Consider the following method:

```
public void mysteryMix(Integer a, Integer b)
{
    a = new Integer(2 * a.intValue());
    b = a;
}
```

What is the output of the following code?

```
Integer a = new Integer(1);
Integer b = new Integer(2);
mysteryMix(a, a);
mysteryMix(a, b);
System.out.println(a + " " + b);
```

(A) 1 1
(B) 1 2
(C) 2 2
(D) 1 4
(E) 4 4

(B) is correct. Integer objects, like String objects are immutable.

STUDENT OUTLINE

Lesson 39 – Sets and Maps

INTRODUCTION: In this guide, you've learned several different data structures. We'll now step back a bit and look at some ways to collect data independent of the implementations that we've learned. Sets and Maps are higher in the inheritance hierarchy than trees and hash tables..

The key topics for this lesson are:

A. Sets
B. Maps
C. Tree vs. Hash Implementations

DISCUSSION:

A. Sets

1. In Math classes you may have studied some set theory. The idea of a Set in Java is very similar to the mathematical concept. A Set is an unordered collection of distinct elements. Elements can be added, located, and removed.

2. Set is a Java *Interface*, so it dictates what actions a Set must have, but the implementation is left open.

JAVA API's

java.util

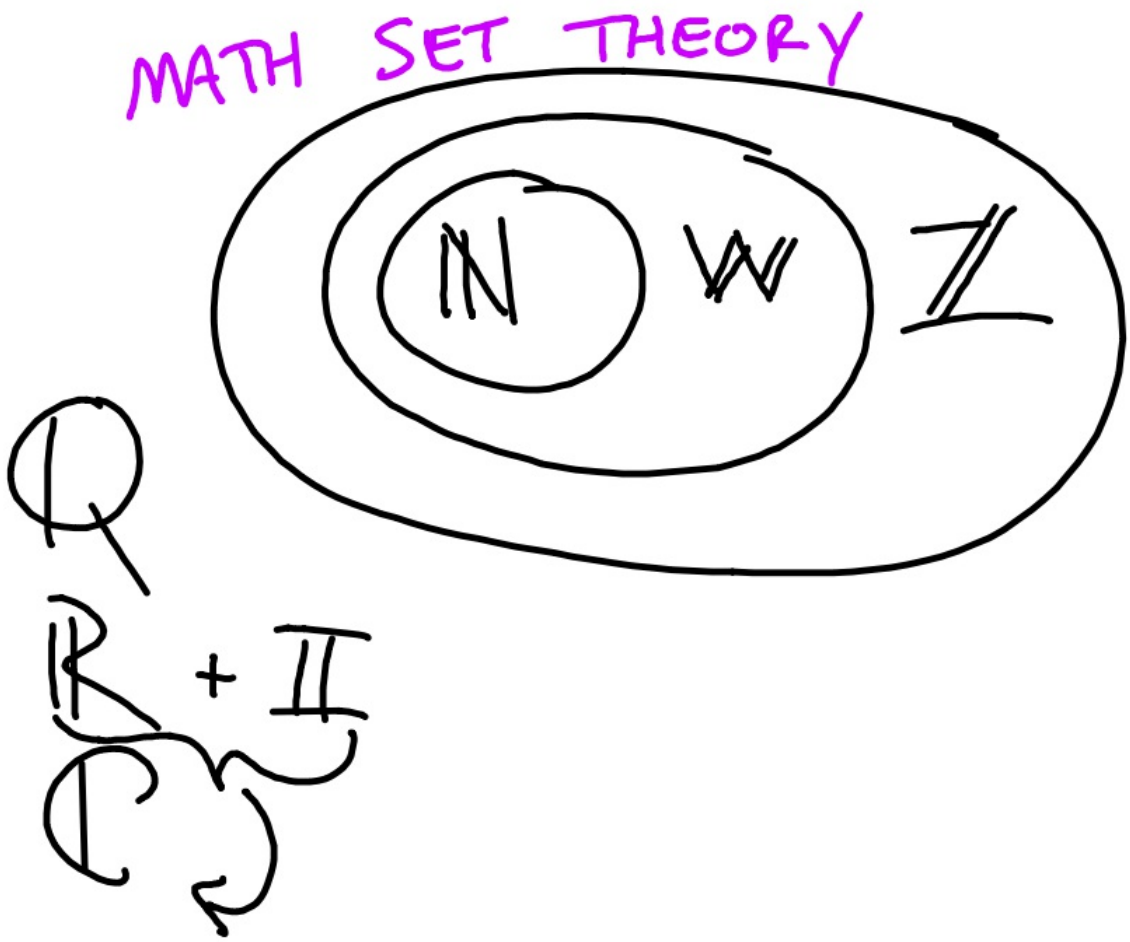
Interface Set

ABOVE SET IN HIERARCHY

All Superinterfaces:
Collection
All Known Subinterfaces:
SortedSet
All Known Implementing Classes:
AbstractSet, HashSet, LinkedHashSet, TreeSet

3. Here is what is known as the deprecation tree for the Set Interface. This interface is part of the Collections framework which is found in java.util The interface javadoc can be found here: https://docs.oracle.com/javase/7/docs/api/java/util/Set.html

4. The implementations we'll focus on in the examples are HashSet and TreeSet.


MATH SET THEORY
IN
W
Z
Q
R
+
II
C

SIMILAR TO: List id → ArrayList

5. Sets don't have duplicates. Adding a duplicate of an element that is already present is silently ignored.

6. To declare and instantiate a Set use the following syntax:

```
Set names = new HashSet();
```

The type of the identifier need only be a Set type. Again, the idea is that to use these we DON'T CARE how they're implemented; we just want to use them, just like with List. We could have easily instantiated this as a TreeSet().

