**Comprehensive Revision Notes**

**Class Outline**

This class covers several fundamental concepts in Java programming, including the use of loops (for and while), the break and continue statements, scope of variables, and basic control structures. Below are the detailed revisions notes including code examples and explanations of each concept.

**1. For Loop in Java**

The for loop in Java is a control flow statement that allows code to be executed repeatedly based on a condition. It's especially useful for iterating over a range of values.

**Syntax**:

for (initialization; condition; increment/decrement) {

// Statements to be executed multiple times

}

**Example: Print "Hello" five times**:

for (int count = 1; count <= 5; count++) {

System.out.println("Hello");

}

**Explanation:**

1. **Initialization**: int count = 1 initializes the loop control variable.
2. **Condition**: count <= 5 checks if the loop should continue.
3. **Increment**: count++ updates the loop control variable.

**2. While Loop in Java**

The while loop in Java repeatedly executes a block of statements as long as a given condition is true.

**Syntax**:

while (condition) {

// Statements to be executed

}

**Comparison with For Loop**: Both for and while loops can achieve the same goal, but for loops are generally used when the number of iterations is known before entering the loop, while while loops are used when the iteration count is not known upfront.

**3. The break and continue Statements**

* **Break Statement**: Used to terminate the nearest enclosing loop prematurely.

**Example**:

for (int i = 1; i <= 10; i++) {

if (i == 5) {

break;

}

System.out.println(i);

}

This code will print numbers 1 to 4. When i becomes 5, the loop is terminated due to the break statement.

* **Continue Statement**: Skips the current iteration of the loop and moves to the next iteration.

**Example**:

for (int i = 1; i <= 10; i++) {

if (i % 2 != 0) {

continue;

}

System.out.println(i);

}

This code will print all even numbers between 1 and 10 because the continue statement skips the iteration whenever i is odd.

**4. Scope of Variables**

The scope of a variable in Java refers to the region of the program where the variable can be accessed.

* **Block Scope**: Variables declared inside a block (within {}) are only accessible within that block.
* **Method Scope**: Variables declared inside a method are accessible throughout the method.
* **Class Scope**: Variables declared as fields of a class are accessible throughout the class.

**Example**:

public class ScopeDemo {

public static void main(String[] args) {

int x = 10; // x is accessible throughout main method

if (x == 10) {

int y = 20; // y is accessible only within this block

System.out.println(x + y);

}

// System.out.println(y); // This would cause an error, y is not accessible here

}

}

**5. Prime Number Check**

A prime number is a number greater than 1 that has no positive divisors other than 1 and itself.

**Algorithm**:

1. Iterate over all numbers from 2 to √N to check if N is divisible by any of these.
2. If N is divisible by any number other than 1 and itself, it is not a prime.
3. Optimized with the break statement to exit early on finding a divisor.

**Example Code**:

public class PrimeCheck {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a number: ");

int N = scanner.nextInt();

boolean isPrime = true;

for (int i = 2; i <= Math.sqrt(N); i++) {

if (N % i == 0) {

isPrime = false;

break;

}

}

if (isPrime && N > 1) {

System.out.println(N + " is a prime number.");

} else {

System.out.println(N + " is not a prime number.");

}

scanner.close();

}

}

**Explanation:**

* This checks divisibility up to √N, reducing the number of checks.
* The break statement exits the loop early if a divisor is found.

**6. Handling Multiple Test Cases**

In many competitive programming scenarios, the program needs to handle multiple test cases.

**Approach**:

1. Read the number of test cases T.
2. For each test case, read the input and apply the required logic.

**Example Code**:

public class MultipleTestCases {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int T = scanner.nextInt(); // Read number of test cases

for (int t = 1; t <= T; t++) {

int N = scanner.nextInt(); // Read a number for each test case

// Apply some logic to N here

System.out.println(N); // Example output

}

scanner.close();

}

}

This code handles multiple test cases by looping T times and applying the logic to each test case individually.

**Conclusion**

This class provided a comprehensive overview of fundamental programming concepts in Java, including loops, control flow statements, variable scope, and basic algorithms. These foundations are critical for writing efficient and effective programs.

**Practice Problems**:

1. Write a program to print the multiplication table for a given number.
2. Modify the prime check algorithm to print all prime numbers up to N.
3. Implement a program to calculate the factorial of a number using both for and while loops.

