

Core java Programming notes

05 July 2025 16:55

Pattern programming:

LOOPS:

1. For loop
 2. While loop
 3. Do-while loop
 4. For-each loop
-

1. For loop:

```
//      ***** - prints in row
for(int i=1; i<=5; i++) {
    System.out.print("*");
}

//      Prints in column i.e in next line
for(int i=1; i<=5; i++) {
    System.out.println("*");
}

//      5x5 * using nested for loop
for(int i=1; i<=5; i++) {
    for(int j=1; j<=5; j++) {
        System.out.print("*");
    }
    System.out.println();
}

n*n * pattern
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
for(int i=1; i<=n; i++) {
    for(int j=1; j<=n; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

Number Pattern

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

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

Hallow space pattern:

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

```

*****
public class hollowSquarePattern {

    public static void main(String[] args) {
        int n = 8;
        for(int i =1; i<=n;i++) {
            for(int j=1; j<=n;j++) {
                if(i==1 || i==n || j==1 || j==n) {
                    System.out.print(j);
                    System.out.println("*");
                }else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}

```

1 to 25 numbers in 5*5 grid:

```

01 02 03 04 05
06 07 08 09 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25

```

```

// printing 1 to 25 in 5*5 grid
int n=5;
int count=1;
for (int i=1; i<=n; i++) {
    for(int j=1; j<=n;j++) {
        if(count < 10) {
            System.out.print("0");
        }
        System.out.print(count+" ");
        count++;
    }
    System.out.println();
}

```

1 to 25 numbers in 5*5 grid

```

int a = 5; // Grid size
for (int i = 1; i <= a; i++) { // Rows
    for (int j = 1; j <= a; j++) { // Columns
        System.out.printf("%-3d", (i - 1) * a + j); // Calculate and print
    }
    System.out.println(); // Move to next line
}

System.out.println();

```

1 to 5 table numbers in 5*5 grid :

```

01 02 03 04 05
02 04 06 08 10
03 06 09 12 15
04 08 12 16 20
05 10 15 20 25

```

```

//print 5*5 grid with 1,2,3,4,5 then 2,4,6,8,10 then 3,6,9,12,15
int m=5;
for (int i=1; i<=n; i++) {
    for(int j=1; j<=n;j++) {
        int x = i*j;
        if(x < 10) {
            System.out.print("0");
        }
        System.out.print(x+" ");
    }
    System.out.println();
}

```

Number pattern:

```
1 2 3 4 5  
2 3 4 5 6  
3 4 5 6 7  
4 5 6 7 8  
5 6 7 8 9
```

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

Right angle triangle *:

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

Right angled triangle with numbers:

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

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

```
01  
02 03  
04 05 06  
07 08 09 10  
11 12 13 14 15
```

```
int x= 5;  
int count = 1;  
for(int i=1; i<=x; i++) {  
    for(int j=1; j<=i; j++) {  
        if(count<10) {  
            System.out.print("0");  
        }  
        System.out.print((count++)+" ");  
    }  
    System.out.println();  
}
```

Mirrored right angled triangle:

```
*  
**
```

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

 int y =5;
 //in a row
 for(int i=1; i<=y; i++) {
 // Spaces:
 for(int k=0; k<=y-i ; k++) {
 System.out.print(" ");
 }

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

Pyramid triangle

```

*
*
* *
* * *
* * * *
* * * * *

/
Pyramid
int b = 5;
for(int i=1;i<=b;i++) {
//spaces
for(int k=1; k<=b-i; k++) {
System.out.print(" ");
}

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

Hallow triangle

```

*
*
* *
*   *
* * *
* * * * *

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

//OR

```

int c = 5;
for(int i=1; i<=c; i++) {

for(int j=1; j<=i; j++) {
if((i==3 & j==2) || (i==4 & j==2) || (i==4 & j==3)) {
System.out.print(" "+ " ");
} else {
System.out.print("*"+ " ");
}
}
System.out.println();
}
```

Mirrored Hallow right angled triangle:

```

*
**
* *
* *
*****
int g =5;
for(int i=1; i<=g; i++) {
    for (int k=1; k<=g-i; k++) {
        System.out.print(" ");
    }

    for(int j=1; j<=i; j++) {
        if(j==1 || i==g || i==j) {
            System.out.print("*");
        }else{
            System.out.print(" ");
        }
    }
    System.out.println();
}

```

Hallow pyramid:

