International study centre

AN INTRODUCTION TO OBJECT-ORIENTATION AND THE JAVA PROGRAMMING LANGUAGE

JAVA COLLECTIONS AND GENERICS INTRODUCTION

- Name : John Alamina
- Email : john.alamina@hud.ac.uk

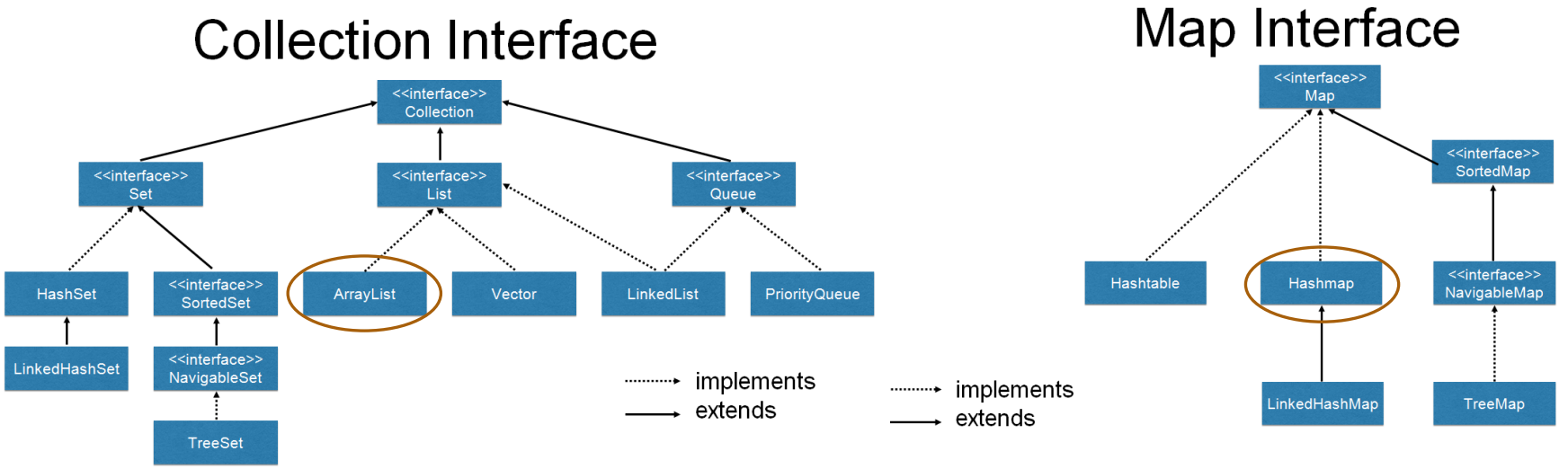
Outline

- Introduction to Collections
- Lists, Sets and Maps
- The collection Interface
- Java Generics
- The Address book Interface
- Address book Array method
- Address book List Method
- The Map interface
- Address book Map method
- Exercises

Introduction to Java collections

- The Java collections framework add additional features to basic arrays
- There are many classes in the collections framework but they are all derived from either the Collection interface or Map interface.
- We will focus on the ArrayList and HashMap
- Let's look at the Collection interface contract
  - Remember an interface is a contract that promises all sub-classes will implement all the methods declared in the interface.

Introduction to Java collections



https://dzone.com/articles/an-introduction-to-the-java-collections-framework

Lists Sets and Maps

- The Java collections framework add additional features to basic arrays
- There are many classes in the collections framework but they are all derived from either the Collection interface or Map interface.
- We will focus on the ArrayList and HashMap
- Collections come with loop objects called iterators that give access to each item in the collection.
- Let’s look at the Collection interface contract
  - Remember an interface is a contract that promises all sub-classes will implement all the methods declared in the interface.