FOR LOOP:

A control flow statement that allows code to be executed repeatedly based on a condition.

WHILE LOOP:

A control flow statement that executes code as long as a condition remains true.

INITIALIZATION:

The initial setup in a loop where variables are defined.

CONDITION:

A statement loop that determines if the loop should continue.

UPDATION:

Updating the loop control variable to eventually terminate the loop.

NESTED LOOP:

A loop inside another loop.

COUNT:

A variable that keeps track of iterations or events.

BREAK STATEMENT:

Terminates the nearest enclosing loop prematurely.

CONTINUE STATEMENT:

Skips the rest of the current loop iteration and moves to next iteration.

SCOPE:

The region in the code where a variable is accessible.

PRIME NUMBER:

A number that has exactly two distinct positive divisors, 1 and itself.

FACTOR:

A number or algebraic expression that divides another number or expression evenly.

<https://github.com/KingsGambitLab/Lecture_Notes/blob/non-dsa/Academy%20DSA%20Typed%20Notes/Java%20Refresher/Refresher%20For%20Loop.md>

FACTOR:

Numbers that divide another number exactly with no remainder.

PRIME NUMBERS:

Numbers with exactly two distinct positive divisors: 1 and the number itself.

PERFECT NUMBERS:

Numbers that are equal to the sum of their proper divisors.

REVERSE A NUMBER:

Process of changing the order of digits of a number from last to first.

LOOP:

A programming construct that repeats a set of instruction until a specified condition is met.

MODULUS OPERATION:

An operation that returns the remainder of the division of one number by another.

FOR LOOP:

A control flow statement for specifying iteration that allows code to be executed repeatedly.

WHILE LOOP:

A control flow statement that allows code to be executed repeatedly based on a Boolean condition.

COUNTING DIGITS:

The process of determining the number of digits in a number using division and modulus.

PERFECT SQUARE:

Integers that are square numbers, i.e., the product of an integer with itself.

SUM OF DIGITS:

The process of adding all the individual digits of a number together.

INFINITE LOOP:

A sequence of instruction in a program that loops endlessly without any condition to stop.

For (1: initialization, 2: Condition, 4: Updation)

{

3: //loop work

}

public class Main {

  public static void main(String[] args) {

    Scanner ip=new Scanner(System.in);

    // Print 1 to N

    int num=ip.nextInt();

    for(int i=1;i<=num;i++) System.out.println(i);

    System.out.println("-----------------------");

    //print odd 1 to N

    int num2=ip.nextInt();

    for(int i=1;i<=num2;i+=2) System.out.println(i);

    System.out.println("-----------------------");

  }

}

CUSTOM INPUT:

3

3

OUTPUT:

1

2

3

-----------------------

1

3

-----------------------

Factors:

X is a factor of N, if N multiple of X.

6= 1, 2, 3, 6

10= 1, 2, 5, 10

24= 1, 2, 3, 4, 6, 8, 12, 24

N= factors for N is with in range [1,N]

public class Main {

  public static void main(String[] args) {

    Scanner ip=new Scanner(System.in);

    // Print factors of N

    int num=ip.nextInt();

    for(int i=1;i<=num;i++)if(num%i==0) System.out.println(i);

  }

}

CUSTOM INPUT:

6  
OUTPUT:

1

2

3

6

Prime Number:

A number with exactly two factors, factors are 1 and number itself.

1 is not prime number.

public class Main {

  public static void main(String[] args) {

    Scanner ip=new Scanner(System.in);

    // check if number is prime or not

    int num=ip.nextInt(), count=0;

    for(int i=1;i<=num;i++)

    if(num%i==0)

    {

      count++;

      if(count>2)break;

    }

    System.out.println(count==2?"Prime":"Not Prime");

  }

}

Break statement:

It is for stopping immediate parent loop.

The **Sieve of Eratosthenes** is a very famous and efficient algorithm for finding **all prime numbers up to a given limit N**.

* Write down all numbers from **2 to N** (since 1 is not prime).

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

* Start with **2** (first prime). Cross out all multiples of 2 (except 2 itself).

2 3 ~4~ 5 ~6~ 7 ~8~ 9 ~10~ 11 ~12~ 13 ~14~ 15 ~16~ 17 ~18~ 19 ~20~ 21 ~22~ 23 ~24~ 25 ~26~ 27 ~28~ 29 ~30~

* Next number that is not crossed out is **3** → prime. Cross out multiples of 3.