```

*
* *
*   *
* * * * *
int d=5;
for(int i=1; i<=d; i++) {
    for(int k=1; k<= d-i; k++) {
        System.out.print(" ");
    }

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

// OR

//hallow pyramid:
int c = 5;
for(int i=1; i<=c; i++) {
    for(int k=1; k<=c-i; k++) {
        System.out.print(" ");
    }

    for(int j=1; j<=i; j++) {
        if((i==3 & j==2) || (i==4 & j==2) || (i==4 & j==3)) {
            System.out.print(" "+ " ");
        }else {
            System.out.print("*"+ " ");
        }
    }
    System.out.println();
}

```

2. While loop:

INTERVIEW QUES ON DATA TYPES

Note :

1. Decimal Integer : 0 1 2 3 4 5 6 7 8 9
2. Octal Integer : 0 1 2 3 4 5 6 7
3. Hexadecimal Integer : 0 1 2 3 4 5 6 7 8 9 A B C D E F
4. Binary : 0 1

Prefix and it Meanings:

1. No-prefix : Decimal
2. 0 : Octal
3. 0x : Hexa decimal
4. 0b : Binary

Note : Assignment is right to left i.e lets say b=a; Right to left - assigning a to b not b to a

45 - is not just integer its a Decimal Integer Literal

Interview ques: OCTAL TO DECIMAL

```
//what will be the output of the below  
    int a =045;  
    System.out.println(a); //37
```

--> Output is 37 because 045 is Octal not decimal as java thinks if there is 0 then its octal not decimal
--> but java displays output that is decimal because it converts octal 045 to decimal

>To convert octal to decimal -convert nos octal to binary format ignoring 0, then convert that to deci

- 045 (ignore 0) - 45
- 4 - 100 (divide by 2 till we get 1 and write the remainder)
- 5 - 101
- Attach both nos binary format - 100101 - Assign each $2^5, 2^4, 2^3, 2^2, 2^1, 2^0$ respectively
- And those has on add those - $2^5+2^2+2^0=32+4+1=37$

--> OR another way to convert octal to decimal:

```
int a = 045; // Octal literal (base 8), 045 = 4×8 + 5 = 37 in decimal  
System.out.println(a); // Prints 37 (Octal 045 converted to decimal)
```

Interview ques : HEXA DECIMAL to DECIMAL

```
int c = 0x45; // Hexadecimal literal (base 16), 0x45 = (4×16) + 5 = 69 in decimal  
System.out.println(c); // Prints 69 (Java auto-converts hex to decimal)
```

Or : 4- 0100, 5- 0101, - 01000101 - assign 2 power and add all - 69

Interview Ques : Binary to decimal - 0b -means binary

```
int d = 0b00000010; // Binary literal (base 2), 0b00000010 = 2 in decimal  
System.out.println(d); // Prints 2 (Java auto-converts binary to decimal)
```

Or : 1 is at 2^1 - 2- prints 2

Interview Ques : What will be the output?

```
int e = 078; // ✗ Error: 078 is invalid because 8 is not allowed in octal (base 8 uses digits  
0-7 only)  
System.out.println(e); // Compilation error (invalid octal literal)
```

INTERVIEW PROGR QUES IN TYPE CASTING:

```
byte a = 45;
double b;
b = a; //right to left assignment always, assigning a to b, not b to a;
System.out.println(a); //45
System.out.println(b); //45.0 -> since its double
```

- Char to int implicit conversion:

```
char d = 'A';           // 'A' has ASCII/Unicode value 65
int e;
e = d;                 // Implicit widening from char to int
System.out.println(d); // Prints A (char)
System.out.println(e); // Prints 65 (int value of 'A')
```

Char A - unicode value 65 will be stored as int in int type.

If its a - 97

```
char x = '0';           // '0' is a character, not the number zero
int y = x;              // Implicit widening: '0' → 48 (ASCII/Unicode value)
System.out.println(x); // Prints 0 (the character)
System.out.println(y); // Prints 48 (its Unicode integer value)
```

Char x has 0 as a char not int, and when we assign it to y and print y, the int unicode value will be printed, 0's unicode value is 48

Honeywell Interview Question:

```
char o = '0';
int O = o;
System.out.println(O); //48
System.out.println(o); //0

double a = 45.5;
byte b = (byte)a;
System.out.println(a); //45.5
System.out.println(b); //45
```

CODE SNIPPETS I: Incrementation and decrementation

```
//      ++ : increment
//      -- : decrement

//      ++a, --a : signs before var : pre increment
//      a ++, a-- : signs after var : post incremenet

