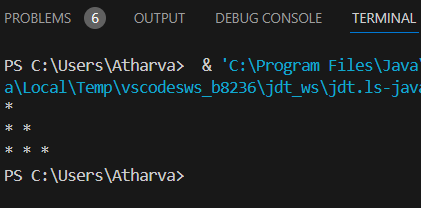
**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Roll no:- \_\_\_\_\_\_\_\_\_\_\_**

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.no** | **Experiment Name** | **Date** | **Sign** |
| **1** | Right Half Pyramid |  |  |
| **2** | Left Half Pyramid |  |  |
| **3** | Inverted Right Half Pyramid |  |  |
| **4** | Inverted Left Half Pyramid |  |  |
| **5** | Full Pyramid |  |  |
| **6** | Inverted Full Pyramid |  |  |
| **7** | Hollow Full Pyramid |  |  |
| **8** | Inverted Hollow Full Pyramid |  |  |
| **9** | Hour Glass Pyramid |  |  |
| **10** | Rhombus Pattern |  |  |
| **11** | Square Pattern |  |  |
| **12** | Rectangle Pattern |  |  |
| **13** | Hollow Square Pattern |  |  |
| **14** | Diamond Pattern |  |  |
| **15** | Hollow Diamond Pattern |  |  |
| **16** | Hollow Hour Glass Pattern |  |  |
| **17** | Floyd’s Triangle Pattern |  |  |
| **18** | Pascal’s Triangle Pattern |  |  |
| **19** | Alphabetic Triangle Pattern |  |  |
| **20** | Numeric Triangle Pattern |  |  |

**Output**

****

**Right Half Pyramid**

* **Code:-**

**// Right Half Pyramid**

class RightHalfPyramidEx {

public static void main(String[] args) {

int n = 3;

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

for (int j = 1; j <= n; j++) {

if (j <= i) {

System.out.print("\* ");

} else {

System.out.print(" ");

}

}

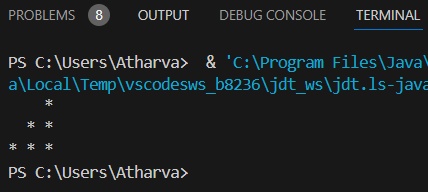
System.out.println();

}

}

}

**Output**

****

**Left Half Pyramid**

* **Code:-**

**// Left Half Pyramid**

class LeftHalfPyramidEx {

public static void main(String[] args) {

int n = 3;

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

for (int j = 1; j <= n; j++) {

if (j <= n - i) {

System.out.print(" ");

} else {

System.out.print("\* ");

}

}

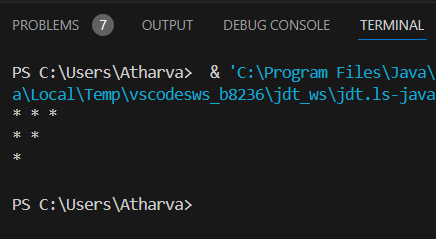
System.out.println();

}

}

}

**Output**

****

**Inverted Right Half Pyramid**

* **Code:-**

**//Inverted Right half Pyramid**

class InvertedRighthalfPyramidEx

{

public static void main(String args[])

{

int n=4;

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

for (int j = 1; j <= n; j++) {

if (j <=n-i ) {

System.out.print("\* ");

} else {

System.out.print(" ");

}

}

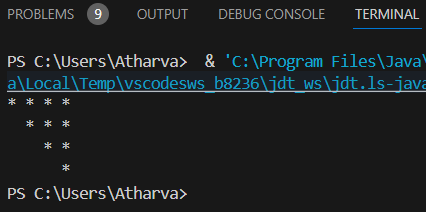
System.out.println();

}

}

}

**Output**

****

**Inverted Left Half Pyramid**

* **Code:-**

**// Inverted Left Half Pyramid**

class InvertedLeftHalfPyramidEx

{

public static void main(String[] args)

{

int n = 4;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (j >= i) {

System.out.print("\* ");

} else {

System.out.print(" ");

}

}

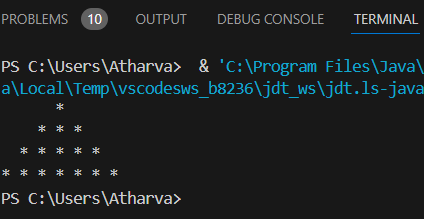
System.out.println();

}

}

}

**Output**

****

**Full Pyramid**

* **Code:-**

**// Full Pyramid using for loop and if-else**

class FullPyramidEx {

public static void main(String[] args) {

int n = 4;

