

Loops

Loops

- **while** loop
- **do-while** loop
- **for** loop
- **Nested loops**

Repetition Statements

- **Repetition statements** allow us to execute a statement (or statements) multiple times.
- **Repetition statements** are often referred to as **loops**.
- Like conditional statements, **loops** are controlled by *boolean expressions*
- Java has three kinds of **looping** structures:
 - while** loop
 - do-while** loop
 - for** loop
- The programmer should choose the right kind of loop for the situation.

while Loop

- The most general one of loop statements is **while** statement.

```
while ( boolean-expression )  
statement
```

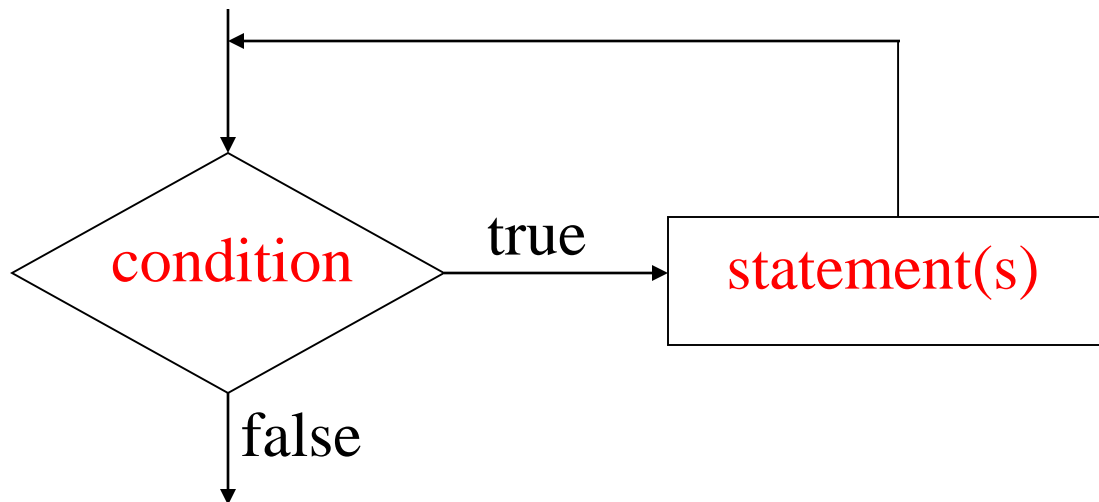
where *statement* can be any statement including a compound statement, an if-statement, another loop statement, ...

- Statement is repeated as long as the **condition** (*boolean-expression*) is **true**.
- When the **condition** gets **false**, the statement is not executed anymore.

while Statement – Flow Diagram

```
while ( condition )  
statement
```

```
while ( condition ) {  
statements  
}
```



- While the condition is true, the statements will execute repeatedly.

while Loop

```
while ( condition )  
    statement
```

- Statement is repeated as long as the **condition** (*boolean-expression*) is **true**.
 - When the **condition** gets **false**, the statement is not executed anymore.
- Somewhere in the loop, some operations should make the **condition false** so that the loop can end.
- If the *condition never gets false* → *infinite loop*
- If the *condition gets immediately false* → *the loop will be repeated zero times*.

