**Sets**

*Sets* in Java correspond to sets in math, which hold objects without duplicates. One would expect to see useful methods like add, remove, contains, and size. Look at the API cheat sheet. One would expect to be able to iterate over a set. Write the code to print the elements in mySet, which holds Strings. Java provides two ways.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

HashSet<E> and TreeSet<E> are two concrete classes that implement the Set<E> interface.

A HashSet stores objects according to its hash function, which means the actual storage location is impossible to predict in advance. A TreeSet stores objects in BST order. Therefore, if the order of the objects is not important, use a HashSet. If the order of the objects is important, use a TreeSet.

A HashSet provides O(1) run times for the add and remove operations. A TreeSet provides O(log n) run times for the add and remove operations. That's because TreeSets are implemented as self-balancing binary search trees. Nice.

To summarize, a HashSet is faster but unordered. A TreeSet is slower but ordered.

**Example code for Sets**

6 Set<String> s = new HashSet<String>();  
 7 s.add("Mary");  
 8 s.add("Joan");  
 9 s.add("Mary"); //duplicate!  
10 s.add("Dennis");  
11 s.add("Bob");  
12 s.add("MaryAnn");  
13 s.add("Zoe");  
14 System.out.println("Size: " + s.size());  
15 Iterator <String> it = s.iterator(); //iterator  
16 while(it.hasNext())  
17 System.out.print(it.next() + " ");  
18 System.out.println();  
19   
20 Set<String> t = new TreeSet<String>(s);//from HashSet to TreeSet  
21 for( String str : t ) //for-each  
22 System.out.print( str + " " );  
23 System.out.println();   
24 System.out.println(s); //print any Collection--wow!  
25 System.out.println(t);

Output: line 14: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lines 15-17: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lines 21-22: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

line 24: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

line 25: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_