Maps and Sets

Topics

1. Sets: HashSet and TreeSet
2. Maps: HashMaps and TreeMaps
3. Iterators with Sets and Maps

Sets

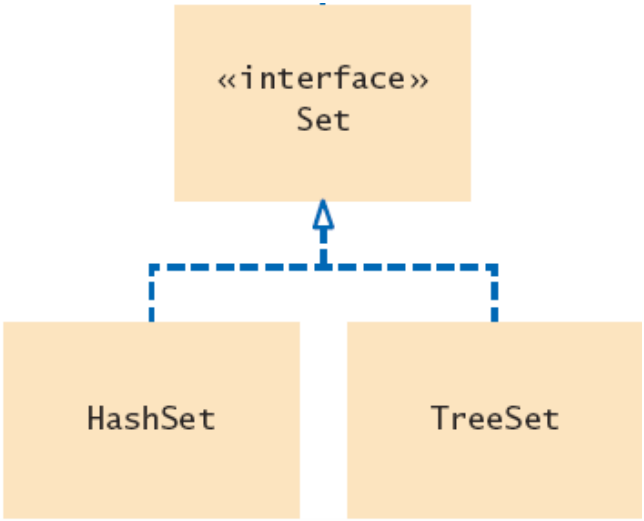
15.3 Sets

- A set is an unordered collection
  - It does not support duplicate elements
- The collection does not keep track of the order in which elements have been added
  - Therefore, it can carry out its operations more efficientl an ordered collection

The `HashSet` and `TreeSet` classes both implement the Set interface.

Sets

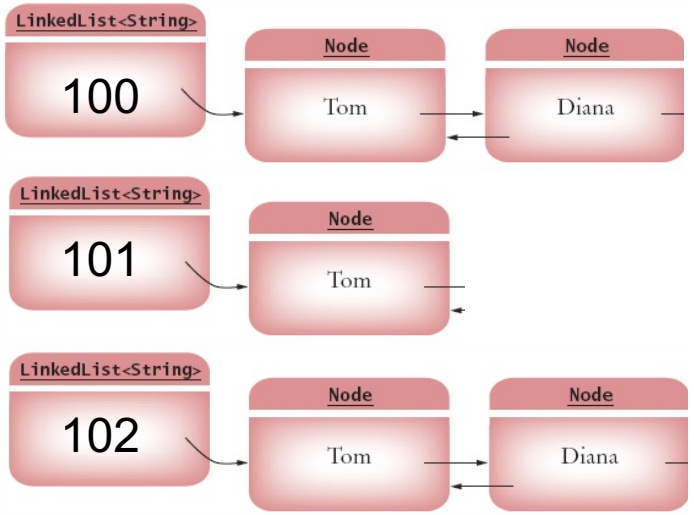
- `HashSet`: Stores data in a Hash Table
- `TreeSet`: Stores data in a Binary Tree
- Both implementations arrange the set elements so that finding, adding, and removing elements is efficient



Set implementations arrange the elements so that they can locate them quickly

Hash Table Concept

- Set elements are grouped into smaller collections of elements that share the same characteristic
  - It is usually based on the result of a mathematical calculation on the contents that results in an integer value
  - In order to be stored in a hash table, elements must have a method to compute their integer values



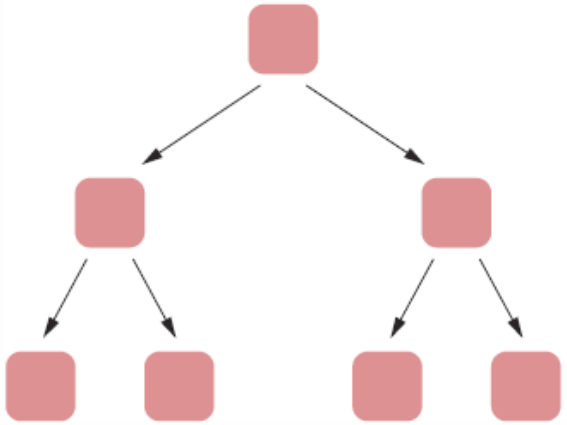
hashCode

- The method is called `hashCode`
  - If multiple elements have the same hash code, they are stored in an ordered list (Linked list)
- The elements must also have an `equals` method for checking whether an element equals another like:
  - String, BankAccount, and all collection classes
- Default `hashCode` just uses address, so often need to override
  - Unless already overridden like String

```
Set<String> names = new HashSet<String>();
```