The Collection interface

1. `coll.size()` — returns an int that gives the number of objects in the collection.
2. `coll.isEmpty()` — returns a boolean value which is `true` if the size of the collection is 0.
3. `coll.clear()` — removes all objects from the collection.
4. `coll.add(tobject)` — adds `tobject` to the collection.
5. `coll.contains(object)` — returns a boolean value that is true if `object` is in the collection.
6. `coll.remove(object)` — removes `object` from the collection.
7. `coll.containsAll(coll2)` — returns a boolean value that is true if every `object` in `coll2` is also in `coll`
8. `coll.addAll(coll2)` — adds all the objects in `coll2` to `coll`. The parameter, `coll2`, can be any collection of type *Collection<T>*
9. `coll.removeAll(coll2)` — removes every `object` from `coll` that also occurs in the collection `coll2`. `coll2` can be any collection.
10. `coll.retainAll(coll2)` — removes every `object` from `coll` that **does not occur** in the collection `coll2`.
11. `coll.toArray()` — returns an array of type `Object[]` that contains all the items in the collection.

Generics introduction

- The Java collections framework add additional features to basic arrays such as the ability to use collections without direct reference to it's index. For example adding elements to a collection.
- Generics allows a uniform way to manipulate collections of varying data types. Consider the snippet below

```
List list = new ArrayList();

list.add("hello");

String s = (String) list.get(0); // without generic code
```

- When re-written to use generics, the code does not require casting:

```
List<String> list = new ArrayList<String>();

list.add("hello");

String s = list.get(0);   // no cast
```

Iterators

- Iterator objects are accessed from the listIterator() method of collection classes
- The code below shows how to use iterators. However, these are hardly used as the simpler "for"-(each)-item-in-syntax also shown below is used as the alternative

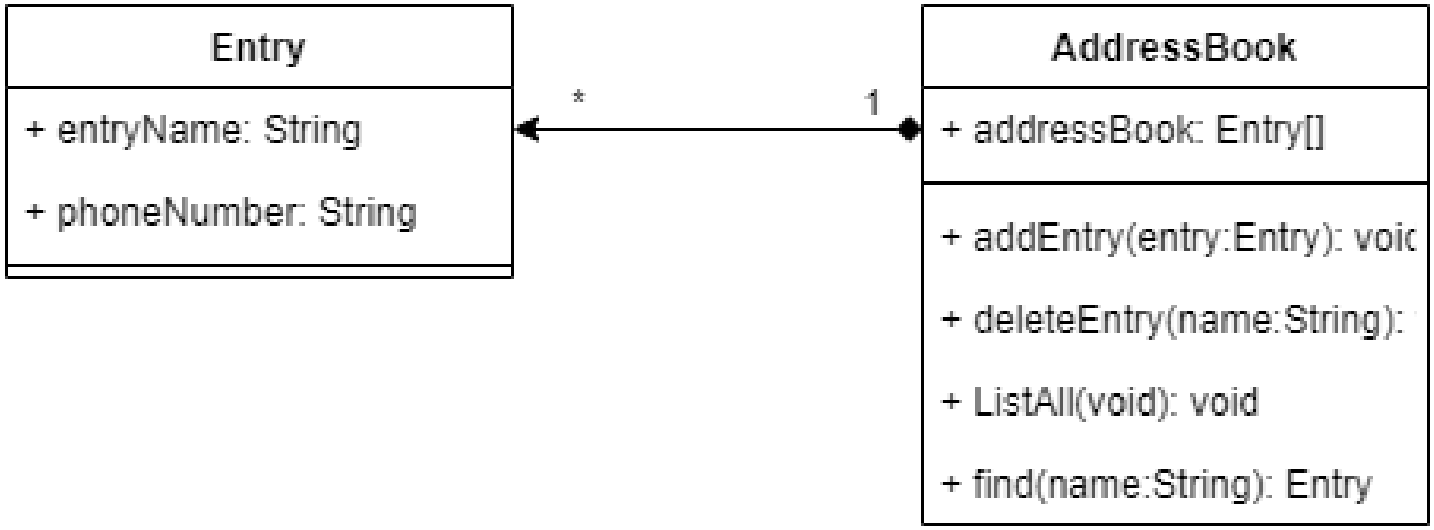
```
//using the iterator syntax
ListIterator<String> iter = stringList.listIterator();
while (iter.hasNext()) {
   String item = iter.next();
   if (newItem.compareTo(item) <= 0) {
      iter.previous();
      break;
   }
}
iter.add(newItem);

//alternative syntax
for (String item:stringList) {
    System.out.println(item);//list all items to console
}
```

Any Questions?

