

# OBJECT ORIENTED PROGRAMMING UNIT4: STORAGE STRUCTURES

# Index

2

- Collections
- ArrayList
- Iterator
- Maps
- Sets

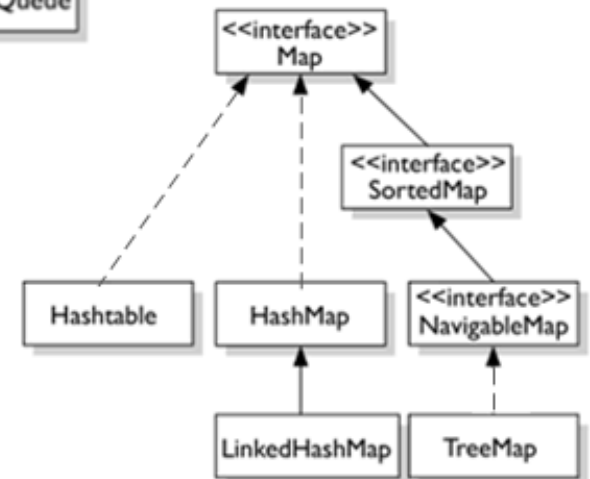
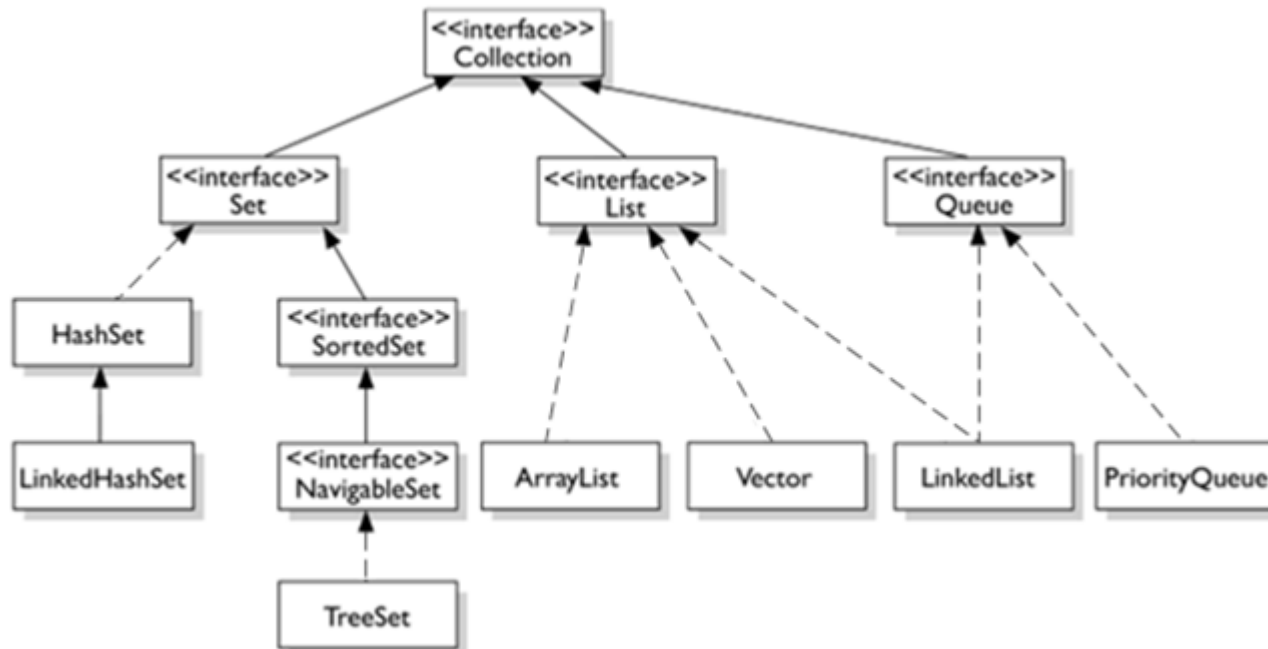
# 4.1 2 Collections

- A collection is an object that holds a group of objects. Some collections:
  - ▣ allow duplicate elements (and others do not)
  - ▣ are ordered (and others are unordered)
  - ▣ restrict access to elements according to certain rules (and others allow unrestricted access)
- A collection is something that can be added to and removed from; you can test membership of items; you can ask for its size and whether it is empty. You can dump its items into an array.
- Once you understand how to use one of them, you can use them all.