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

for (int j = 1; j <= (2 \* n - 1); j++) {

if (j >= n - i + 1 && j <= n + i - 1) {

System.out.print("\* ");

} else {

System.out.print(" ");

}

}

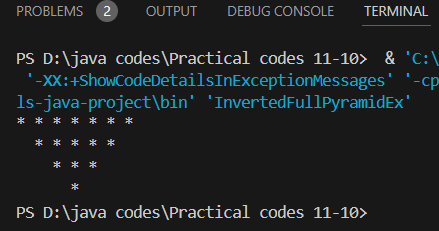
System.out.println();

}

}

}

**Output**

****

**Inverted Full Pyramid**

* **Code:-**

**//Inverted Full Pyramid using for loop and if-else**

class InvertedFullPyramidEx

{

public static void main(String[] args)

{

int n = 4;

for (int i = n; i >= 1; i--) {

for (int j = 1; j <= (2 \* n - 1); j++)

{

if (j >= n - i + 1 && j <= n + i - 1)

{

System.out.print("\* ");

}

else

{

System.out.print(" ");

}

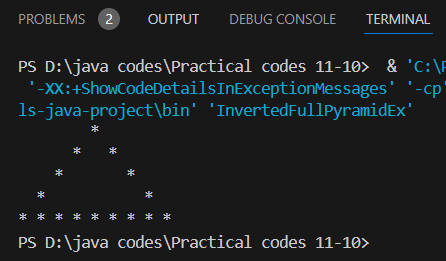
}

System.out.println();

}

}}

**Output**



**Hollow Full Pyramid**

* **Code:-**

**//Hollow Full Pyramid using for loop and if-else**

class HollowFullPyramidEx

{

public static void main(String[] args)

{

int n = 5;

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

for (int j = 1; j <= (2 \* n - 1); j++) {

// Print '\*' at the borders or at the base row

if (j == n - i + 1 || j == n + i - 1 || i == n) {

System.out.print("\* ");

} else {

System.out.print(" ");

}

}

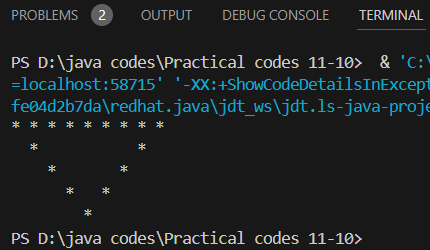
System.out.println();

}

}

}

**Output**

****

**Inverted Hollow Full Pyramid**

* **Code:-**

**//Inverted Hollow Full Pyramid using for loop and if-else**

class InvertedHollowFullPyramidEx {

public static void main(String[] args) {

int n = 5;

for (int i = n; i >= 1; i--) {

// Print leading spaces for alignment

for (int j = 1; j <= n - i; j++) {

System.out.print(" ");

}

for (int j = 1; j <= 2 \* i - 1; j++) {

// Print '\*' at the start, end of the row, or the top row

if (j == 1 || j == 2 \* i - 1 || i == n) {

System.out.print("\* ");

} else {

System.out.print(" ");

}

}

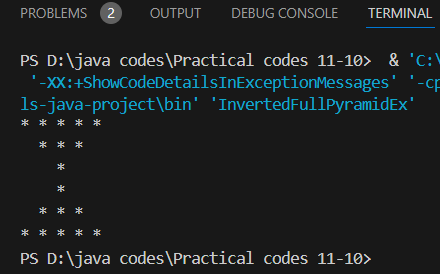
System.out.println();

}

}

}

**Output**

****

**Hour Glass Pyramid**

* **Code:-**

**//Hour Glass Pyramid using for loop**

class HourGlassPyramidEx {

public static void main(String[] args) {

int n = 3;

for (int i = n; i >= 1; i--) {

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(); }

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

for (int j = 1; j <= n - i; j++) {

System.out.print(" ");

}

for (int j = 1; j <= 2 \* i - 1; j++) {

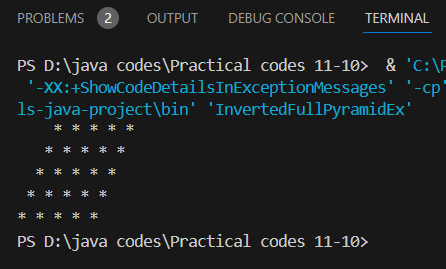
System.out.print("\* ");

}

System.out.println();

}}}

**Output**

****

**Rhombus Pattern**

* **Code:-**

**//Roumbus Pattern using for loop**

public class RhombusPattern

{

public static void main(String[] args)