while Loop – Example

```
int product = 2;
```

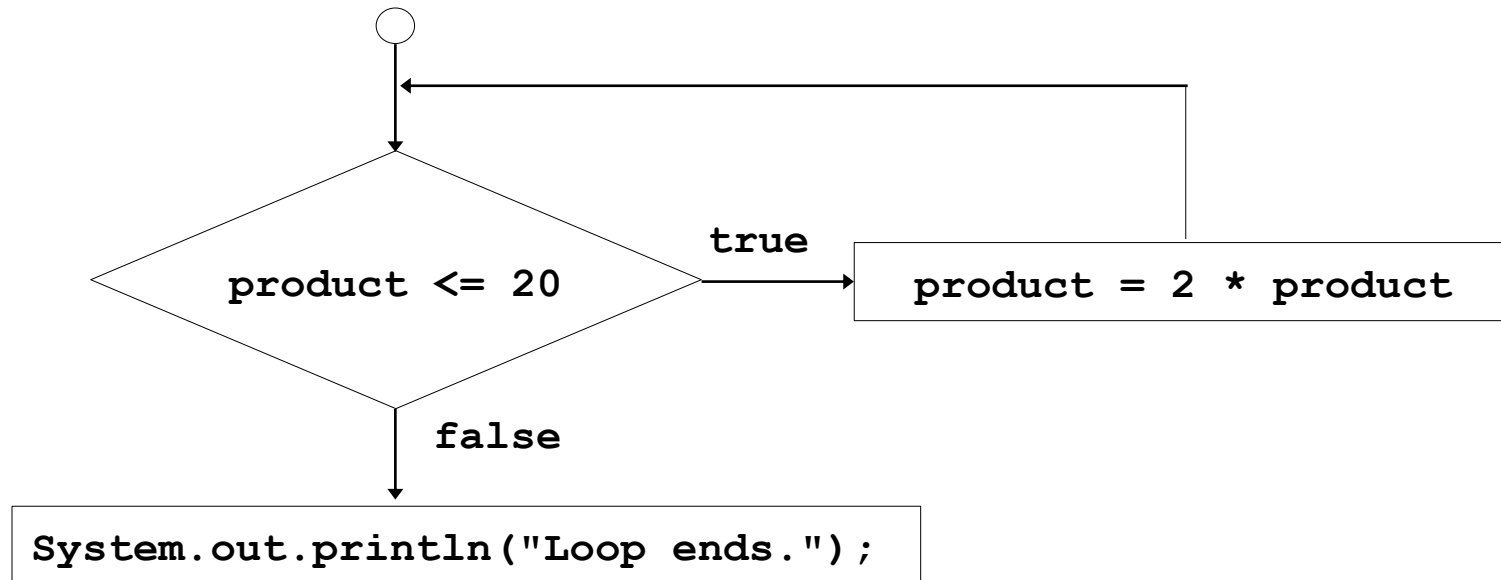
```
while ( product <= 20 )
```

```
    product = 2 * product;
```

```
System.out.println("Loop ends.");
```

loop condition

loop body



while Loop with multiple statements – Example

```
int number = 1;
```

```
while (number < 5)
```

```
{  
    System.out.println("Hello");  
    number++;  
}
```

```
System.out.println("after loop.");
```

loop body




Infinite Loops

- In order for a `while` loop to end, the **condition** must become false.
- A statement in the loop eventually must make the **condition false**.
- If not, it is called an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error.
- You should always double check the logic of a program to ensure that your loops will terminate normally.

Infinite Loops

- The following loop will not end:

```
int x = 20;
while(x > 0)
{
    System.out.println("x is greater than 0: " + x);
}
    x--;
```




- The variable **x** never gets decremented so it will always be greater than 0.
- Adding the **x--;** above fixes the problem.

Infinite Loops

- The following loop will not end:

```
int count = 1;
while (count <= 25)
{
    System.out.println("count is " + count);
    count = count - 1;
}
```



```
count = count + 1;
```

Infinite Loops

- The following loop will not end:

```
int count = 1;
```

```
while (count != 50)
```

```
{
```

```
    System.out.println("count is " + count);
```

```
    count = count + 2;
```

```
}
```

```
while (count <= 50)
```



while Loop for Input Validation

- *Input validation* is the process of ensuring that user input is valid.

```
System.out.print("Enter a number in the range of 1 through 100:");  
number = keyboard.nextInt();  
// Validate the input.  
while (number < 1 || number > 100)  
{  
    System.out.println("That number is invalid.");  
    System.out.print("Enter a number in the range of 1 through 100:");  
    number = keyboard.nextInt();  
}
```

