HASH SET

HashSet in Java implements the Set interface of Collections Framework. It is used to store the unique elements and it doesn’t maintain any specific order of elements.

- Can store the Null values.
- Uses HashMap (implementation of hash table data structure) internally.
- Also implements Serializable and Cloneable interfaces.

Methods in Java HashSet

Method	**Description**
add(E e)	Used to add the specified element if it is not present, if it is present then return false.
clear()	Used to remove all the elements from the set.
contains(Object o)	Used to return true if an element is present in a set.
remove(Object o)	Used to remove the element if it is present in set.
iterator()	Used to return an iterator over the element in the set.
isEmpty()	Used to check whether the set is empty or not. Returns true for empty and false for a non-empty condition for set.
size()	Used to return the size of the set.

Method	Description
clone()	Used to create a shallow copy of the set.

Performing Various Operations on HashSet

Let’s see how to perform a few frequently used operations on the HashSet.

1. Adding Elements in HashSet

To add an element to the HashSet, we can use the add() method. However, the insertion order is not retained in the HashSet. We need to keep a note that duplicate elements are not allowed and all duplicate elements are ignored.

Example: Java program to Adding Elements to HashSet

```
import java.util.*;

class GFG

{

    public static void main(String[] args)

    {

        // Creating an empty HashSet of string entities

        HashSet<String> hs = new HashSet<String>();


        // Adding elements using add() method

        hs.add("harman");

        hs.add("samsung");

        hs.add("company");


        // Printing all string entries inside the Set

        System.out.println("HashSet : " + hs);
    }
}
```

Output

```
HashSet : [harman,samsung, company]
```

2. Removing Elements in HashSet

The values can be removed from the HashSet using the remove() method.

Example:

Removing Elements in HashSet

```
import java.util.*;
class GFG
{
    public static void main(String[] args)
    {

        HashSet<String> hs = new HashSet<String>();

        // Adding elements to above Set
        // using add() method
        hs.add("harman");
        hs.add("samsung");
        hs.add("company");
        hs.add("A");
        hs.add("B");
        hs.add("Z");
        // Printing the elements of HashSet elements
        System.out.println("HashSet : " + hs);

        // Removing the element B
        hs.remove("B");

        // Printing the updated HashSet elements
        System.out.println("HashSet after removing element : " + hs);

        // Returns false if the element is not present
        System.out.println("B exists in Set : " + hs.remove("B"));
    }
}
```

Output

```
HashSet : [A, B, harman, samsung, company, Z]
HashSet after removing element [A, harman, samsung, company, Z]
B exists in Set : false
```

3. Iterating through the HashSet

Iterate through the elements of HashSet using the iterator() method. Also, the most famous one is to use the enhanced for loop.

Example: Iterating through the HashSet

```
import java.util.HashSet;
import java.util.Iterator;
```

```
public class GFG

{

    public static void main(String[] args)

    {

        // Create a HashSet of Strings

        HashSet<String> hs = new HashSet<>();


        // Add elements to the HashSet

        hs.add("A");

        hs.add("B");

        hs.add("G");

        hs.add("F");

        hs.add("G");

        hs.add("Z");


        // Using iterator() method to iterate

          // Over the HashSet

        System.out.print("Using iterator : ");

        Iterator<String> iterator = hs.iterator();


          // Traversing HashSet

        while (iterator.hasNext())

            System.out.print(iterator.next() + ", ");
        System.out.println();

        // Using enhanced for loop to iterate
          // Over the HashSet
        System.out.print("Using enhanced for loop : ");
        for (String element : hs)
            System.out.print(element + " , ");

    }
}
```

Output

```
Using iterator : A, B, G, F, Z,
Using enhanced for loop : A , B , G , F , Z ,
```

Pattern matching problem:

Input :

for n=6

Output:

```
                         1
                    2    4
               3    5    7
          6    8    10   12
     9    11   13   15   17
14   16   18   20   22   24
```

Code:

```
import java.util.Scanner;


public class Main {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        System.out.println("Enter n:");

        int n = in.nextInt();

        int num = 1

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

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

                System.out.print("   ");

            }

            int current = num;
            for (int j = 0; j <= i; j++) {
                System.out.printf("%3d", current);
                current += 2; // Increment by 2 within the row
            }
            if(i%2==0)
            num += i +1;
            else
            num+=i;// Move to the next unused smallest number for the next row
            System.out.println();
        }
    }
}
```