7. Adding and removing set elements is easy:

```
names.add("Romeo");
names.remove("Juliet");
```

ORIGINAL ITER?

8. Because the Set interface is part of the Collections framework, we have an old friend available to us. Use an Iterator to list all the elements of a Set:

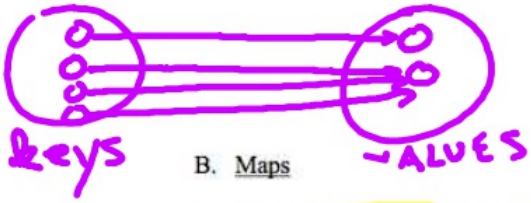
```
Iterator iter = names.iterator();
while(iter.hasNext())
{
      String name = (String)iter.next();
      do something with name
}
```

A Set iterator does not visit the elements in the order in which you inserted them. The set implementation rearranges the elements so that it can locate them quickly. In other words, you can't add an element to a set at an iterator position.

8. Peruse the example codes to get a better idea of how a Set works.

IN LABS FOLDER.

PLAY WITH LEGOS!



B. Maps

1. A Map is a collection that keeps associations between keys and values. Mathematically, a map is a function from one set, the *key set*, to another set, the *value set*. Every key in the map has a unique value, but a value may be associated with several keys.

2. Map is a Java *Interface*, so it dictates what actions a Map must have, but the implementation is left open.

java.util

Interface Map

All Known Subinterfaces:
SortedMap

All Known Implementing Classes:
AbstractMap, Attributes, HashMap, Hashtable, IdentityHashMap, RenderingHints, TreeMap, WeakHashMap

3. Here is what is known as the deprecation tree for the Map Interface. This interface is part of the Collections framework which is found in java.util The interface javadoc can be found here: https://docs.oracle.com/javase/7/docs/api/java/util/Map.html

4. The implementations we'll focus on in the examples are HashMap and TreeMap.

6. To declare and instantiate a Map, use the following syntax:

```
Map favoriteColors = new HashMap();
```

The type of the identifier need only be a Map type. Again, the idea is that to use these we DON'T CARE how they're implemented; we just want to use them. We could have easily declared this as a TreeMap()

7. To create a situation where a persons name is the key and the value is their favorite color we can add to the map in the following way:

```
favoriteColors.put("Juliet", Color.pink);
```

We can also change an association by using put again:

```
favoriteColors.put("Juliet", Color.red);
```

This association is very similar to something we've done before. In the Store lab, we used the "id number" as a way of identifying a product and storing its object into the data structure. A Map works in much the same way. We have a key (the id number) which will "map" to a unique value (the Item object).

8. The get() method returns the value associated with a key:

```
Color julietsFavoriteColor =
   favoriteColors.get("Juliet");
```

When there is no associated value for a key, the get() method will return null.

9. To remove a key and its associated value, use the remove() method:

```
favoriteColors.remove("Juliet");
```

10. Again, we can use an iterator over a Map. Actually, a Map has no iterator inherent to it. But, it does have a keySet() method. The keySet() method will return a Set of all the keys to a Map. To find all keys and values in a map, iterate through the key set and find the values that correspond to the keys. The example works with a Map called m.

```
Set keySet = m.keySet();
Iterator iter = keySet.iterator();
while(iter.hasNext())
{
      Object key = iter.next();
      Object value = m.get(key);
      System.out.println(key + "->" +
                                 value);
}
```

DISPLAYS ALL ASSOCIATIONS IN THIS MAP

11. Peruse the example codes to get a better idea of how a Map works.

C. Tree vs. Hash Implementations

1. If you have a good hash function for your objects, then hashing is usually faster than tree-based algorithms. But the TreeSet and TreeMap classes use a form of **balanced** binary trees that guarantees that adding and removing an element takes O(log(n)) time, whereas the HashSet and HashMap are entirely at the mercy of the hash function.

2. If you don't want to define a hash function, then a TreeSet or TreeMap is an attractive option. TreeSets have another advantage: The iterators visit elements in sorted order rather than the completely random order given by the hash codes. To use a TreeSet, your objects must belong to a class that is Comparable.

SUMMARY/ REVIEW: HashSet, HashMap, TreeSet, and TreeMap are all part of the java.util package and can be used to collect like or similarly deprecated objects.

ASSIGNMENT: Examine and run these files in the following order.

1) HashSetTester.java (note output order)
2) TreeSetTester.java (note output order)
3) HashMapTester.java (note output order)
4) TreeMapTester.java (note output order)