Counter-Controlled Loop

- A loop that repeats a specific number of times is known as a *counter-controlled loop*.
 - its condition depends on the value of a **counter variable**.
- A counter-controlled loop possesses three elements:
 1. It should *initialize a **control variable** to a starting value*.
 2. It should *test the **control variable** by comparing it to a maximum (or minimum) value*. When the control variable reaches its maximum (minimum) value, the loop terminates.
 3. It should *update (increment or decrement) the **control variable***.

while Statement: Counter-Controlled Loop

- Print 5 asterisk characters
- The loop will be repeated for 5 times (for count values: 0,1,...,4)

```
int count = 0;
while ( count < 5 )
{
    System.out.println("*");
    count = count + 1;
}
System.out.println("done");
```

```
*
*
*
*
*
done
```

while Statement: Counter-Controlled Loop

- Read and sum 5 integer values

```
int num;
int sum = 0;
int count = 0;
while ( count < 5 )
{
    System.out.print("Enter an integer: ");
    num = keyboard.nextInt();
    sum = sum + num;
    count = count + 1;
}
System.out.println("sum is " + sum);
```

```
Enter an integer: 5
Enter an integer: 3
Enter an integer: 7
Enter an integer: 4
Enter an integer: 1
sum is 20
```


Example: Counter-Controlled Repetition

- A class of 10 students took a quiz.
- The grades are integers in the range 0 to 100.
- Determine the class average on the quiz

The algorithm:

Set total to zero

Set grade counter to one

While grade counter is less than or equal to 10

Input the next grade

Add the grade into the total

Add one to the grade counter

Set the class average to the total divided by ten

Print the class average

Example: Counter-Controlled Repetition

The Java Program

```
import java.util.*;
/* Class average program with counter-controlled repetition */
public class ClassAverage {
    public static void main(String[] args){
        int counter, total, grade;
        double average;
        Scanner keyboard = new Scanner(System.in);
        total = 0;
        counter = 1;
        while (counter <= 10) {
            System.out.print("Enter grade: ");
            grade = keyboard.nextInt();
            total = total + grade;
            counter = counter + 1;
        }
        average = total / 10.0;
        System.out.println("Class average is " + average);
    }
}
```

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81.7
```

Sentinel-Controlled Loops

- What happens if we don't know how many times a loop will run.
- Ex: a loop which reads the scores of the students in an exam and find the average of the scores;
 - we don't know the number of students.
 - How are we going to stop the loops?
 - ➔ use a **sentinel-value**
- We choose a sentinel-value which can not be a score (e.g. -1)
- We read the scores until this sentinel-value has been entered.
 - When this sentinel-value has been read, we stop the loop.

A Similar Problem: Sentinel-Controlled Loops

- Problem becomes:

Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

- Unknown number of students
 - How will the program know to end?
-
- Use sentinel value
 - Also called signal value, dummy value, or flag value
 - Indicates “end of data entry.”
 - Loop ends when user inputs the sentinel value
 - Sentinel value chosen so it cannot be confused with a regular input (such as **-1** in this case)

A Similar Problem: Solution

```
import java.util.*;
/* Finding the average of an arbitrary number of grades */
public class ClassAverageArbitrary{
    public static void main(String[] args){
        int counter, total, grade;
        double average=0;
        Scanner keyboard = new Scanner(System.in);
        total = 0;
        counter = 0;

        System.out.print("Enter grade, -1 to end:");
        grade = keyboard.nextInt();
        while ( grade != -1 ) {
            total = total + grade;
            counter = counter + 1;
            System.out.print("Enter grade, -1 to end:");
            grade = keyboard.nextInt();
        }
        if (counter != 0)
            average = (double)total / counter;