{

int n = 5;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

System.out.print(" ");

}

for (int j = 0; j < n; j++)

{

System.out.print("\* ");

}

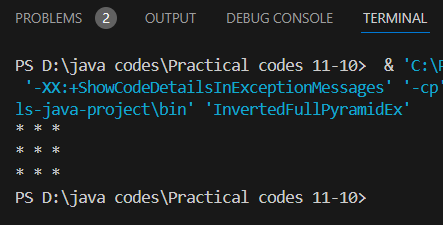
System.out.println();

}

}

}

**Output**



**Square Pattern**

* **Code:-**

**//Square Pattern using for loop**

public class SquarePattern

{

public static void main(String[] args)

{

int n = 3;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

System.out.print("\* ");

}

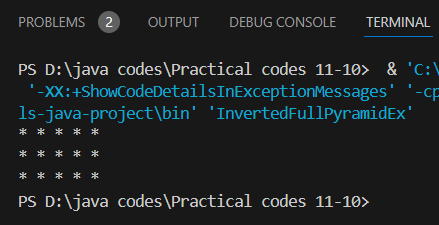
System.out.println();

}

}

}

**Output**



**Rectangle Pattern**

* **Code:-**

**//Rectangle Pattern using for loop**

public class RectanglePattern

{

public static void main(String[] args)

{

int rows = 3;

int columns = 5;

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < columns; j++)

{

System.out.print("\* ");

}

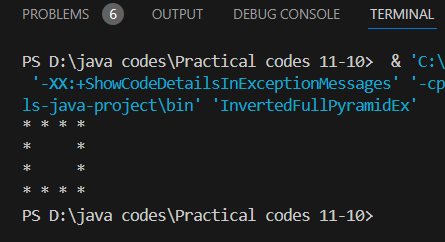
System.out.println();

}

}

}

**Output**



**Hollow Square Pattern**

* **Code:-**

**//Hollow Square Pattern using for loop**

public class HollowSquarePattern

{

public static void main(String[] args) {

int n = 4;

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("\* ");

}

else

{

System.out.print(" ");

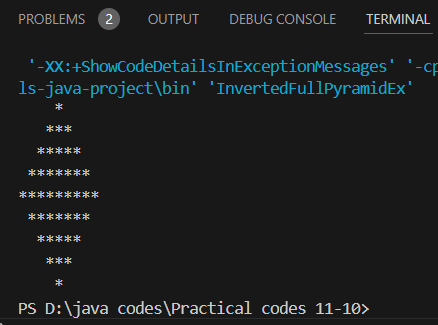
}

}

System.out.println();

}}}

**Output**

****

**Diamond Pattern**

* **Code:-**

**//Diamond Pattern using for loop**

public class SimpleDiamondPattern

{

public static void main(String[] args)

{

int n = 5;

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

for (int j = n - i; j > 0; j--) {

System.out.print(" ");

}

for (int k = 1; k <= (2 \* i - 1); k++) {

System.out.print("\*");

}

System.out.println(); }

for (int i = n - 1; i >= 1; i--) {

for (int j = n - i; j > 0; j--) {

System.out.print(" ");

}

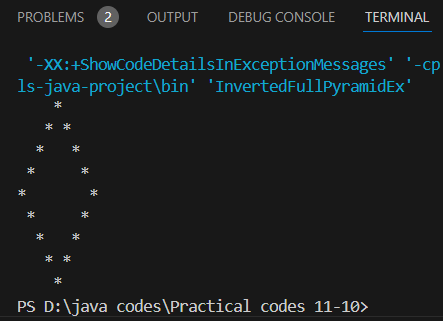
for (int k = 1; k <= (2 \* i - 1); k++) {

System.out.print("\*"); }

System.out.println();

}}}

**Output**

****

**Hollow Diamond Pattern**

* **Code:-**

**//Hollow Diamond Pattern using for loop**

public class HollowDiamondPattern {

public static void main(String[] args) {

int n = 5;

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

// Print leading spaces

for (int j = n - i; j > 0; j--) {

System.out.print(" "); }

for (int k = 1; k <= (2 \* i - 1); k++) {

if (k == 1 || k == (2 \* i - 1)) {

System.out.print("\*");

} else {

System.out.print(" ");

}}

System.out.println(); }

for (int i = n - 1; i >= 1; i--) {

for (int j = n - i; j > 0; j--) {

System.out.print(" "); }

for (int k = 1; k <= (2 \* i - 1); k++) {

if (k == 1 || k == (2 \* i - 1)) {

System.out.print("\*");