2 3 ~4~ 5 ~6~ 7 ~8~ ~9~ ~10~ 11 ~12~ 13 ~14~ ~15~ ~16~ 17 ~18~ 19 ~20~ ~21~ ~22~ 23 ~24~ ~25~ ~26~ ~27~ ~28~ 29 ~30~

* Next number that is not crossed out is **5** → prime. Cross out multiples of 5.

2 3 5 7 11 13 17 19 23 29

* Continue this process up to √N (square root of N).

Why √N? Because any composite number > √N must have a smaller factor ≤ √N.

* The remaining uncrossed numbers are **all primes**.

Continue Statement:

Skips to the next iteration.

public class Main {

  public static void main(String[] args) {

    Scanner ip=new Scanner(System.in);

    // print odd number till n using continue statement

    int num=ip.nextInt();

    for(int i=1;i<=num;i++)if(i%2==0)continue; else System.out.println(i);

  }

}

CUSTOM INPUT:

8

OUTPUT:

1

3

5

7

Q: Given T testcases, each testcase has an integer N as input, print its last digit

public class Main {

  public static void main(String[] args) {

    Scanner ip=new Scanner(System.in);

    int numOfTestcases=ip.nextInt();

    for(int i=1;i<=numOfTestcases;i++) System.out.println(ip.nextInt()%10);

  }

}

CUSTOM INPUT:

3

1001

2

304

OUTPUT:

1

2

4

Scope of variable:

Useful lifetime of variable.

{

Int x=10;

{

Int y=15;

}

}

Variable never available outside his curly brace where variable is created.

public class Main {

  public static void main(String[] args) {

    //CASE 1

    int x=10;

    int y=15;

    {

      System.out.println(x+" "+y);

    }

    //CASE 2

    int a=10;

    {

      int b=15;

      System.out.println(a+" "+b);

    }

    {

      //System.out.println(a+" "+b);

      //Above throw error As b don't have scope in this block of code

      /\*

        [CompilationError] Your code was terminated due to compilation error

        Main.java:19: error: cannot find symbol

        System.out.println(a+" "+b);

        symbol:   variable b

        location: class Main

      \*/

    }

    //CASE 3

    int p=10;

    {

      int q=15;

      {

        System.out.println(p+" "+q);

      }

    }

  }

}

OUTPUT:

10 15

10 15

10 15

public class Main {

  public static void main(String[] args) {

    //CASE 1

    int x=10;

    int y=15;

    {

      y=10;

      System.out.println(x+" "+y);

    }

    {

      System.out.println(x+" "+y);

    }

  }

}

OUTPUT:

10 10

10 10

public class Main {

  public static void main(String[] args) {

    //CASE 1

    int x=10;

    int y=15;

    {

      //int y=10; //throw error

      /\*

        Main.java:8: error: variable y is already defined in method main(String[])

        int y=10;

      \*/

      System.out.println(x+" "+y);

    }

    {

      System.out.println(x+" "+y);

    }

  }

}

Unary Operator:

++i

Update and then do the work

i++

do the work and then update

**Q1. Count factors**

**Problem Description**

Take an integer **N** as input and print the count of its factors.  
**The factor** of a number is the number that divides it perfectly leaving no remainder.

Example: 1, 2, 3, and 6 are factors of 6  
**Problem Constraints**

1 <= N <= 300  
**Input Format**

The first and only line of input contains a single integer N.  
**Output Format**

Print the count of factors of N.  
**Example Input**

Input 1:

5

Input 2:

10  
**Example Output**

Output 1:

2

Output 2:

4  
**Example Explanation**

Explanation 1:

Factors of 5 are 1 and 5.

Explanation 2:

Factors of 10 are 1, 2, 5 and 10.

CODE:

public class Main {

    public static void main(String[] args) {

       int num=new Scanner(System.in).nextInt(),sum=0;

       for(int i=1;i<=num;i++)if(num%i==0)sum++;

       System.out.println(sum);

    }

}

**Q2. Is It Prime?**

**Problem Description**

Take an integer **A** as input, you have to tell whether it is a prime number or not.

A prime number is a natural number **greater** than **1** which is divisible only by **1** and **itself**.  
**Problem Constraints**

1 <= **A** <= 106  
**Input Format**

First and only line of the input contains a single integer **A**.  
**Output Format**

Print **YES** if **A** is a prime, else print **NO**.  
**Example Input**

Input 1:

3

Input 2:

4   
**Example Output**

Output 1:

YES

Output 2:

NO   
**Example Explanation**

Explanation 1:

3 is a prime number as it is only divisible by 1 and 3.