        System.out.println("Class average is " + average );
    }
}
```

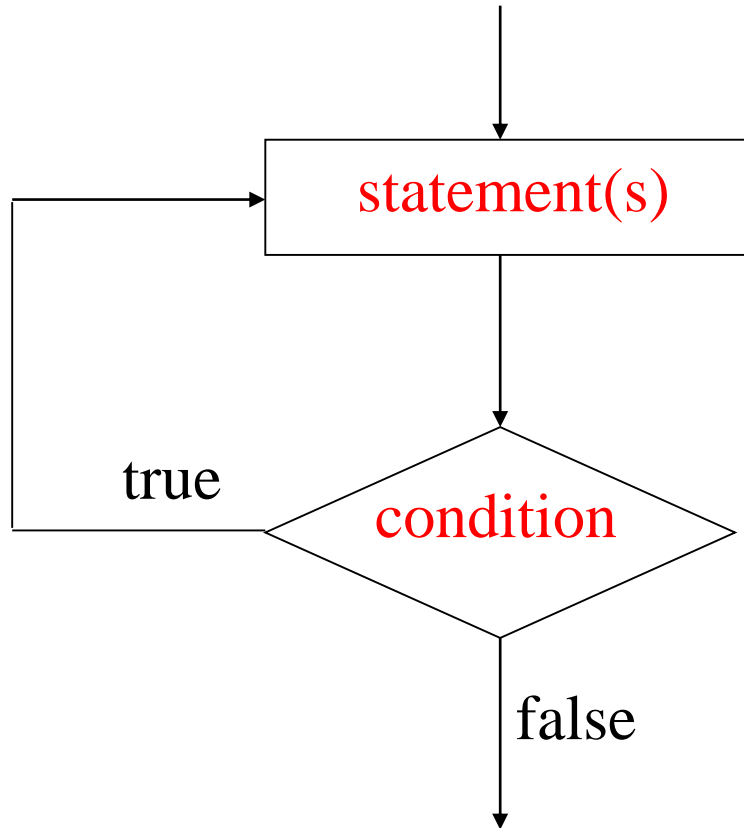
```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.5
```

do/while Loop

- The **do/while** loop
 - Similar to the **while** loop
 - The condition of the loop is tested after the body of the loop is performed.
 - The loop body is performed at least once.

```
do {  
    statement (s)  
} while ( condition );
```

do-while statement – Flow Diagram



do-while loop – Example

```
public class DoWhileTest {  
    public static void main(String[] args) {  
        int counter = 1;  
        do {  
            System.out.print( counter + " " );  
            counter = counter + 1;  
        } while ( counter <= 10 );  
    }  
}
```

Program Output:

1 2 3 4 5 6 7 8 9 10

do-while loop – Example