//      1)
int x = 5;           // x is initialized to 5
int y;               // y declared
y = ++x;             // Pre-increment: x becomes 6, then assigned to y → y = 6
System.out.println(x); // 6 → because x was incremented before assignment
System.out.println(y); // 6 → because y = incremented value of x

//      2)
int j = 5;           // j is initialized to 5
int i;               // i declared
i = j++;             // Post-increment: i = 5 first, then j becomes 6
System.out.println(j); // 6 → because j was incremented after assignment
System.out.println(i); // 5 → because i got the original value of j before increment

//      3)
int c = 5;           // c is initialized to 5
int d;               // d is declared
d = ++c + ++c;       // First ++c → 6, second ++c → 7 → d = 6 + 7 = 13
System.out.println(c); // 7 → c incremented twice
System.out.println(d); // 13 → result of 6 + 7
```

```

//          4)
int e=5;
int f;
f = e++ + e++; // 5+6 = 11
System.out.println(e); //7
System.out.println(f); //11

//          5)
int g = 6;
int h;
h = ++g + g++; //7 + 7
System.out.println(g); //8
System.out.println(h); //14

//          6)
int n =5;
int m;
m= n++ + ++n + n-- + --n + ++n + --n; //5 + 7 + 7 + 5 + 6 + 5 =35
System.out.println(n); //5
System.out.println(m); //35

//          7)
int o =5;
int p;
p = o++;
System.out.println(o); //o is 6
System.out.println(p); // 5

//          8)
int q = 5;
q = q++;           // Post-increment: q++ returns 5, but it's reassigned to q, so q remains 5
System.out.println(q); // 5 → because the incremented value was not stored anywhere

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

int a = 5;
int b;
b = a++ + ++a + a++ + --a + a-- + a++ + a++;
System.out.println(a); // 8 → final value of a
System.out.println(b); // 46 → sum of all used values

// Step-by-step evaluation of:
// b = a++ + ++a + a++ + --a + a-- + a++ + a++;

// a = 5 initially
// a++ → use 5, then a = 6
// ++a → a becomes 7, use 7
// a++ → use 7, then a = 8
// --a → a becomes 7, use 7
// a-- → use 7, then a = 6
// a++ → use 6, then a = 7
// a++ → use 7, then a = 8

//b = 5 + 7 + 7 + 7 + 7 + 6 + 7; // = 46
-----
```

CODE SNIPPETS II :

Note:

1. Whenever we do arithmetic operations on integer types like byte, short, int, long Java promotes them to int by default.

So byte + byte, or short + short → result is int (even if both are byte or short).
But if one operand is long → result will be long.

2. **Truncation** means cutting off the decimal part of a number without rounding.
E.g., 12.9 → 12, -7.6 → -7

```

int g = 25;
int h = 2;
int i = g / h;
System.out.println(i); // 12 → result is int (25/2 = 12.5 but decimal part is truncated)

```

Explanation:

In integer division (int / int), Java drops the decimal part, not round.
So $25 / 2 = 12.5$ becomes 12 → only the integer part is kept.

3. In case we are doing int/int and assigning it to float then it would be implicit so there will no error and no need of explicit casting as already implicit casting is done by java internally.

```

int n = 25;
int m = 2;
float o = n / m; // int / int = 12 → assigned to float → o = 12.0
// No casting needed: Java does implicit casting from int to float (widening)
System.out.println(o); // 12.0 → decimal but no fraction as int division happened 1st

// 1)
byte b = 127; // Max value byte can hold is 127
b++; // -128 // Overflow: wraps around to -128
b++; // -127 // Increments to -127
System.out.println(b); // -127 → because of byte overflow

// 2)
byte a = 1;
byte c = 1;
byte d = (byte)(a + c); // a + c → promoted to int, so we cast back to byte
System.out.println(d); // 2 → valid output after casting otherwise error without casting

// Whenever we do mathematical operations on the integer datatypes byte, short , int , long will
always give int, as above

// 3)
int g = 25;
int h = 2;
int i = g/h;
System.out.println(i); // Since whenever we do mathematical operation on integer types returns int,
even if it is 12.5 as it should be int it will print 12
// And this is called truncation. 12 → result is int (25/2 = 12.5 but decimal part is truncated)
// Truncation means cutting off the decimal part of a number without rounding.