# 4.1 2 Collections

4

- The primary kinds of collections are:



# 4.1 3 ArrayList

5

- ArrayList is a standard class that encapsulates the functionality of an array but allows it to expand automatically (*resizable*).
- Because ArrayList is a class and isn't part of the syntax of Java, you can't use Java's array syntax; you must use methods to access the ArrayList's data.

# 4.1 3 ArrayList

6

## ➤ Some of the methods:

| Method signature  | Usage  |
|---|--|
| <code>add(Object o)</code>                                  | Add the given element at the end                         |
| <code>add(int i, Object o)</code>                           | Insert the given element at the specified position       |
| <code>clear()</code>  | Remove all element references from the Collection        |
| <code>contains(Object o)</code>                             | True if the List contains the given Object               |
| <code>get(int i)</code>                                     | Return the object reference at the specified position    |
| <code>indexOf(Object o)</code>                              | Return the index where the given object is found, or -1  |
| <code>remove(Object o)</code><br><code>remove(int i)</code> | Remove an object by reference or by position             |
| <code>toArray()</code>                                      | Return an array containing the objects in the Collection |

# 4.13 ArrayList

7

```
public class pruebaArrayList {  
  
    public static void main(String[] args) {  
        List<String> lst = new ArrayList<String>(); // Inform compiler about the  
                                                    // type  
        lst.add("alpha"); // compiler checks if argument's type is String  
        lst.add("beta");  
        lst.add("charlie");  
        System.out.println(lst); // [alpha, beta, charlie]  
  
        for (String str : lst) {  
            System.out.println(str);  
        }  
    }  
}
```



# 4.13 ArrayList

8



A.4.9.

1) Open the `MusicOrganizer` file and create a `MusicOrganizer` object. Store the names of a few audio files into it - they are simply strings. Check that the number of files returned by `numberOfFiles` matches the number you stored.

When you use the `listFile` method, you will need to use a parameter value of 0 to print the first file, 1 (one) to print the second, and so on.





# 4.13 ArrayList

9



2) Continuing with the previous project, what happens if you create a new MusicOrganizer object and then call `removeFile(0)` before you have added any files to it? Do you get an error? Would you expect to get an error?



3) Create a MusicOrganizer and add two file names to it. Call `listFile(0)` and `listFile(1)` to show the two files. Now call `removeFile(0)` and then `listFile(0)`. What happened? Is that what you expected? Can you find an explanation of what might have happened when you removed the first file name from the collection?



# 4.13 ArrayList

10



4) Add a method called `checkIndex` to the `MusicOrganizer` class. It takes a single integer parameter and checks whether it is a valid index for the current state of the collection.


To be valid, the parameter must lie in the range 0 to `size()-1`.

If the parameter is not valid, then it should print an error message saying what the valid range is. If the index is valid, then it prints nothing. Test your method on the object `bench` with both valid and invalid parameters. Does your method still work when you check an index if the collection is empty?




# 4.13 ArrayList

11



5) Write an alternative version of `checkIndex` called `validIndex`. It takes an integer parameter and returns a boolean result. It does not print anything, but returns `true` if the parameter's value is a valid index for the current state of the collection and `false` otherwise.

Test your method on the object bench with both valid and invalid parameters. Test the empty case too.



6) Rewrite both the `listFile` and `removeFile` methods in `MusicOrganizer` so that they use your `validIndex` method to check their parameter, instead of the current boolean expression. They should only call `get` or `remove` on the `ArrayList` if `validIndex` returns `true`.




# 4.13 ArrayList

12



7) Add a method called `listAllFiles()`, that prints out every file name.



8) Add the following method.  
Can you find a way to print a message, once the for-each loop has finished, if no file names matched the search string? *Hint:* Use a **boolean** local variable.

```
/**
 * List the names of files matching the given search string.
 * @param searchString The string to match.
 */
public void listMatching(String searchString)
{
    for(String filename : files) {
        if(filename.contains(searchString)) {
            // A match.
            System.out.println(filename);
        }
    }
}
```



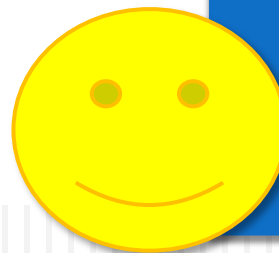
# 4.14 Iterator

13

- An Iterator provides just three methods, and two of these are used to iterate over a collection: `hasNext` and `next`.
- For instance, in our example, we can add a method like this:

```
public void listAllFiles() {  
    /*  
     * for (String filename : files) { System.out  
     */  
    Iterator<String> it = files.iterator();  
    while (it.hasNext()) {  
        System.out.println(it.next());  
    }  
}
```

1. We use a while loop, but we do not need to take care of the index variable.
2. Once `hasNext` has returned false, it would be an error to try to call `next` on that particular Iterator object—in effect, the Iterator object has been “used up” and has no further use.