```
import java.util.Scanner;
public class CircleArea {
    public static void main(String[] args) {
        double radius, area;
        String input;
        char repeat;
        Scanner keyboard = new Scanner(System.in);
        do {
            System.out.print("Enter radius: ");
            radius = keyboard.nextDouble();
            keyboard.nextLine();
            area = radius*radius*3.14;
            System.out.println(
                "The area of the circle with radius "
                + radius + " is " + area);
            System.out.println("\nDo you want continue?");
            System.out.print("Enter Y or N : ");
            input = keyboard.nextLine(); // Read line.
            repeat = input.charAt(0); // Get first char.
        } while (repeat == 'Y' || repeat == 'y');
    }
}
```

Enter radius: 10

The area of the circle with radius 10.0 is 314.0

Do you want continue?

Enter Y or N : y

Enter radius: 5

The area of the circle with radius 5.0 is 78.5

Do you want continue?

Enter Y or N : Y

Enter radius: 4

The area of the circle with radius 4.0 is 50.24

Do you want continue?

Enter Y or N : n

for Loop

- Another loop statement in Java is **for-statement**.
- **for-statement** is more suitable for *counter-controlled loops*.
- The `for` loop allows the programmer to initialize a control variable, test a condition, and modify the control variable all in one line of code.

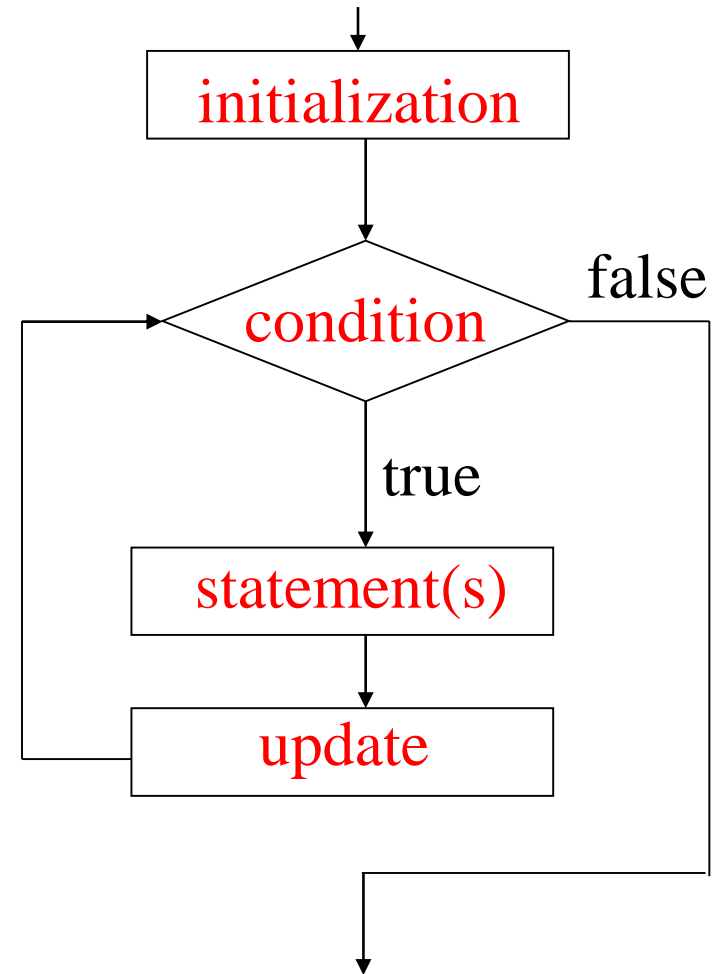
```
for ( initialization ; test ; update ) {  
    statement(s) ;  
}
```

```
for ( initialization ; test ; update )  
    statement
```

for Statement – Flow Diagram

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

- The *initialization section* of the for loop allows the loop to initialize its own control variable.
- The *test section* of the for statement acts in the same manner as the condition section of a while loop.
- The *update section* of the for loop is the last thing to execute at the end of each loop.



Sections of `for` Loop

initialization:

- Typically, `for` loops initialize a counter variable that will be tested by the test section of the loop and updated by the update section.
- The initialization section can initialize multiple variables.
- Variables declared in this section have scope only for the `for` loop.

test:

- Typically, the boolean expression in the test section tests the value of the control variable.

update :

- The update expression is usually used to increment or decrement the counter variable(s) that are initialized in the initialization section of the `for` loop.
- The update section of the loop executes last in the loop.
- The update section may update multiple variables.

for Statement

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

is equivalent to

```
initialization ;  
while ( test ) {  
    statement(s) ;  
    update ;  
}
```

for Statement – Example

```
for(counter = 1; counter <= 10; counter++ )  
    System.out.print( counter + " " );
```

- Prints the integers from one to ten. **1 2 3 4 5 6 7 8 9 10**
- This for loop is equivalent to

```
counter = 1;  
while(counter <= 10) {  
    System.out.print( counter + " " );  
    counter++;  
}
```

for Statement – Example

```
int i;  
int sum = 0;  
for (i=1; i<=20; i++)  
    sum = sum + i;
```



```
int i;  
int sum = 0;  
i = 1;  
while (i<=20) {  
    sum = sum + i;  
    i++;  
}
```

```
int i;  
int sum = 0;  
for (i=100; i>=1; i--)  
    sum = sum + i;
```



```
int i;  
int sum = 0;  
i = 100;  
while (i>=1) {  
    sum = sum + i;  
    i--;  
}
```

Multiple Initializations and Updates

- The `for` loop may initialize and update multiple variables.
 - **Multiple initializations** and **multiple update expressions** are comma-separated lists.

