Sets and Maps

Weiss, sections 6.7 and 6.8 and from section 6.10 to the end of chapter 6 (pages 261–274 and 277–292)

I am indebted to my colleagues, John David Stone and Peter-Michael Osera, for assistance with this handout. This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/4.0/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.
Set

• A collection of elements without duplicates
• Start from an empty set and add elements to it
  \{ 1, 4, 2, 9, 8 \}
  After adding 4:  \{ 1, 4, 2, 9, 8 \}
• Like list, set supports adding, removing, and iterating over elements
• Unlike list, set elements do not have indices.
  – add ( AnyType a )
  – remove( AnyType a )
  – contains( AnyType a)
SortedSet

- Maintain items in sorted order
  - Iterate through the sorted elements
- Support finding the smallest and largest elements
- TreeSet implements SortedSet

```java
Set<String> names = new TreeSet<String>();
names.add(joe); names.add(hal); names.add(bob);
names: [bob, joe, hal]
```
HashSet

• Store elements in an unpredictable order
  Set<String> names = new HashSet<String>();
  names.add( hal );
  names.add( joe );
  names.add( bob );
  [joe, hal, bob]

• Use hash table to store elements
  – Add, remove, and contains take O(1) time.
equals method

- Need to override the equals method to avoid duplicates in sets
- How to override the equals method for inheritance

```java
public static void main( String[ ] args ) { 
    BaseClass a = new BaseClass( 5 );
    DerivedClass b = new DerivedClass( 5,8 );
    DerivedClass c = new DerivedClass( 5,8 );
    System.out.println( b.equals( c ) );
    System.out.println( a.equals( b ) );
    System.out.println( b.equals( a ) );
}
```
equals method (cont.)

Class BaseClass {
    private int x;
    public BaseClass( int i ) { x = i; }
    public boolean equals( Object rhs ){
        if( rhs == null || getClass( ) != rhs.getClass( ) )
            return false;
        return x == ( (BaseClass) rhs ).x;
    }
}

class DerivedClass {
    private int y;
    public DerivedClass( int i, int j ) { super( i ), y = j }
    public boolean equals( Object rhs ){
        return super.equals( rhs ) && y == ( ( DerivedClass ) rhs ).y;
    }
}
Maps

• Capture mapping from keys of type K to values of type V (keys are unique)
• Call each key-value pair \((k, v)\) an entry in the map
  – Dictionary: keys and values are strings
  – Account holder-balance: keys are string and values are integer
• Map operations

  \[
  \begin{align*}
  \text{put}( K k, V v ) & \quad \text{containsKey}( K k ) \\
  \text{remove}( K k ) & \quad \text{keys} \\
  \text{get}( K k ) & \quad \text{values}( )
  \end{align*}
  \]
Maps (cont.)

- Implement map as a list of key-value pairs
- Association list

```java
public class Pair<T, U> {
    public T fst;
    public U snd;
    public Pair(T fst, U snd) {
        this.fst = fst;
        this.snd = snd;
    }
}
```

Implement map as HashMap

```java
Map<String, Integer> m = new HashMap<String, Integer>();
m.put("dog", 3);
...
Set<String> keys = m.keySet();
Collection<Integer> values = m.values();
```