Explanation 2:

4 is not a prime number as it is divisible by 2.

CODE:

public class Main {

    public static void main(String[] args) {

        int num = new Scanner(System.in).nextInt(),count=0;

        for(int i=1;i<=num;i++) if(num%i==0) if(count<3) count++; else break;

        System.out.println(count==2?"YES":"NO");

    }

}

CODE2:

public class Main {

    public static void main(String[] args) {

        int c=0;

        int N=new Scanner(System.in).nextInt();

        for(int i=1;i<=N;i++) {

            if(N%i==0) c++;

            if(c>2)break;

        }

        if(c==2)

        System.out.print("YES");

        else

        System.out.print("NO");

    }

}

**Q3. Is It Perfect?**

**Problem Description**

Given the **Number of Test Cases** as **T**,  
For each test case, take an integer **N** as input, you have to tell whether it is a **perfect number or not.**  
  
A perfect number is a positive integer that is **equal** to the sum of its **proper positive divisors** (excluding the number itself).   
A positive proper divisor divides a number **without** leaving any **remainder**.  
**Problem Constraints**

1 <= **T** <= 10

1 <= **N** <= 106  
**Input Format**

The first line of the input contains a single integer **T**.

Each of the **next T lines** contains a single integer **N**.  
**Output Format**

For each testcase, print **YES** if the given integer is perfect, else print **NO**, in a separate line  
**Example Input**

Input 1:

2

4

6

Input 2:

1

3  
**Example Output**

Output 1:

NO

YES

Output 2:

NO  
**Example Explanation**

Explanation 1:

For the first test case A = 4, the answer is "NO" as sum of its proper divisors = 1 + 2 = 3, is not equal to 4.

For the second test case A = 6, the answer is "YES" as sum of its proper divisors = 1 + 2 + 3 = 6, is equal to 6.

Explanation 2:

For the first test case A = 3, the answer is "NO" as sum of its proper divisors = 1, is not equal to 3.

CODE:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        int numOfInt=ip.nextInt();

        for(int i=1;i<=numOfInt;i++){

            int num=ip.nextInt(),sum=0;

            for(int j=1;j<num;j++)if(num%j==0)sum+=j;

            System.out.println(sum==num?"YES":"NO");

        }

    }

}

**Q4. Print N stars**

**Problem Description**

Given an integer **N**, print **N** stars in a single line.

For example if **N = 5** then pattern will be like:

\*\*\*\*\*  
**Problem Constraints**

2 <= N <= 100  
**Input Format**

Single line input contains a single integer **N**.  
**Output Format**

Output **N** stars in a single line.  
**Example Input**

Input 1:

2

Input 2:

3  
**Example Output**

Output 1:

\*\*

Output 2:

\*\*\*  
**Example Explanation**

Print the pattern as described.

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        for(int i=1;i<=num;i++)System.out.print("\*");

    }

}

**Q5. Print a matrix of stars**

**Problem Description**

Given two integers **N** and **M** as inputs, print a rectangle of **N \* M** stars.

For example if **N = 3, M = 4** then pattern will be like:

\*\*\*\*

\*\*\*\*

\*\*\*\*  
**Problem Constraints**

2 <= N, M <= 100  
**Input Format**

First line of input contains an integers **N (no of rows).**

Second line of input contains an integer **M (no of cols).**  
**Output Format**

Output **N \* M** rectangle of stars.  
**Example Input**

Input 1:

2  
 3

Input 2:

3  
 1  
**Example Output**

Output 1:

\*\*\*

\*\*\*

Output 2:

\*

\*

\*  
**Example Explanation**

Print the pattern as described.

CODE:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        int N=ip.nextInt(),M=ip.nextInt();

        for(int i=1;i<=N;i++){

            for(int j=1;j<=M;j++) System.out.print("\*");

            System.out.println();

        }

    }

}

**Q6. Stair Pattern**

**Problem Description**

Take an integer **N** as input, print the corresponding stair pattern for **N**.