Tree Concept

- Set elements are kept in sorted order
  - Nodes are not arranged in a linear sequence but in a tree shape



  - In order to use a `TreeSet`, it must be possible to compare the elements and determine which one is "larger"

Iterators and Sets

- Iterators are also used when processing sets
  - `hasNext` returns true if there is a next element
  - `next` returns a reference to the value of the next element
  - `add` via the iterator is not supported for TreeSet and HashSet

```
Iterator<String> iter = names.iterator();
while (iter.hasNext())
{
  String name = iter.next();
  // Do something with name
}
```

```
for (String name : names)
{
  // Do something with name
}
```

  - Note that the elements are not visited in the order in which you inserted them.
  - They are visited in the order in which the set keeps them:
    - Seemingly random order for a HashSet
    - Sorted order for a TreeSet

Working With Sets (1)

Table 4 Working with Sets	
`Set<String> names;`	Use the interface type for variable declarations.
`names = new HashSet<String>();`	Use a `TreeSet` if you need to visit the elements in sorted order.
`names.add("Romeo");`	Now `names.size()` is 1.
`names.add("Fred");`	Now `names.size()` is 2.
`names.add("Romeo");`	`names.size()` is still 2. You can't add duplicates.
`if (names.contains("Fred"))`	The `contains` method checks whether a value is contained in the set. In this case, the method returns `true`.

Working With Sets (2)

Table 4 Working with Sets

`System.out.println(names);`	Prints the set in the format `[Fred, Romeo]`. The elements need not be shown in the order in which they were inserted.

`for (String name : names)`
`{`
`. . .`
`}`
	Use this loop to visit all elements of a set.

`names.remove("Romeo");`	Now `names.size()` is 1.
`names.remove("Juliet");`	It is not an error to remove an element that is not present. The method call has no effect.

SpellCheck.java (1)

```
1  import java.util.HashSet;
2  import java.util.Scanner;
3  import java.util.Set;
4  import java.io.File;
5  import java.io.FileNotFoundException;
6
7  /**
8     This program checks which words in a file are not present in a dictionary.
9  */
10 public class SpellCheck
11 {
12    public static void main(String[] args)
13       throws FileNotFoundException
14    {
15       // Read the dictionary and the document
16
17       Set<String> dictionaryWords = readWords("words");
18       Set<String> documentWords = readWords("alice30.txt");
19
20       // Print all words that are in the document but not the dictionary
21
22       for (String word : documentWords)
23       {
24          if (!dictionaryWords.contains(word))
25          {
26             System.out.println(word);
27          }
28       }
```

SpellCheck.java (2)

```
29    }
30
31    /**
32       Reads all words from a file.
33       @param filename the name of the file
34       @return a set with all lowercased words in the file. Here, a
35       word is a sequence of upper- and lowercase letters.
36    */
37    public static Set<String> readWords(String filename)
38       throws FileNotFoundException
39    {
40       Set<String> words = new HashSet<String>();
41       Scanner in = new Scanner(new File(filename));
42       // Use any characters other than a-z or A-Z as delimiters
43       in.useDelimiter("[^a-zA-Z]+");
44       while (in.hasNext())
45       {
46          words.add(in.next().toLowerCase());
47       }
48       return words;
49    }
50 }
```

Program Run

```
neighbouring
croqueted
pennyworth
dutchess
comfits
xii
dinn
clamour
...
```

Programming Tip 15.1

- Use Interface References to Manipulate Data Structures
  - It is considered good style to store a reference to a `HashSet` or `TreeSet` in a variable of type `Set`.

```
Set<String> words = new HashSet<String>();
```

    - This way, you have to change only one line if you decide to use a `TreeSet` instead.

