HashSet Class

HashSet class is used when we want to store unique values or unique object. They are not like arrays or ArrayList where each element can be stored multiple times. HashSet maintains the uniqueness. If we try to add a duplicate element it simple rejects them. In ArrayList or arrays we can access an element using its index. However, there is no such method like get() in HashSet.

We use HashSet where we are only concerned about the presence of a value or object in given HashSet. And we want to maintain the uniqueness. We don't want to access elements using any index.

Use HashSet in following situations:

- Maintaining the uniqueness of data is more important and the order in which they are stored isn't required.
- Just want to confirm if certain element is present or not in a very quick and efficient manner (skipping iteration, accessing and comparison loop while doing it).

Syntax:

```
HashSet<ClassName> set_name = new HashSet<ClassName>();
```

Like ArrayList, HashSet can only hold objects. So if you want to store any primitive data type then you have to use their corresponding wrapper class.

Class Name for different data types:

- **Integer** class for int
- **String** class for String
- **Character** class for char

Example 1: Creating Simple HashSet

```
1  HashSet<Integer> set1 = new HashSet<Integer>();
2  HashSet<String> set2 = new HashSet<String>();
```

Basic Methods: add() and size()

Some basic methods:

- add(element) : adds the given "element" in the HashSet if it's not already present. This ensures no duplicates.
- size() : returns the size of HashSet (total elements present)

Example 2.1: Using simple methods

```
1  HashSet<String> studentSet = new HashSet<String>();
2
3  studentSet.add("Rohit");                  //adding element to our HashSet
4  studentSet.add("Manish");
5
6  int len = studentSet.size();              //getting current size of our HashSet
```

```
7  System.out.println(len);
8
9  studentSet.add("Manish");                  //notice we are adding Manish again
10 len = studentSet.size();
11 System.out.println(len);
12
13 System.out.println(studentSet);            //printing our HashSet
```

Output:

```
2
2
[Rohit, Manish]
```

Explanation:

In the above example we can see that String "Manish" has been added 2 times. But, the HashSet size remains same because only one occurrence of "Manish" is present there.

Example 2.2: Printing unique elements in an array

```
1  int[] arr = {2, 1, 2, 4, 3, 3, 4, 5};                  //notice duplicates
2  HashSet<Integer> mySet = new HashSet<Integer>();
3
4  for(int i = 0; i < arr.length; i++)
5  {                                          //we are directly adding all the array
6     int n = arr[i];                         //elements. HashSet automatically
7     mySet.add(n);                           //ensures uniqueness in elements.
8  }
9  int len = mySet.size();                    //total unique elements
10 System.out.println(len);
11
12 System.out.println(mySet);                 //printing our HashSet
```

Output:

```
5
[2, 1, 4, 3, 5]
```

Explanation:

Above we have shown one of the common use of HashSet to get unique elements from an array. We just keep on adding every element from the array to our HashSet. HashSet keeps the elements which are added first time. Their duplicates are simply discarded.

Example 2.3: Printing unique elements from ArrayList

```
1  //Assume ArrayList<String> visitors:["neha", "prakash", "neha", "vivek",
2  "prakash", "prakash"];
3
4  HashSet<String> uniqueVisitors = new HashSet<String>();
5
6  int len = arrList.size();
7
```

```
8  for(int i = 0; i < len; i++)
9  {
10    String visitor = visitors.get(i);            //getting element from ArrayList
11    uniqueVisitors.add(visitor);
12 }
13
14 int len = uniqueVisitors.size();               //total unique elements
15 System.out.println(len);
16
17 System.out.println(uniqueVisitors);            //printing our HashSet
```

Output:

```
3
[neha, Prakash, vivek]
```

Problem 1	**Give output of following code.**

```
1  HashSet<String> gatePassSwipe = new HashSet<String>();
2
3  gatePassSwipe.add("Surbhi");
4  gatePassSwipe.add("Rohan");
5  gatePassSwipe.add("Sakshi");
6  gatePassSwipe.add("Rohit");
7  gatePassSwipe.add("Sakshi Sharma");
8  gatePassSwipe.add("Rohan");
9  gatePassSwipe.add("Tushar");
10
11 len = gatePassSwipe.size();
12 System.out.println("Unique: " + len);
13
14 System.out.println(gatePassSwipe);
```

Problem 2	**Give output of following code.**

```
1  char[] arr = {'T', 'R', 'I', 'S', 'E', 'C', 'T'};
2  HashSet<Character> set1 = new HashSet<Character>();
3
4  for(int i = 0; i < arr.length; i++)
5  {
6     char ch = arr[i];
7     set1.add(ch);
8  }
9  int len = set1.size();
10 System.out.println(len);
11
12 System.out.println(set1);
```

Searching an element

Searching an element in given HashSet is very simple and very fast.

- contains(element) : returns true if "element" is present in given HashSet.

Example 3.1: Checking if the given array elements are present in the given HashSet

```
1  //Assume HashSet : mySet<Integer> contains [3, 1, 7, 4]
2
3  int[] arr = {1, 2, 3, 4, 5};
4
5  for(int i = 0; i < arr.length; i++)
6  {
7     int n = arr[i];
8     if(mySet.contains(n))
9     {
10          System.out.println("Present : " + n);
11    }
12    else
13    {
14          System.out.println("Absent : " + n);
15    }
16 }
```

Output:

```
Present : 1
Absent : 2
Present : 3
Present : 4
Absent : 5
```

Explanation:

The contains() method of HashSet is very efficient. Due to internal working of HashSet searching an elements takes constant time irrespective of input size. This is very efficient in comparison to standard method of looping through every element and then comparing. We sometimes employ HashSet when we need very fast lookups.

