a×count+b=y → $\begin{bmatrix} a+b=2 \\ 2a+b=7 \end{bmatrix}$
→ solve the equation → 5×count-3

- Statement in the loop:

```
for (int count = 1; count <= 5; count++) {
    System.out.print(5 * count - 3 + " ");
}
```

- Any alternative implementation?

```
for (int count = 2; count <= 22; count=count+5) {
        System.out.print(count + " ");
}
```

Loop Pattern Exercise

- What statement in the body would cause the loop to print:

```
4   7   10   13   16
```

```
for (int count = 1; count <= 5; count++) {
    System.out.print( … );
}
```

- Statement in the loop:

```
for (int count = 1; count <= 5; count++) {
    System.out.print(3 * count + 1 + " ");
}
```

- Any alternative implementation?

```
for (int count = 4; count <= 16; count=count+3) {
        System.out.print(count + " ");
}
```

Loop Pattern Exercise

- What statement in the body would cause the loop to print:

```
17   13   9   5   1
```

```
for (int count = 1; count <= 5; count++) {
    System.out.print( … );
}
```

- Statement in the loop:

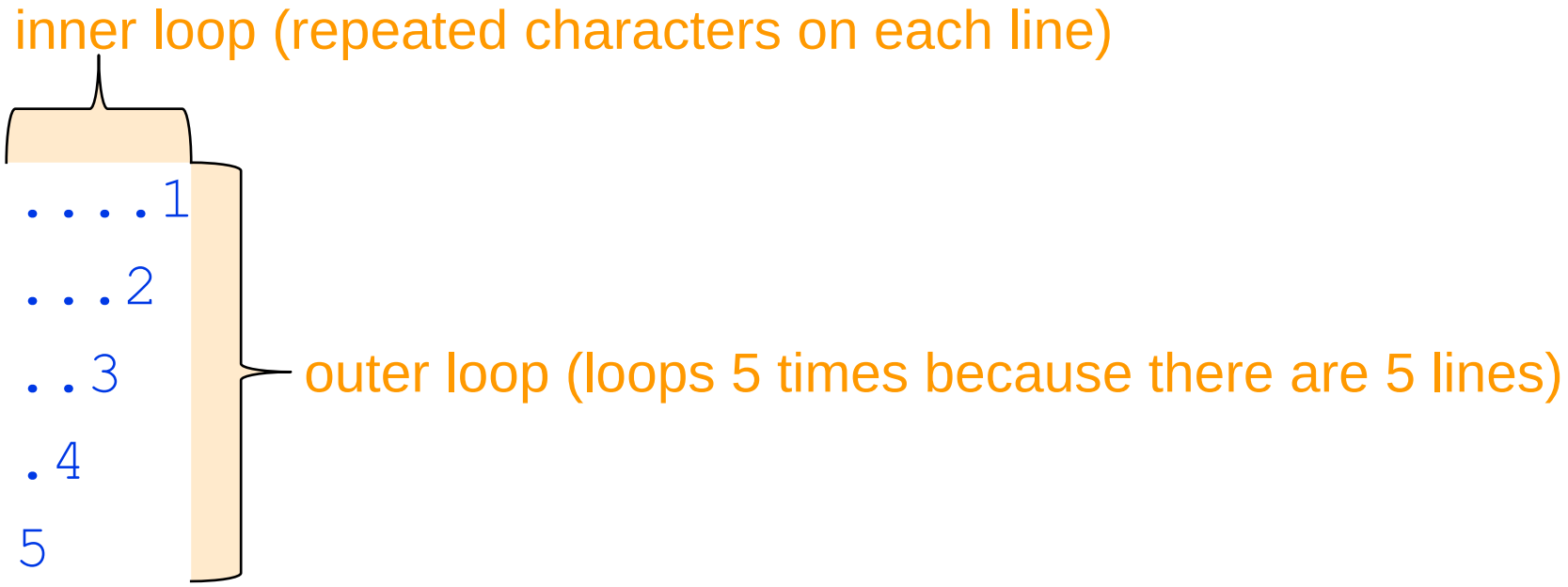
```
for (int count = 1; count <= 5; count++) {
    System.out.print(-4 * count + 21 + " ");
}
```

- Any alternative implementation?

```
for (int count = 17; count >= 0; count=count-4) {
        System.out.print(count + " ");
}
```

Loop Pattern Exercise

- What nested `for` loops produce the following output?



- We must build multiple complex lines of output using:
  - an outer "vertical" loop for each of the lines
  - inner "horizontal" loop(s) for the patterns within each line

Loop Pattern Exercise

- First write the outer loop, from 1 to the number of lines.

```
for (int line = 1; line <= 5; line++) {
    ...
}
```

- Now look at the line contents. Each line has a pattern:
  - some dots (0 dots on the last line), then a number

```
....1
...2
..3
.4
5
```

  - Observation:
    - the number of dots is related to the line number.

Loop Pattern Exercise

- Make a table to represent any patterns on each line.

```
....1
...2
..3
.4
```

Line	Number of Dots
1	4
2	3
3	2
4	1
5	0

- Pattern: -1×line+5

- To print a character multiple times, use a `for` loop.

```
for (int j = 1; j <= 4; j++) {
    System.out.print(".");          // 4 dots
}
```

Loop Pattern Exercise

- Answer:

```
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");
    }
    System.out.println(line);
}
```

- Output:

```
....1
...2
..3
.4
5
```

Loop Pattern Exercise

- What is the output of the following nested `for` loops?

```
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");
    }
    for (int k = 1; k <= line; k++) {
        System.out.print(line);
    }
    System.out.println();
}
```

- Answer:

```
....1
...22
..333
.4444
55555
```

Loop Pattern Exercise

- Modify the previous code to produce this output:

```
....1
...2.
..3..
.4...
5....
```

- Answer:

```
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");
    }
    System.out.print(line);
    for (int k = 1; k <= line-1; k++) {
        System.out.print(".");
    }
    System.out.println();
}
```

Any Questions?