Programming Tip 15.1 (continued)

- Unfortunately the same is not true of the `ArrayList`, `LinkedList` and `List` classes
  - The `get` and `set` methods for random access are very inefficient
- Also, if a method can operate on arbitrary collections, use the `Collection` interface type for the parameter:

```
public static void removeLongWords(Collection<String> words)
```

Maps

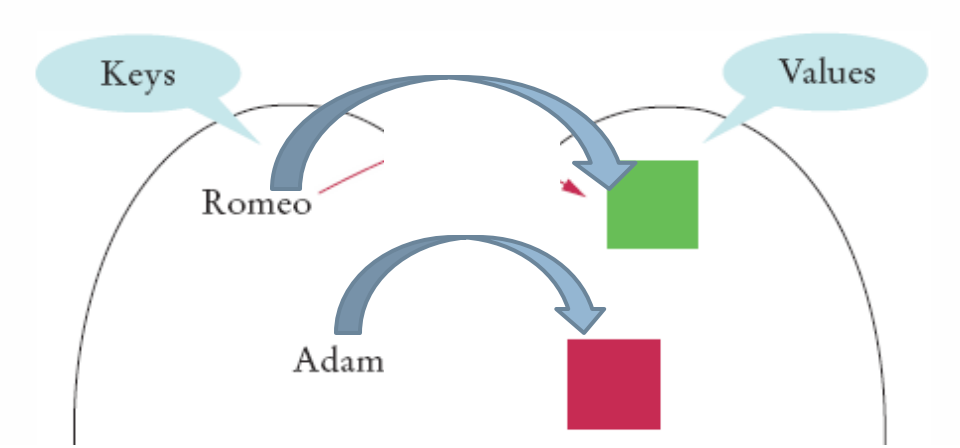
15.4 Maps

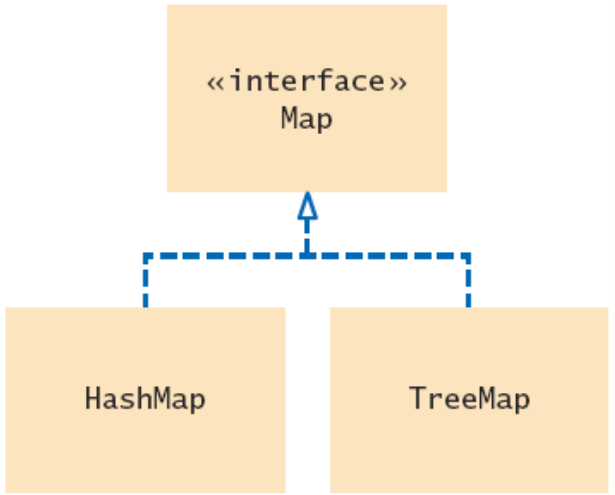
- A map allows you to associate elements from a key set with elements from a value collection.
  - The `HashMap` and `TreeMap` classes both implement the `Map` interface.
  - Use a map to look up objects by using a key.
- Only key needs a good hashCode to use HashMap
- Only the key need to be Comparable for TreeMap

Maps

Key Value Key Value

```
Map<String, Color> favoriteColors = new HashMap<String, Color>();
```





Working with Maps (Table 5)