```
for ( i = 0, j = 0; i+j <= 10; i++, j++ )  
    System.out.println( i+j );
```

Output:

```
0  
2  
4  
6  
8  
10
```


Control Variable Declarations

- **Control (loop) variables** *can be also declared in the initialization section.*
 - In this case, the scope of those variables will be only that for loop structure.
 - i.e. That declaration will be only valid in that loop structure.
- Or, **control (loop) variables** can be declared at the beginning of the method.
 - The scope of those variables will be only that method.

```
for (int i=1; i<=10; i++)  
    System.out.print(i+" ");
```

```
int i;  
for(i = 1; i<= 10; i++ )  
    System.out.print(i+" ");
```

Control Variable Declarations

```
for (int i=0, j=0; i+j<=10; i++, j++)  
    System.out.println(i+j);
```

```
int i, j;  
for (i=0, j=0; i+j<=10; i++, j++)  
    System.out.println(i+j);
```

Scope

scope: The part of a program where a variable exists.

- From its declaration to the end of the { } braces
- A variable declared in a `for` loop exists only in that loop.

```
public static void main(String[] args) {  
    int x = 3;  
    for (int i = 1; i <= 10; i++) {  
        System.out.println(x);  
    }  
    // i no longer exists here  
} // x ceases to exist here
```

i's scope {

x's scope }

Scope implications

- Variables without overlapping scope can have same name.

```
for (int i = 1; i <= 100; i++) {  
    System.out.print("\\");  
}  
int i = 5;                // OK: outside of loop's scope
```

- A variable can't be used out of its scope.

```
for (int i = 1; i <= 100 * line; i++) {  
    System.out.print("/");  
}  
i = 4;                    // ERROR: outside scope
```

for Loop – Example

- Adding even numbers

```
/* Summation of even numbers up to 100 with for */  
int sum = 0;  
for ( int number = 2; number <= 100; number += 2 )  
    sum += number;  
System.out.println("Sum is " + sum );
```

Output:

Sum is 2550

for Loop – Example

- Printing Celsius and Fahrenheit table.

```
for ( int i = 0; i <= 5; i++ )  
    System.out.println( i + "    " + (i * 1.8 + 32) );
```

Output:

```
0    32.0  
1    33.8  
2    35.6  
3    37.4  
4    39.2  
5    41.0
```

Nested Loops

- The body of a loop can contain any kind of statements, including another loop.
- When *a loop body includes another loop construct* this is called a *nested loop*.
- In a nested loop structure the inner loop is executed from the beginning *every time the body of the outer loop is executed*.

Example 1:

```
int value = 0;
for (int i=1; i<=10; i=i+1)
    for (int j=1; j<=5; j=j+1)
        value = value + 1;
```

- How many times the inner loop is executed?
→ value is $10 * 5 = 50$

Nested Loops

Example 2:

```
int value = 0;
for (int i=1; i<=10; i=i+1) {
    for (int j=1; j<=i; j=j+1) {
        value = value + 1;
    }
}
```

How many times the inner loop is executed?

→ value is $1+2+3+4+5+6+7+8+9+10=55$

Nested Loops

Example 3:

```
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 Loops - Printing a triangle

- Write a program to draw a triangle like the following:

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

- We can use a nested for-loop:

```
public class NestedLoop {  
    public static void main(String[] args) {  
        int numOfLines = 5;  
        for (int i=1; i<=numOfLines; i++) {  
            for (int j=1; j<=i; j++) {  
                System.out.print("*");  
            }  
            System.out.println();  
        }  
    }  
}
```

Nested Loops – Example

- A triangle shape (1st line contains 1 star, 2nd line contains 3 star, ... , nth line contains 2n-1 stars)

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

```
import java.util.Scanner;
public class NestedLoop2 {
    public static void main(String[] args) {
        int n;
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Number of lines: ");
        n = keyboard.nextInt();
        for (int i=1; i<=n; i++) { // for each line
            for (int j=1; j<=(n-i); j++)
                System.out.print(" ");
            for (int j=1; j<=(2*i-1); j++)
                System.out.print("*");
            System.out.println();
        }
    }
}
```

