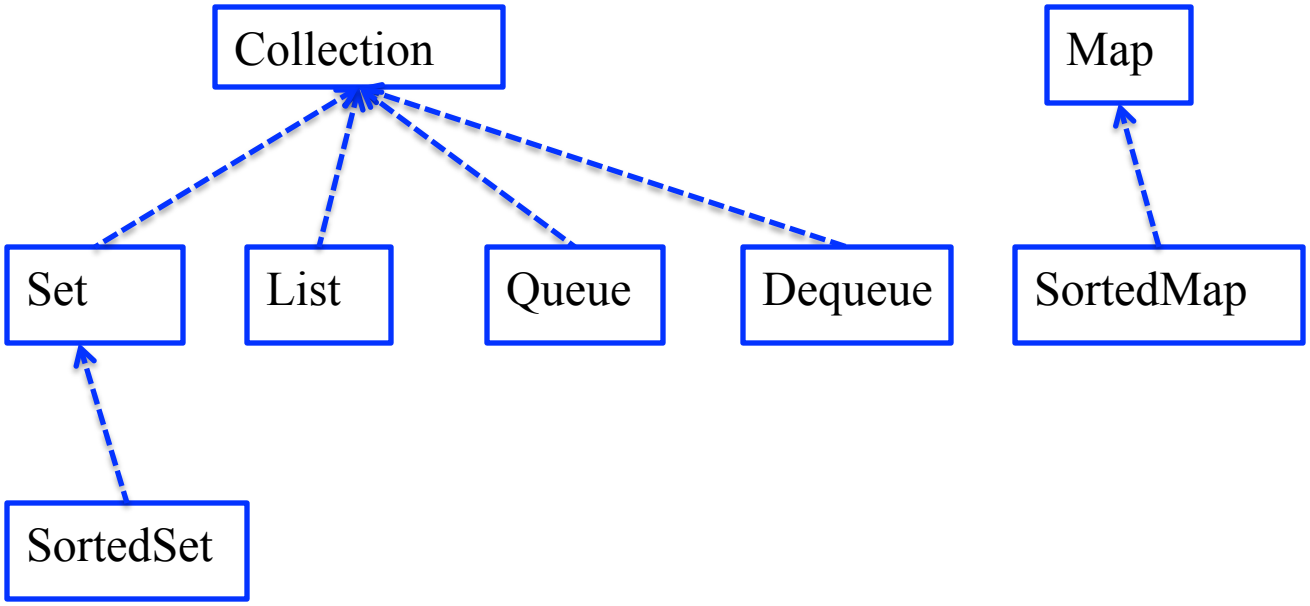
Collections:
ArrayList
HashMap

IST 256

Application Programming for Information Systems

Collections

- Informally, a Java collection is “an object that groups multiple elements into a single unit”
  - Arrays are built into the language core
  - All other collections are given as classes that implement one of the the collection interfaces, headed by Collection and Map



ArrayList

- Lists's most commonly used implementation is the ArrayList
- Provides the same functionality as primitive arrays, but you don't need to declare a size
  - this is a big advantage when you don't know a maximum number of elements in advance
  - Add elements and index them with integers from 0 to the number

array

index	value
0	**7**
1	4
2	0
3	23
4	-5
5	14

ArrayList

index	value
0	**7**
1	4
2	0
3	23
4	-5
5	14

Using ArrayLists

- ArrayList declaration – give the type of the elements

  ArrayList<Integer> myList = new ArrayList<Integer>()

- ArrayList methods
  - myList.add(number) // adds int number to the end of the arraylist
  - myList.get(index) // returns the integer stored at index
  - myList.size() // returns the length of the ArrayList
  - myList.set(index, number) // assigns number to be the element at index

ArrayList Example

- Example makes an ArrayList of Strings

```
import java.util.ArrayList;

    //Create a list of names
   ArrayList names = new ArrayList();

    //Add some names in list
    names.add("Eve");
    names.add("Anna");
    names.add("Tony");
    names.add("Steve");

    //Print out the names at each index using a regular for loop
    for (int i = 0; i < names.size(); i++) {
        System.out.println(names.get(i));
    }
```

HashMap

- A map is a data structure where we can use Strings or arbitrary numbers to index into the data values
  - Other languages call these dictionaries, hash tables or associative arrays

array

index	**value**
0	**7**
1	4
2	0
3	23
4	-5
5	14

hashmap

index	**value**
“a”	**7**
“b”	4
“c”	0
“d”	23
“e”	-5
“f”	14

Using a HashMap

- Declaring a HashMap
  - Give the type of the indices and the type of the values
    HashMap<String,Integer> myHashMap =
    new HashMap<String,Integer>()
- Methods, for a String key and int number:
  - myHashMap.put(key, number) // adds value under the key
  - myHashMap.get(key) // returns the value stored under the key
  - myHashMap.size() // returns the number of elements
  - myHashMap.keySet() // returns the keys

HashMap Examples

- Example creates a hashmap where the student's names are keys

```
import java.util.HashMap;

HashMap<String , String> studentGrades = new HashMap<String, String>();

    // add key/value pairs
    studentGrades.put("Alvin", "A+");
    studentGrades.put("Becca", "A-");
    studentGrades.put("Chuck", "B+");

    //find element using key
    System.out.println("Becca's Marks:: "+studentGrades.get("Becca"));

    }
```

“for each” loops over Collections

- Java provides another type of for loop to go over Collections that iterates without an index number
  - Formally called an *advanced for loop*
- Format

```
for  (<type> <iterator variable> : <collection>)
{
// in the body of the loop use the iterator variable
//     for one element of the collection
}
```

For Each over HashMap

- For a HashMap, although we can iterate over the values, we most likely want to iterate over the keys of the HashMap, using each key to get the value

```
HashMap<String , String> studentGrades = new HashMap<String, String>();

// add key/value pairs
studentGrades.put("Alvin", "A+");
studentGrades.put("Becca", "A-");
studentGrades.put("Chuck", "B+");

// iterate over keys of the HashMap
for(String key: studentGrades.keySet()){
    System.out.println(key  +" :: "+ studentGrades.get(key));
```

For Each over ArrayList

- You can also use the *for each* loop on an ArrayList if you don't need to know the index of the element

```
//Create a list of names
List names = new ArrayList();

//Add some names in list
names.add("Eve");
names.add("Anna");
names.add("Tonny");
names.add("Steve");

//Iterate using enhanced for loop (for each)
for (String name : names) {
   System.out.println(name);
}
```

Use HashMaps for Quick Lookup by Key

- The HashMap class is implemented so that it is fast to access an element by its key
- Suppose that you know that you repeatedly want to access your data by some feature such as an id number or a product code
- Build a HashMap that uses the id for a key

- Example: Suppose each Student has an id field and we have an array or ArrayList of objects of type Student. We can build a HashMap using the id field as the keys:

```
HashMap<String> studentMap = new HashMap<String>()
for (Student oneStudent : studentArray)
{    studentMap.put(oneStudent.getID(), oneStudent)    }
```

- In later code, we can then get a Student object by using an id String, selectedID:

```
Student selectedStudent = studentMap.get(selectedID)
```