For example if **N = 4** then stair pattern will be like:

\*

\*\*

\*\*\*

\*\*\*\*  
**Problem Constraints**

1 <= N <= 100  
**Input Format**

First and only line of input contains a single integer **N**.  
**Output Format**

Output the stair pattern corresponding to the given **N**.  
**Example Input**

Input 1:

2

Input 2:

3  
**Example Output**

Output 1:

\*

\*\*

Output 2:

\*

\*\*

\*\*\*  
**Example Explanation**

Print the pattern as described.

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        for(int i=1;i<=num;i++){

            for(int j=1;j<=i;j++)System.out.print("\*");

            System.out.println();

        }

    }

}

**Q1. LCM – Easy**

**Problem Description**

Implement a program that takes two positive integers **A**and **B**in the input and prints their LCM.

**Definition of LCM :**The Least Common Multiple or LCM of two numbers say A and B, is denoted as LCM (A,B). And the LCM is the smallest or least positive integer that is divisible by both A and B.  
**Problem Constraints**

1 <= A,B <= 200  
**Input Format**

Two space separated integers A and B in the input.  
**Output Format**

Output a single integer that is the LCM of A and B.  
**Example Input**

Input 1:

5 10

Input 2:

2 3  
**Example Output**

Output 1:

10

Output 2:

6  
**Example Explanation**

Explanation 1:

LCM(5,10)=10

Explanation 2:

LCM(2,3)=6

CODE:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        int A=ip.nextInt(),B=ip.nextInt(),result=0;

        for(int i=(A>B?A:B);i<=(A\*B);i++)

            if(i%A==0 && i%B==0){

                result=i;

                break;

            }

        System.out.print(result);

    }

}

CODE2:

public class Main {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        short A=sc.nextShort(),B=sc.nextShort();

        for(int i=A>B?A:B;i<=A\*B;**i=i+(A>B?A:B)){**

            if(i%A==0 && i%B==0){

            System.out.print(i);

            break;

            }

        }

    }

}

**Q2. HCF – Easy**

**Problem Description**

Write a program to input two integers **A & B** from user and print their HCF.

**Definition Of HCF:**The HCF(Highest Common Factor) or the GCD(greatest common divisor) of two positive integers happens to be the largest positive integer that divides the numbers without leaving a remainder.  
**Problem Constraints**

1 <= A,B <= 100000  
**Input Format**

First line will contain 2 integers **A** and **B**  
**Output Format**

An integer which is the HCF of A & B.  
**Example Input**

Input 1:

15 105

Input 2:

24 36   
**Example Output**

Output 1:

15

Output 2:

12

CODE:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        int A=ip.nextInt(),B=ip.nextInt(),result=0;

        for(int i=2;i<=(A<B?A:B);i++) if(A%i==0 && B%i==0) result=i;

        System.out.print(result);

    }

}

Below is the best code

CODE2:

public class Main {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        int A=sc.nextInt(),B=sc.nextInt();

        for(int hcf=A>B?B:A;hcf>=1;hcf--){

            if(A%hcf==0 && B%hcf==0){

            System.out.print(hcf);

            break;

            }

        }

    }

}

**Q3. Armstrong Numbers!**

**Problem Description**

You are given an integer **N** you need to print all the Armstrong Numbers between **1** to **N**. (N inclusive).

If sum of cubes of each digit of the number is equal to the number itself, then the number is called an Armstrong number.

For example, 153 = ( 1 \* 1 \* 1 ) + ( 5 \* 5 \* 5 ) + ( 3 \* 3 \* 3 ).

Note: All the test cases in this problem are limited to 3 digit numbers.  
**Problem Constraints**

1 <= N <= 500  
**Input Format**

First and only line of input contains an integer **N**.  
**Output Format**

Output all the Armstrong numbers in range **[1,N]** each in a new line.  
**Example Input**

Input 1:

5

Input 2:

200  
**Example Output**

Output 1:

1

Output 2:

1

153  
**Example Explanation**

Explanation 1:

1 is an armstrong number.

Explanation 2:

1 and 153 are armstrong number under 200.

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        for(int i=1;i<=num;i++){

            int calculate=i,sum=0;

            for(;calculate>0;calculate/=10){

                int rem=calculate%10;

                sum=sum+(rem\*rem\*rem);

            }

            if(sum==i)System.out.println(i);

        }

    }

}

**Q4. Print the Primes!**

**Problem Description**

You are given an integer **N** you need to print all the **Prime Numbers** between **1** and **N**.