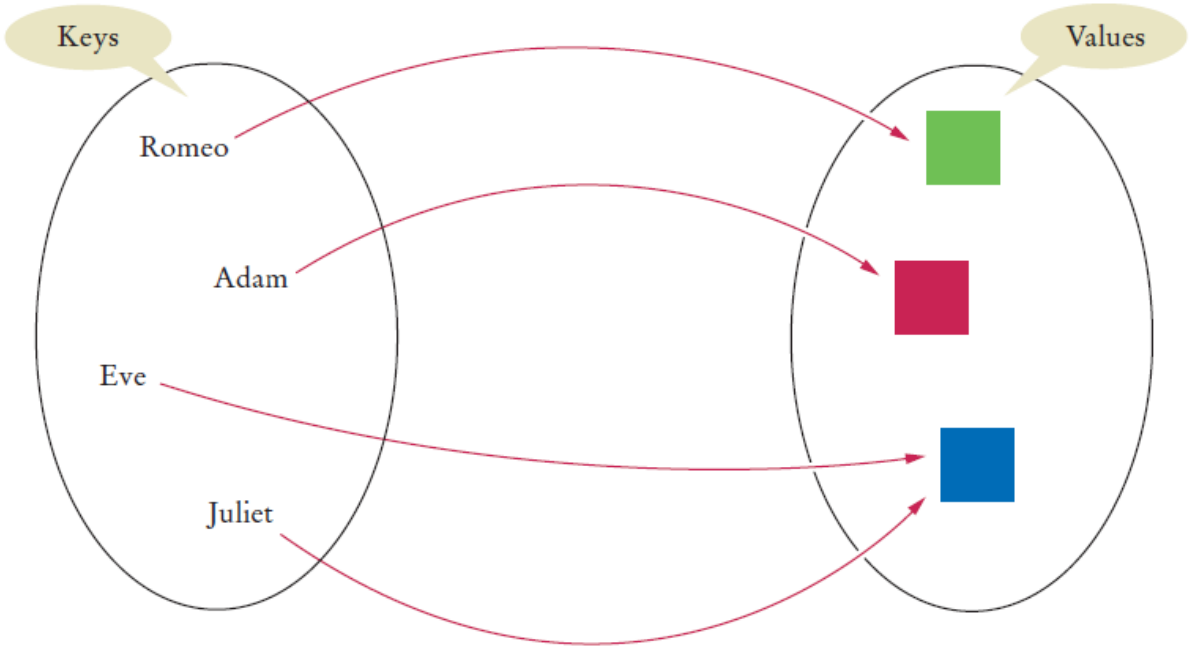
`Map<String, Integer> scores;`	Keys are strings, values are `Integer` wrappers. Use the interface type for variable declarations.
`scores = new TreeMap<String, Integer>();`	Use a `HashMap` if you don't need to visit the keys in sorted order.
`scores.put("Harry", 90);` `scores.put("Sally", 95);`	Adds keys and values to the map.
`scores.put("Sally", 100);`	Modifies the value of an existing key.
`int n = scores.get("Sally");` `Integer n2 = scores.get("Diana");`	Gets the value associated with a key, or `null` if the key is not present. `n` is 100, `n2` is `null`.
`System.out.println(scores);`	Prints `scores.toString()`, a string of the form `{Harry=90, Sally=100}`

`for (String key : scores.keySet())`
`{`
`  Integer value = scores.get(key);`
`  . . .`
`}`
	Iterates through all map keys and values.

`scores.remove("Sally");`	Removes the key and value.

Key Value Pairs in Maps

- Each key is associated with a value




```
Map<String, Color> favoriteColors = new HashMap<String, Color>();
favoriteColors.put("Juliet", Color.RED);
favoriteColors.put(“Romeo", Color.GREEN);
Color julietsFavoriteColor = favoriteColors.get("Juliet");
favoriteColors.remove("Juliet");
```

Iterating through Maps

- To iterate through the map, use a `keySet` to get the list of keys:

```
Set<String> keySet = m.keySet();
for (String key : keySet)
{
  Color value = m.get(key);
  System.out.println(key + "->" + value);
}
```

To find all values in a map, iterate through the key set and find the values that correspond to the keys.

MapDemo.java

```
1  import java.awt.Color;
2  import java.util.HashMap;
3  import java.util.Map;
4  import java.util.Set;
5
6  /**
7     This program demonstrates a map that maps names to colors.
8  */
9  public class MapDemo
10 {
11    public static void main(String[] args)
12    {
13       Map<String, Color> favoriteColors = new HashMap<String, Color>();
14       favoriteColors.put("Juliet", Color.BLUE);
15       favoriteColors.put("Romeo", Color.GREEN);
16       favoriteColors.put("Adam", Color.RED);
17       favoriteColors.put("Eve", Color.BLUE);
18
19       // Print all keys and values in the map
20
21       Set<String> keySet = favoriteColors.keySet();
22       for (String key : keySet)
23       {
24          Color value = favoriteColors.get(key);
25          System.out.println(key + " : " + value);
26       }
27    }
28 }
```

Program Run

```
Juliet : java.awt.Color[r=0,g=0,b=255]
Adam : java.awt.Color[r=255,g=0,b=0]
Eve : java.awt.Color[r=0,g=0,b=255]
Romeo : java.awt.Color[r=0,g=255,b=0]
```

Topics

1. Sets: HashSet and TreeSet
2. Maps: HashMaps and TreeMaps
3. Iterators with Sets and Maps