Example 3.2: Printing all duplicate elements in given integer array

```
1  int[] arr = {20, 10, 10, 20, 30, 40, 50, 30};                  //notice duplicates
2  HashSet<Integer> mySet = new HashSet<Integer>();
3
4  for(int i = 0; i < arr.length; i++)
5  {
6     int n = arr[i];
7
8     System.out.println("Iteration : " + i);
9
10    if(mySet.contains(n))
11    {
12          //Printing element if HashSet already contains it
13          System.out.println("Duplicate : " + n);
14    }
```

```
15
16      mySet.add(n);
17      System.out.println(mySet);
18  }
```

Output:

```
Iteration : 0
[20]
Iteration : 1
[20, 10]
Iteration : 2
Duplicate : 10
[20, 10]
Iteration : 3
Duplicate : 20
[20, 10]
Iteration : 4
[20, 10, 30]
Iteration : 5
[20, 10, 30, 40]
Iteration : 6
[20, 10, 30, 40, 50]
Iteration : 7
Duplicate : 30
[20, 10, 30, 40, 50]
```

Explanation:

The above program prints only duplicates. The contains() method helps us here. We are starting with empty HashSet and one by one adding elements in the set. However, at each iteration we are checking if the set already contains the current element. If yes then it means we have already encountered it before i.e. it is duplicate.

Problem 3	**Give output of following code.**

```
1   //Assume HashSet : mySet<Character> contains ['c', 'o', 'l', 'a'];
2
3   String str = "chocolate";
4   String newStr = "";
5
6   for(int i = 0; i < str.length(); i++)
7   {
8       char ch = str.charAt(i);
9       if(mySet.contains(ch))
10      {
11              newStr = newStr + ch;
12      }
13      else
14      {
15              newStr = newStr + "#";
16      }
17  }
18
```

```
19 System.out.println(newStr);
```

Problem 4 **Give output of following code.**

```
1  //Assume ArrayList<Integer> anonymousUsers:      [1, 2, 3, 5, 2, 3, 7, 1];
2
3  HashSet<Integer> uniqueUsers = new HashSet<Integer>();
4
5  int len = anonymousUsers.size();
6
7  for(int i = 0; i < len; i++)
8  {
9     int userId = anonymousUsers.get(i);
10    uniqueUsers.add(userId);
11 }
12
13 int total = uniqueUsers.size();
14 System.out.println(total);
15
16 System.out.println(uniqueUsers);
```

Problem 5 **Give output of following code.**

```
1  String str = "abracadabra"
2
3  HashSet<Character> mySet = new HashSet<Character>();
4
5  for(int i = 0; i < str.length(); i++)
6  {
7     char ch = str.charAt(i);
8
9     if(mySet.contains(ch))
10    {
11          System.out.println("$ : " + ch);
12    }
13
14    mySet.add(ch);
15 }
16 System.out.println(mySet);
```

Key Points

HashSet is part of Java Collection Framework. Some of the key points about HashSet are:

- HashSet class implements Set interface. Set interface provides basic methods like add(), contains(), etc.
- HashSet do not allow any duplicates.
- HashSet do not guarantee iteration order with time. This means traversing a given HashSet may or may not give us the elements in the same order in which they were inserted.
- HashSet do not allow getting an element based on its index.
- Searching an element is very fast in HashSet.

Problem 6 **UniqueOdds**	**Complete the function given below. The function takes an integer array as input. Print all the unique odd numbers in this array. Use a HashSet to solve this problem.**	

`public void uniqueOdds(int[] arr)`
`{`
`  //print all the unique odd numbers in the input array`
`  //use HashSet to solve this problem`
`}`		

Test Cases	**Input**	**Output**
	`uniqueOdds({7,8,2,5,7,3,9,3,4,2})`	`[7,5,3,9]` //only 7,5,3 and 9 are odd

Problem 7 **ReplaceDup**	**Complete the function given below. The function takes a String ArrayList as input. It then replaces all the repeat occurrences of any String in this ArrayList with "Hello" and print the modified version.**	
Steps	**1) Traverse the ArrayList and keep adding elements in a HashSet** **2) If the element is already present in HashSet then replace that element in ArrayList with "Hello"** **3) Print the modified ArrayList**	

`public void replaceDup(ArrayList<String> arrList)`
`{`
`  //replace all the repeat occurrences of`
`  //any String in input ArrayList with "hello"`
`  //Print the modified ArrayList`
`}`		

Test Cases	**Input**	**Output**
	`replaceDup([mat,bad,cat,mad,cat,bat,mat])`	`[mat,bad,cat,hello,` `hello,bat,hello]`

Problem 8 **UniqueFromTwo**	**Complete the function given below. The function takes two integer arrays as input. It then prints all the unique elements in both the arrays.**
Steps	**1) Add all elements from first array to a HashSet** **2) Add all elements from second array to a HashSet** **3) Print the HashSet**

```
public void uniqueFromTwo(int[] arr1, int[] arr2)
{
	//print all the unique elements from both the arrays
}
```

Test Cases	**Input**	**Output**
	uniqueFromTwo({2,1,3,2,5},{3,4,0,1,4})	[2,1,3,5,4,0]

Problem 9 **CommonFromTwo**	**Complete the function given below. The function takes two integer arrays as input. It then prints all the elements from second array that are present in the first (common elements). Assume there are no duplicates in each array.**
Steps	**1) Add all elements from first array to a HashSet** **2) Traverse the second array. And print all the elements which are present in the Set.**

```
public void commonFromTwo(int[] arr1, int[] arr2)
{
	//print all the common elements in input arrays
}
```

Test Cases	**Input**	**Output**
	commonFromTwo({2,1,5,3},{3,4,0,1,10})	3, 1