# 4.14 Iterator

14

```
public class pruebaArrayList {  
    public static void main(String[] args) {  
        List<String> lst = new ArrayList<String>(); // Inform compiler about the  
                                                    // type  
        lst.add("alpha"); // compiler checks if argument's type is String  
        lst.add("beta");  
        lst.add("charlie");  
        System.out.println(lst); // [alpha, beta, charlie]  
  
        for (String str : lst) {  
            System.out.println(str);  
        }  
  
        Iterator<String> iter = lst.iterator(); // Iterator of Strings  
        while (iter.hasNext()) {  
            String str = iter.next(); // compiler inserts downcast operator  
            System.out.println(str);  
        }  
    }  
}
```



# 4.14 Iterator

15



A.4.10. Use the club project to complete the following exercises. Your task is to complete the Club class, an outline of which has been provided in the project. The Club class is intended to store Membership objects in a collection.

1) Within Club, define a field for an ArrayList. Think carefully about the element type of the list. In the constructor, create the collection object and assign it to the field. Make sure that all the files in the project compile before moving on to the next exercise.



# 4.14 Iterator

16



2) Complete the **numberOfMembers** method to return the current size of the collection. Until you have a method to add objects to the collection, this will always return zero.





# 4.14 Iterator



3) Membership of a club is represented by an instance of the **Membership** class. An instance of **Membership** contains details of a person's name and the month and year in which they joined the club. All membership details are filled out when an instance is created. A new **Membership** object is added to a **Club** object's collection via the **Club** object's **join** method, which has the following description:

```
/**
```

```
* Add a new member to the club's collection of members.
```

```
* @param member The member object to be added.
```

```
*/
```

```
public void join (Membership member)
```

```
Complete the join method.
```

```
(...)
```

# 4.14 Iterator



4) Create and complete a print method that belongs to the Club class, where you have to print all the information of the members of the club. USE an ITERATOR.

**public void print( )**



# 4.14 Iterator



5) Create a new class name **ClubDemo**, with a main method where you have to create a club and insert new members. As well as try the different methods that belong to the **Club** class.

Clue: When you wish to add a new **Membership** object to the **Club** object from the object bench, there are two ways you can do this. Either create a new **Membership** object on the object bench, call the **join** method on the **Club** object, and click on the **Membership** object to supply the parameter or call the **join** method on the **Club** object and type into the method's parameter dialog box:  
**new Membership ("member name ...", month, year)**

Each time you add one, use the **numberOfMembers** method to check both that the **join** method is adding to the collection and that the **numberOfMembers** method is giving the correct result.



# 4.15 Maps

20

- Maps are an unordered collection of key-value pairs.
- Maps cannot contain multiple pairs with the same key, but values can be duplicated.
- Maps have a `get(Object key)` method that returns the value to which that key is mapped, or null if the key is not mapped to anything.

# 4.15 Maps

21

## ➤ **HashMap:**

- ▣ It is a particular implementation of Map.
- ▣ The most important methods of the HashMap class are *put* and *get*.
- ▣ Example:

```
public static void main(String[] args) {  
  
    HashMap<String, String> phoneBook = new HashMap<String, String>();  
  
    phoneBook.put("Charles Nguyen", "(531) 9392 4587");  
    phoneBook.put("Lisa Jones", "(402) 4536 4674");  
    phoneBook.put("William H. Smith", "(998) 5488 0123");  
  
    System.out.println(phoneBook.get("Charles Nguyen"));  
  
}
```



# 4.15 Maps

22

## ■ Some of the methods:

1. **void clear():** It removes all the key and value pairs from the specified Map.
2. **Object clone():** It returns a copy of all the mappings of a map and used for cloning them into another map.
3. **boolean containsKey(Object key):** It is a boolean function which returns true or false based on whether the specified key is found in the map.
4. **boolean containsValue(Object Value):** Similar to containsKey() method, however it looks for the specified value instead of key.
5. **Value get(Object key):** It returns the value for the specified key.
6. **boolean isEmpty():** It checks whether the map is empty. If there are no key-value mapping present in the map then this function returns true else false.
7. **Set keySet():** It returns the Set of the keys fetched from the map.
8. **value put(Key k, Value v):** Inserts key value mapping into the map. Used in the above example.
9. **int size():** Returns the size of the map – Number of key-value mappings.
10. **Collection values():** It returns a collection of values of map.
11. **Value remove(Object key):** It removes the key-value pair for the specified key. Used in the above example.
12. **void putAll(Map m):** Copies all the elements of a map to the another specified map.

