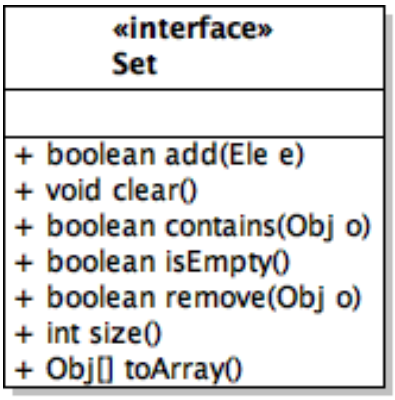
CECS 277 – Lecture 9 – Collections



Set – Sets are used when you want your values to be unique (ie. no duplicate values), and you don't need them to be indexable. They have very fast add(), remove(), and contains() operations. Types of sets include: HashSet, LinkedHashSet, and TreeSet.

HashSet – Adds, removes, and finds elements very quickly. To use a HashSet, the elements of your set must be hashable (which means any classes you create will need to override the hashCode() and equals() methods from the Object class). A hashable value is one that can be used to calculate a storage location in the set. The value is then stored at that location, and can quickly be found again by rehashing the value. Several commonly used types already have the hashCode() and equals() methods overridden: String, Integer, Point, Rectangle, and Color.

Example: A set of strings stored as a HashSet.

```
HashSet<String> food = new HashSet<String>();
food.add( "Pizza" );
food.add( "Ice Cream" );
food.add( "Pizza" );  //no duplicates, not added
for( String s : food )
     System.out.println( s );
```

LinkedHashSet – Has nearly the same performance as a HashSet. It is implemented by keeping a linked list of the hashed locations, and as such, preserves insertion order.

Example: A set of strings stored as a LinkedHashSet.

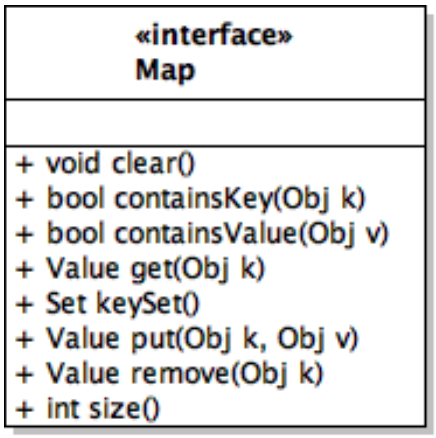
```
LinkedHashSet<String>food = new LinkedHashSet<String>();
food.add( "Pizza" );
food.add( "Ice Cream" );
for( String s : food )
     System.out.println( s );
```

TreeSet – A TreeSet stores your elements in sorted order in so that the container can quickly locate them in the tree. TreeSet's add, remove, and contains functions are not as fast as HashSet, but it does provide additional functions to access the elements. Any objects being placed into a TreeSet are required to implement the Comparable interface and override the compareTo() method. Common classes that implement Comparable: Integer, Double, Boolean, Date, Character, String, Time,

Example: Student objects stored in a TreeSet, compared by id number.

```
TreeSet<Student> students = new TreeSet<Student>();
students.add( new Student( "Jones", 12395 ));
students.add( new Student( "Smith", 12390 ));
System.out.println( students.first() );  //Smith 12390
System.out.println( students.last() );   //Jones 12395
```

Map – A map is a type of collection that stores elements based on a key-value pair. Each key is associated with a value, and the key is used to locate that value. Because of this, maps cannot have duplicate keys, however, they can have duplicate values. A few types of maps are: HashMap, TreeMap, and LinkedHashMap.



HashMap – Stores and accesses elements quickly. In order to use a HashMap, the key elements of your map must be hashable (your class will need to override the hashCode() and equals() methods).

Example: Names and grades stored in a HashMap.

```
HashMap <String, Integer> grades =
                new HashMap <String, Integer>();
grades.put( "Stewart", 92 );
grades.put( "Mary", 80 );
grades.put( "George", 67 );
grades.put( "Stewart", 80 ); //overwrites old grade
System.out.println( "Size = " + grades.size() );
if( grades.containsValue( 100 ) )
     System.out.println( "Perfect score recorded" );
```

LinkedHashMap – Has nearly the same performance as a HashMap. It is implemented by keeping a linked list of the hashed locations, and as such, preserves insertion order.

Example: A set of strings stored as a LinkedHashMap.

```
LinkedHashMap<String, Integer> grades =
                new LinkedHashMap<String, Integer>();
grades.put( "Stewart", 92 );
grades.put( "Mary", 80 );
grades.put( "George", 67 );
grades.put( "Scott", 80 );
Set <String> keys = grades.keySet();
for( String s : keys )
     System.out.println(s+" got a "+ grades.get( s ));
```

TreeMap – A TreeMap stores your elements in sorted order. The objects stored as keys will need to implement the Comparable interface.

Example: A set of strings stored as a TreeMap.

```
TreeMap<String, Integer> grades =
                new TreeMap<String, Integer>();
grades.put( "Stewart", 92 );
grades.put( "Mary", 80 );
grades.put( "George", 67 );
grades.put( "Scott", 80 );
String first = grades.firstKey();  //George
String last = grades.lastKey();    //Stewart
```