// 4)
int n = 25;
int m = 2;
float o = n / m; // int / int = 12 → assigned to float → o = 12.0
// No casting needed: Java does implicit casting from int to float (widening)
System.out.println(o); // 12.0 → decimal but no fraction as int division happened first

// 5)
double y = 128;
byte v = (byte)y; // Explicit cast: 128 is out of byte range (-128 to 127), so wraps to -128
System.out.println(v); // -128 → because of overflow during narrowing conversion

// byte range: -128 to 127
// since its out of range by 1 it will come to -128 if its 2 out of range it would have been -127
Casting 128 to byte causes overflow:
128 - 256 = -128 → wraps around

double f=129;
byte u = (byte)f;
System.out.println(u); // -127 // byte out of range by 2 so it goes to -127 next to -128

```

CODE SNIPPET III:

Note : Java does follow BODMAS , but since we have increment and decrement java follows **OPERATOR PRECEDENCE or PRIORITY TABLE:**

Table :

1. () parenthesis
2. ++, -- Increment / decrement
3. /, *, % --> Division Multiplication, Modulus
4. +, - --> (Addition and subtraction)
5. =, +=, -=, *=, /=, %= --> Assignment

```
int m=10, n=10;
int res = m++ / (++n * n--); // --> 10 / (11 * 11) / 10 = 10/121 * 1/10 = 1/121
System.out.println(res); //0 ==> 10 / 121 = 0 (int division, decimal truncated), ==> 0 / 10 = 0
```

Here first solve what's inside bracket -> increment and decrement -> then multiply

Then m++ and --m then division

Then first 10/121 then the result of that 0/10 - 0

```
// 1)
int m=10, n=10;
int res = m++ / (++n * n--); // --> 10 / (11 * 11) / 10 = 10/121 * 1/10 = 1/121
System.out.println(res); //0 ==> 10 / 121 = 0 (int division, decimal truncated), ==> 0 / 10 = 0

// 2)
int x=001, y=010, z=100; // x= 0x8+1=1, y=1x8+0=8, z=100
int i = --x + y++ - z-- - -z + ++y - --x + y-- - -x; // 0 + 8 - 100 - 98 + 10 + 1 + 10 + 2
System.out.println(x); // -2
System.out.println(y); // 9
System.out.println(z); // 98
System.out.println(i); // -167

// 3)
int ch = 'A'; // A unicode value 65 is assigned to int as integer , char > int : implicit
// System.out.println(ch++); //65
System.out.println(++ch); //66
System.out.println(ch); //66

// 4)
int c = 'A', C = 'a'; //c is 65 and C is 97
System.out.println(c++ + ++C); // 65 + 98 = 163

// 5)
byte j = 5;
// j = j*10; // will give error since int cant be put into byte - needs explicit casting
j = (byte)(j*10);
System.out.println(j); //50

// 6)
double a = 150.0;
byte b = (byte)a; //needs explicit casting to work
// 150 converted into byte is out of range by 23, so from 128 we should subtract 23-1 =22 i.e 128-22 = 106, answer is -106
System.out.println(a); //150.0
System.out.println(b); // -106
```

```
package com.fizzbuzz;
```

```
import java.util.Arrays;
import java.util.Scanner;
```

```
/*
```

QUESTION:

Write a program that takes an integer A as input and generates a list of strings representing numbers from 1 to A, replacing:

- Multiples of 3 with "Fizz"
- Multiples of 5 with "Buzz"
- Multiples of both 3 and 5 with "FizzBuzz"

The program should then print the sequence as an array.

Example:

Input: 5

Output: [1, 2, Fizz, 4, Buzz]

Input: 15

Output: [1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz]

*/

```
public class Solution {

    public String[] fizzbuzz(int A) {

        String[] res = new String[A];
        for(int i=0; i< A; i++) {
            int num = i+1;
            if(num%3 ==0 && num%5==0) {
                res[i] = "FizzBuzz";
            }else if(num%3==0) {
                res[i] = "Fizz";
            }else if(num%5==0){
                res[i] = "Buzz";
            }else {
                res[i] = String.valueOf(num);
            }
        }

        return res;
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number: ");
        int A = sc.nextInt();

        Solution sol = new Solution();
        String[] op = sol.fizzbuzz(A);

        System.out.println(Arrays.toString(op)); // Converts the String[] array into a readable string format like [1, 2, Fizz, 4, Buzz]
    }
}
```

```
package programmingLogic;

import java.util.Scanner;

/*
    1. Even-Odd Replacement
    Write a program that takes an integer N and prints numbers from 1 to N, but replaces:
        Multiples of 2 with "Even"
        Multiples of 3 with "Odd"
        Multiples of both 2 and 3 with "EvenOdd"*/
class EvenOdd{
```