# 4.15 Maps

23

A.4.1 1. Create a class PhoneBook and use a **HashMap** to implement a phone book similar to the one in the example above (slide 21).

- 1) How do you check how many entries are contained in a map?
- 2) In this class, implement two methods:
  - `public void enterNumber(String name, String number)`
  - `public String lookupNumber(String name)`

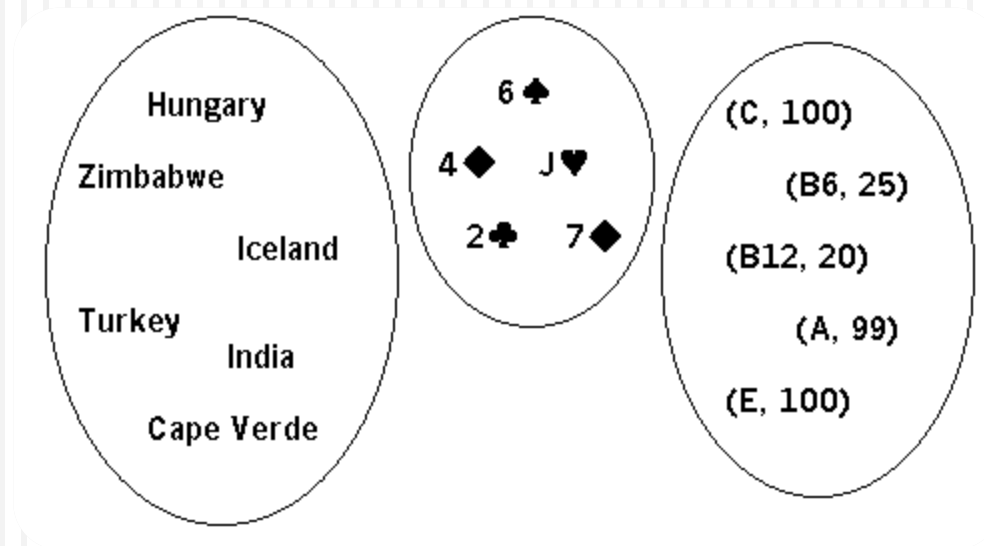
The methods should use the put and get methods of the HashMap class to implement their functionality.
- 3) What happens when you add an entry to a map with a key that already exists in the map?
- 4) How do you check whether a given key is contained in a map? (Give a Java code example.)
- 5) What happens when you try to look up a value and the key does not exist in the map?
- 6) How do you print out all keys currently stored in a map?



# 4.16 Sets

24

- Sets are an unordered collection of objects.
- Duplicates are not allowed.
- Examples:





# 4.16 Sets

25

## ➤ HashSet:

- ▣ The Java standard library includes different variations of sets implemented in different classes.
- ▣ The class we shall use is called HashSet.
- ▣ Example:

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    HashSet<String> mySet = new HashSet<String>();  
    mySet.add("one");  
    mySet.add("two");  
    mySet.add("three");  
  
    for(String item : mySet) {  
        System.out.println(item.toString());  
    }  
}
```



# 4.16 Sets

26

## ■ Some of the methods:

|   |   |
|---|---|
| 1 | <b>boolean add(Object o)</b><br>Adds the specified element to this set if it is not already present.              |
| 2 | <b>void clear()</b><br>Removes all of the elements from this set.   |
| 3 | <b>Object clone()</b><br>Returns a shallow copy of this HashSet instance: the elements themselves are not cloned. |
| 4 | <b>boolean contains(Object o)</b><br>Returns true if this set contains the specified element                      |
| 5 | <b>boolean isEmpty()</b><br>Returns true if this set contains no elements.  |
| 6 | <b>Iterator iterator()</b><br>Returns an iterator over the elements in this set.                                  |
| 7 | <b>boolean remove(Object o)</b><br>Removes the specified element from this set if it is present.                  |
| 8 | <b>int size()</b><br>Returns the number of elements in this set (its cardinality).                                |