Sets and Maps

- Sets
- Maps
- The Comparator Interface
- Sets and Maps in Java Collections API
  - TreeSet
  - TreeMap
- Review for Exam
- Reading: 13.1-13.6

Sets

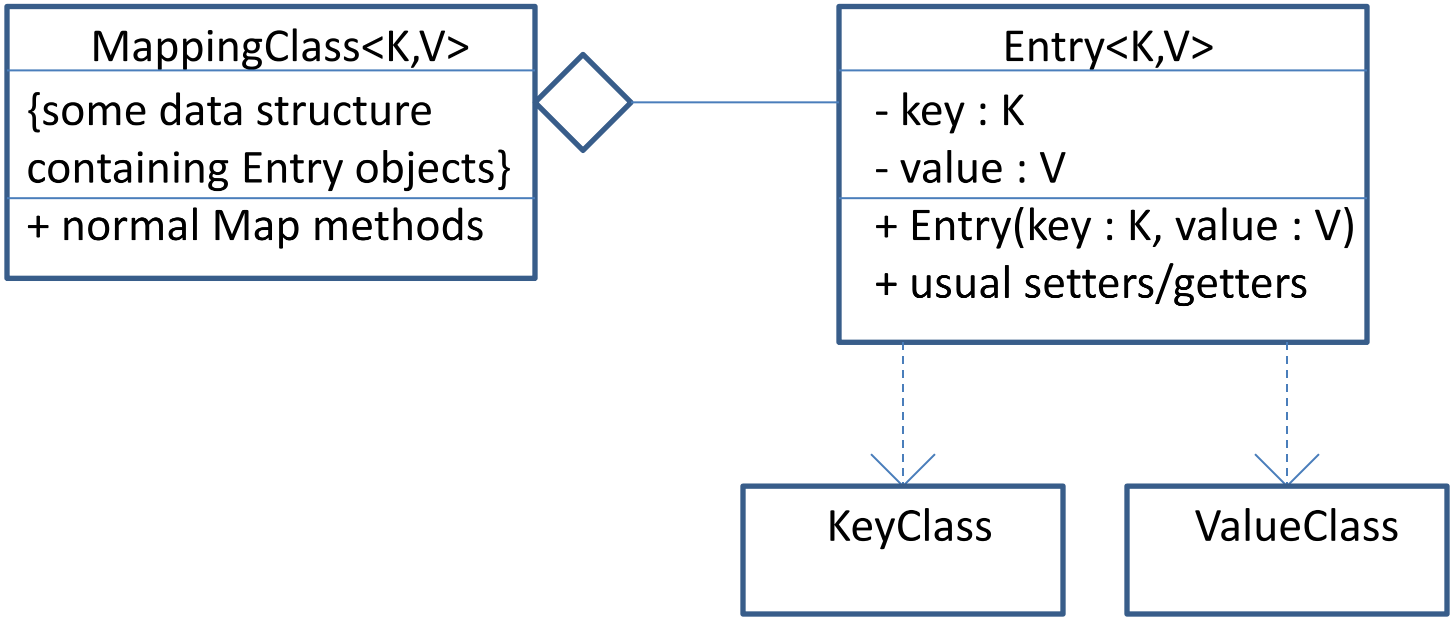
- A set is a collection of elements with no duplicates
- We had a set application in Lab 3 where we needed a data structure to represent a drum full of Bingo balls for random drawing
- Since there should only be one Bingo ball with each number, the correct type of collection is a set

Maps

- A map is a collection that establishes a relationship between keys and values
- The implementation should provide an efficient way to retrieve a value given its key
- There must be a one-to-one mapping from a key to a value – each key must have only one value, but multiple keys can have the same value
- Therefore, there isn’t necessarily a one-to-one mapping from a value to a key

Map Entry Class

- To implement a map collection using any data structure, we need a class for objects that link a key with its corresponding value



Map Entry Class

- The Entry class is not used outside its Map class
- The Entry class code is usually written as an inner class of the Map class that it supports

The Comparator Interface

- In the Java Collections API, either the Comparator or Comparable interface may be used
- A Comparator class object can be passed to the collection class's constructor to use in comparing
- The Comparator's "compare" method takes two objects as parameters and returns a value like the Comparable compareTo method does (< 0, 0, or > 0 representing <, ==, or >)
- The compare method is not implemented within the key class but uses two objects of that class

The Comparator Interface

- Implementing a Comparator for Strings that uses their length as the basis for the comparison

```
public class StringComparator
            implements Comparator<String>
{
  public int compare(String s1, String s2)
  {
    return s1.length() - s2.length();
  }
}
```

Java Collections API: Implementing Sets and Maps

- The Java class library provides thorough and efficient implementations of underlying binary search trees in these two classes:
  - TreeSet
  - TreeMap
- Both of those classes can be used with either the normal ordering of the elements (via the Comparable interface) or via a Comparator

TreeSet<T>

- In a TreeSet, we store elements in an order determined either by their natural ordering (based on their CompareTo method) or an ordering based on a provided Comparator
- Each element stored in a TreeSet contains all of the data associated with that object
- The TreeSet class implements a set using a Red/Black binary search tree for efficiency in the add, contains, and remove operations

TreeSet<T>

- Some of the TreeSet unique methods are:

```
TreeSet() // constructs a new set sorted
  according to natural order of the objects
TreeSet (Comparator<T> c) // constructs a
  new set sorted according to Comparator c
boolean add(T o) // adds the specified
  element to the set if not already present
boolean contains(Object o) // returns true
  if this object is present in the set
boolean remove(Object o)  // removes this
  element from the set if it is present
```

TreeMap<K,V>

- In a TreeMap, we separate the data being stored into a key and the rest of the data (the value)
- Internally, node objects are stored in the tree
- Each node object contains
  - a reference to the key
  - a reference to the object containing the rest of the data
  - two links to the child nodes
  - and a link to the parent node
- The TreeMap class implements a map using a Red/Black binary search tree

TreeMap<K,V>

- Some of the TreeMap unique methods are:

```
TreeMap ()   // constructs a new map sorted
  according to natural order of the objects
TreeMap (Comparator<K> c)  // constructs a
  new map sorted according to Comparator c
V put(K key, V value)  // associates the
  value with the key
boolean containsKey(Object key) // returns
  true if the map contains a mapping for key
boolean containsValue(Object value) //
  returns true if the mapping contains a key
  value pair for this value
V get(Object key) // returns the value V
  mapped to the key
```

Using Set/Map APIs with a Comparator

- Instantiate the Comparator

```
Comparator<String> comp
            = new StringComparator();
```

- Instantiate a TreeSet containing Strings

```
TreeSet<String> mySet
= new TreeSet<String>(comp);
```

- Instantiate a TreeMap with Strings as keys

```
TreeMap<String,ValueClass> myTree
= new TreeMap<String,ValueClass>(comp);
```

Set and Map Efficiency

- The TreeSet and TreeMap classes provide O(log n) access to their data
- When the sort order is not important, there is a more efficient way to implement sets and maps with a data structure called a hash table
- A hash table provides approximately O(1) access to its data and will be covered in CS310

Review for Exam

- Review for Exam 3
  - Practice exam is on-line
  - Question and Answer Session
- Exam 3
  - Open Book / Open Notes
  - No Electronic Devices (calculators, laptops, etc)
  - Students with E-books must sit in front row so that I can see what’s on their screen at any time