```

void solution(int N) {
    for(int i=1; i<=N; i++) {
        if(i%2==0 && i%3==0) {
            System.out.println("EvenOdd");
            continue;
        }else if(i%2==0) {
            System.out.println("Even");
            continue;
        }else if(i%3==0) {
            System.out.println("Odd");
            continue;
        }
        System.out.println(i);
    }
}

public class EvenOddReplacement {
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter N: ");
        int N = sc.nextInt();

        EvenOdd e = new EvenOdd();
        e.solution(N);
    }
}

```

```

package programmingLogic;

import java.util.Arrays;
import java.util.Scanner;

/*
2. Prime Checker for Range
Write a program that takes an integer N and prints numbers from 1 to N,
but replaces:
- Prime numbers with "Prime"
- Non-prime numbers with "Not Prime"
*/
class PrimeCheck {

    public String[] prime(int N) {
        String[] res = new String[N];

        for(int i = 1; i<=N; i++) {
            if(isPrime(i)) {
                res[i-1] = "Prime"; //shifts index by -1, since we start from 1 to N
            }else {
                res[i-1] = "Not prime";
            }
        }

        return res;
    }

    boolean isPrime (int num) {
        if (num < 2) return false;

        for(int i=2; i<=Math.sqrt(num); i++) {
            if(num%i==0) return false; //not prime since it has divisor
        }

        return true; //prime
    }
}

public class PrimeChecker {
    public static void main(String[] args) {

```

```

Scanner sc = new Scanner(System.in);
System.out.println("Enter N: ");
int N = sc.nextInt();

PrimeCheck p = new PrimeCheck();
String[] op = p.prime(N);

System.out.println(Arrays.toString(op));

}

```

package programmingLogic;

import java.util.Scanner;
/*

3. Reverse Words in a Sentence

Take a string input and reverse each word individually without changing the order of words.

Example: "Hello World" → "olleH dlrow"*/

```

class ReverseSentence{

    public String ReversEachWord(String s) {
        String[] word = s.split(" ");

        StringBuilder res = new StringBuilder();

        for(String w : word) {
            String reversed = new StringBuilder(w).reverse().toString();
            res.append(reversed).append(" ");
        }

        return res.toString().trim();
    }
}

```

public class ReverseWordInSentence {

```

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a sentence: ");
        String s = sc.nextLine();

        ReverseSentence reverse = new ReverseSentence();
        String op = reverse.ReversEachWord(s);

        System.out.println(op);
    }
}

```

package programmingLogic;

import java.util.Scanner;

/*4. Armstrong Number Finder

Find all Armstrong numbers between 1 and a given number N.

(An Armstrong number is one whose sum of its own digits raised to the power of the number of digits equals the number itself.)*/

//72 - 7^2 + 9^2 = 49 + 81 -> not equal to 72 - not Armstrong number

```
//153 -> 1^3 + 5^3 + 3^3 = 1+125+27 = 153 - Armstrong number  
//Other egs - 370 371 407 1634 8208 9474...
```

```
class FindArmStrong{  
  
    //to check if arm - num is each number not N  
    boolean find(int num) {  
  
        // String s = new String(String.valueOf(num)); //takes time and space so create count method as below  
        int count = s.length();  
        int count = count(num);  
        int originalNum = num;  
  
        int sum = 0;  
  
        while(num>0) {  
            int eachDigit = num%10; //%10 gets last digit  
            sum+=Math.pow(eachDigit, count);  
            num /=10; // /10- removes last digit  
        }  
  
        if(sum == originalNum) {  
            return true;  
        }  
        return false;  
    }  
  
    int count(int num) {  
        int count = 0;  
        int temp = num;  
        while(temp>0) {  
            temp/=10;  
            count++;  
        }  
  
        return count;  
    }  
}  
  
public class AmstrongNumberFinder {  
  
    public static void main(String[] args) {  
  
        Scanner sc= new Scanner(System.in);  
        System.out.println("Enter N to find armstrin numbers between 1 to N: ");  
        int N = sc.nextInt();  
  
        FindArmStrong arm = new FindArmStrong();  
  
        for(int i=1; i<=N; i++) {  
            if(arm.find(i)) {  
                System.out.print(i+ " "); //1, 2, 3, 4, 5, 6, 7, 8, 9 are Armstrong numbers by definition.-  
since number of digits = 1 -> 5^1 = 5  
            }  
        }  
    }  
}
```