Nested Loops – Example

- Nesting can be more than one level

```
int sum=0;
```

```
for (int i=1; i<=5; i++)
```

```
    for (int j=1; j<=5; j++)
```

```
        for (int k=1; k<=5; k++)
```

```
            sum=sum+1;
```

➔ sum is 125

Loop - Exercises

- Write a program segment that computes $N!$.

```
int fact = 1;
for (int j = 1; j <= n; j++)
    fact = fact * j;
```

- Write a program segment for finding the sum of the first N terms of the series $1 + 1/2 + 1/3 + \dots$

```
double sum = 0;
for (int j = 1; j <= n; j++)
    sum = sum + 1.0 / j;
```

Loop - Exercises

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

inner loop (repeated characters on each line)

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

outer loop (loops 5 times
because there are 5 lines)

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

Loop - Exercises

- 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.**

**there are 4 dots in line 1 (# of dots= 5 - 1);
there are 3 dots in line 2 (# of dots= 5 - 2);**

...

→ There must be (5-i) dots in line i.

Loop - Exercises

Answer:

```
// repeat for each line
for (int line = 1; line <= 5; line++) {
    // print dots in the line
    for (int j = 1; j <= (5 - line); j++) {
        System.out.print(".");
    }
    // print line number and move to the next line
    System.out.println(line);
}
```

Output:

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


Loop Exercises – Program 1

```
// A program that reads numbers until a negative number is read and
// prints number of values read, largest value and smallest value
import java.util.Scanner;
public class MaxMinVal {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int num, maxVal, minVal, numOfVals = 0;
        System.out.println("Enter a sequence of positive numbers ending with -1: ");
        num = keyboard.nextInt();
        minVal = num;
        maxVal = num;
        while (num >= 0) {
            numOfVals++;
            if (num < minVal)
                minVal = num;
            else if (num > maxVal)
                maxVal = num;
            num = keyboard.nextInt();
        }
        System.out.println("Number of values: " + numOfVals +
            "\nLargest value : " + maxVal +
            "\nSmallest value: " + minVal);
    }
}
```

```
Enter a sequence of positive numbers ending with -1:
5 2 3 6 1 8 3 2 4 -1
Number of values: 9
Largest value : 8
Smallest value: 1
```

Loop Exercises – Program 2

```
// A program to find the nth Fibonacci number.
import java.util.Scanner;
public class FibonacciNum {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int n, prevnum1, prevnum2, temp;
        System.out.print("Enter a positive integer: ");
        n = keyboard.nextInt();
        if (n==0 || n== 1)
            System.out.println(n + " th fibonacci number is " + n);
        else {
            prevnum2 = 0;
            prevnum1 = 1;
            for (int j=2; j <= n; j++) {
                temp = prevnum1 + prevnum2;
                prevnum2 = prevnum1;
                prevnum1 = temp;
            }
            System.out.println(n + " th fibonacci number is " + prevnum1);
        }
    }
}
```

Enter a positive integer: 4
4 th fibonacci number is 3

Loop Exercises – Program 3

```
// A program to test whether a given positive integer (> 1)
// is a prime number or not
import java.util.Scanner;
public class PrimeNum {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int n;
        boolean flag;
        System.out.print("Enter a positive integer > 1: ");
        n = keyboard.nextInt();
        if (n == 2)
            System.out.println(n + " is a prime number.");
        else if (n%2 == 0)
            System.out.println(n + " is NOT a prime number.");
        else {
            flag = true;
            for (int i=3; i<(n/2) && flag; i += 2) {
                if (n%i == 0)
                    flag = false;
            }
            if (flag)
                System.out.println(n + " is a prime number.");
            else
                System.out.println(n + " is NOT a prime number.");
        }
    }
}
```

Enter a positive integer > 1: 53
53 is a prime number.