**Prime numbers** are numbers that have only 2 factors: 1 and themselves. For example, the first 5 prime numbers are 2, 3, 5, 7, and 11.  
**Problem Constraints**

1 <= N <= 300  
**Input Format**

First and only line of input contains a single integer **N**.  
**Output Format**

Print all the prime numbers between between **1** and **N** each in a new line.  
**Example Input**

Input 1:

5

Input 2:

10  
**Example Output**

Output 1:

2

3

5

Output 2:

2

3

5

7  
  
**Example Explanation**

Explanation 1:

Prime numbers between [1, 5] are (2, 3, 5).

Explanation 2:

Prime numbers between [1, 10] are (2, 3, 5, 7)

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        for(int i=2;i<=num;i++){

            int count=0;

            for(int j=2;j<i;j++)if(i%j==0)count++;

            if(count==0)

            System.out.println(i);

        }

    }

}

CODE2:

public class Main {

    public static void main(String[] args) {

        short N=new Scanner(System.in).nextShort();

        for(short i=2;i<=N;i++){

            byte count=0;

            for(short j=2;j<=i;j++){

            if(i%j==0) count++;

            if(count>1) break;

            }

            if(count==1)System.out.println(i);

        }

    }

}

**Q5. Numeric Stair Pattern**

**Problem Description**

Take an integer **N** as input, print the corresponding pattern for **N**.  
For example if **N = 4** then pattern will be like:  
1  
1 2  
1 2 3  
1 2 3 4  
**NOTE:** There should be **no extra spaces after last integer**.  
**Problem Constraints**

1 <= N <= 100  
**Input Format**

First and only line of input contains a single integer **N**.  
**Output Format**

Output the pattern corresponding to the given **N**.

**NOTE:**

* There should be**no extra spaces** **after last integer** and **before first integer** in any row.
* All integers in any row in the pattern are **space separated**.

**Example Input**

Input 1:

2

Input 2:

3  
**Example Output**

Output 1:

1

1 2

Output 2:

1

1 2

1 2 3  
**Example Explanation**

Print the pattern as described.

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        for(int i=1;i<=num;i++)

        {

            for(int j=1;j<=i;j++)System.out.print(i==j?j:j+" ");

            System.out.println();

        }

    }

}

**Q6. Inverted Half Pyramid**

**Problem Description**

Given an integer **N**, print the corresponding **Inverted Half Pyramid** pattern for **N**.

For example if **N = 4** then pattern will be like:

\*\*\*\*

\*\*\*

\*\*

\*  
**Problem Constraints**

1 <= N <= 100  
**Input Format**

First and only line of input contains a single integer **N**.  
**Output Format**

Output the Inverted Half Pyramid pattern corresponding to the given **N**.  
**Example Input**

Input 1:

2

Input 2:

3  
**Example Output**

Output 1:

\*\*

\*

Output 2:

\*\*\*

\*\*

\*  
**Example Explanation**

Print the pattern as described.

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        for(int i=num;i>=1;i--){

            for(int j=1;j<=i;j++)System.out.print("\*");

            System.out.println();

        }

    }

}

**Q7. Two Line Star Pattern**

**Problem Description**

Print a pattern consisting of **N rows**, where each row contains **an asterisk (\*)** at the beginning and end of the line, with **N-2 spaces** in between.  
The Pattern should look like:  
\*<N-2 Spaces>\*  
Print the above pattern for a total of **N Rows.**  
Refer Example ouput, for better clarification.  
**Problem Constraints**

2 <= N <= 100  
**Input Format**

First and only line of input contains a single integer **N**.  
**Output Format**

Output the pattern corresponding to the given **N**.  
**Example Input**

Input 1:

2

Input 2:

4  
**Example Output**

Output 1:

\*\*

\*\*

Output 2:

\* \*

\* \*

\* \*

\* \*  
**Example Explanation**

Explanation 1:

Here N = 2, So each row should have **N - 2** spaces which is **0**.

Thus the N rows(i.e, **2** rows) should be in the form **(asterisk)(0 spaces)(asterisk)**

Explanation 2:

Here N = 4, So each row should have **N - 2** spaces which is **2**.

Thus the N rows(i.e, **4** rows) should be in the form **(asterisk)(2 spaces)(asterisk)**

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        for(int i=1;i<=num;i++){

            for(int j=1;j<=num;j++){

                System.out.print(j==1||j==num?"\*":" ");

            }

            System.out.println();

        }

    }

}