} else {

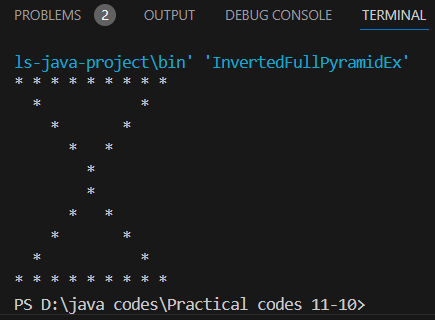
System.out.print(" ");

}}

System.out.println();

}}}

**Output**



**Hollow Hour Glass Pattern**

* **Code:-**

**//Hellow Hour Glass Pattern using for loop**

public class HollowHourglassPattern {

public static void main(String[] args) {

int n = 5;

for (int i = n; i >= 1; i--) {

for (int j = 1; j <= n - i; j++) {

System.out.print(" ");

}

for (int j = 1; j <= 2 \* i - 1; j++) {

if (j == 1 || j == 2 \* i - 1 || i == n) {

System.out.print("\* ");

} else {

System.out.print(" ");

}}

System.out.println();

}

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

for (int j = 1; j <= n - i; j++) {

System.out.print(" ");

}

for (int j = 1; j <= 2 \* i - 1; j++) {

if (j == 1 || j == 2 \* i - 1 || i == n) {

System.out.print("\* ");

} else {

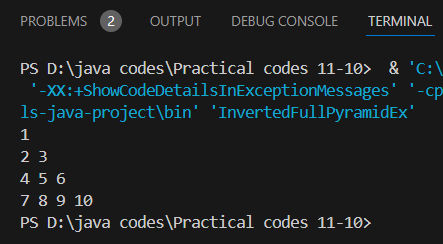
System.out.print(" ");

}}

System.out.println();

}}}

**Output**



**Floyd’s Triangle Pattern**

* **Code:-**

**//Floyd’s Triangle Pattern using for loop**

public class FloydsTriangle

{

public static void main(String[] args)

{

int n = 4;

int num = 1;

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= i; j++)

{

System.out.print(num + " ");

num++;

}

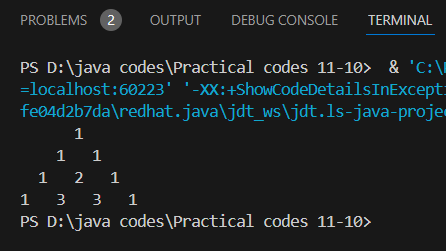
System.out.println();

}

}

}

**Output**



**Pascal’s Triangle Pattern**

* **Code:-**

**//Pascal’s Triangle Pattern using for loop**

class PascalsTriangle

{

public static void main(String[] args) {

int n = 4;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n - i - 1; j++)

{

System.out.print(" ");

}

int value = 1;

for (int j = 0; j <= i; j++)

{

System.out.print(value + " ");

value = value \* (i - j) / (j + 1);

}

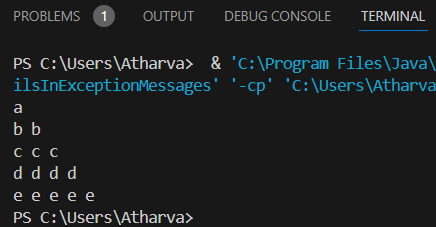
System.out.println();

}

}

}

**Output**

****

**Alphabetic Triangle Pattern**

* **Code:-**

**//Alphabetic Triangle Pattern using array and for loop**

public class alphabeticpattern

{

public static void main(String[] args)

{

char[] arr = {'a', 'b', 'c', 'd','e'};

for(int i = 0; i < 5; i++) {

for(int j = 0; j <= i; j++)

{

System.out.print(arr[i]+" ");

}

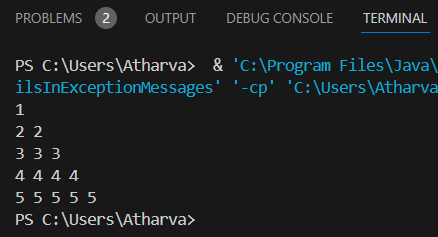
System.out.print("\n");

}

}

}

**Output**



**Numeric Triangle Pattern**

* **Code:-**

**//Numeric Triangle Pattern using array and for loop**

**public class Numberpattern {**

**public static void main(String[] args) {**

**char[] arr = {'1', '2', '3', '4','5'};**

**for(int i = 0; i < 5; i++) {**

**for(int j = 0; j <= i; j++)**

**{**

**System.out.print(arr[i]+" ");**

**}**

**System.out.print("\n");**

**}**

**}**

**}**