The Address book Interface

- For an address book we would like to maintain a list of addresses that will have the following attributes and behaviours.
- Attributes (properties)
  1. Entry list (type=class {Entry})
- Methods (behaviours)
  1. Add Entry – add an address book entry
  2. Remove entry – remove and address book entry
  3. List Entries – shows all entries
  4. Find entry – find and entry by name
- Entry Class properties
  - Name (type=string, public)
  - Phone Number (type=string,public)

Address book – class diagram



Address book – Array method

```
public class AddressBookArray {
private Entry[] addressBook= new Entry[3];

public void ListAll(){
    for(int i=0;i<addressBook.length;i++){
        if(addressBook[i]!=null){
            System.out.print(" Name: "+addressBook[i
].entryName);
            System.out.println(" Phone: "+addressBoo
k[i].phoneNumber);
        }
    }
}
private void addEntry(Entry entry){
    for(int i=0;i<addressBook.length;i++){
        if(addressBook[i]==null){
            addressBook[i]=new Entry();
            addressBook[i].entryName=entry.entryName
;
            addressBook[i].phoneNumber=entry.phoneNu
mber;
            System.out.println(" Entry successful!")
;
            return;
        }
    }
    System.out.println(" No room found for entry!");
}
```

```
private Entry findEntry(String name){
    for(int i=0;i<addressBook.length;i++){
        if(addressBook[i]!=null && addressBook[i].en
tryName.equals(name)){
            System.out.println(" Found! ");
            System.out.print(" Name: "+addressBook[i
].entryName);
            System.out.println(" Phone: "+addressBoo
k[i].phoneNumber);
            return addressBook[i];
        }
    }
    System.out.println(" Not found! ";
    return null;
}
private void deleteEntry(String name){
    for(int i=0;i<addressBook.length;i++){
        if(addressBook[i]!=null && addressBook[i].en
tryName.equals(name)){
            System.out.println(" Found and deleted!
");
            addressBook[i]=null;
           return;
        }
    }
    System.out.println(" Not found! ";
    return ;
}
}
```

Address book – List method

```
public class AddressBookList {
private List addressBook= new A
rrayList();

public void ListAll(){
    for(Entry entry:addressBook
){
            System.out.print("
Name: "+ entry.entryName);
            System.out.println(
" Phone: "+entry.phoneNumber);
        }
    }
}
private void addEntry(Entry ent
ry){
    addressBook.add(entry);
    System.out.println(" Entry
successful!");
}
```

```
private Entry findEntry(String name){
  for(Entry entry:addressBook){
    if(entry.entryName.equals(name)){
            System.out.println(" Found! ");
            System.out.print(" Name: "+entr
y.entryName);
            System.out.println(" Phone: "+e
ntry.phoneNumber);
            return entry;
        };
    }
    System.out.println(" Not found! ";
    return null;
private void deleteEntry(String name){
    Entry entry=findEntry(name);
    if(entry!=null)addressBook.remove(entry
);
    System.out.println(" Deleted! ");
    return null;
}

}
```

Any Questions?

The map interface

- `map.get(key)` -- returns the object of type *V* that is associated by the map to the `key`
- `map.put(key,value)` -- Associates the specified `value` with the specified `key`, where `key` must be of type *K* and `value` must be of type *V*.
- `map.putAll(map2)` -- if `map2` is another map of type *Map<K,V>*, this copies all the associations from `map2` into map.
- `map.remove(key)` -- if `map` associates a value to the specified `key`, that association is removed from the map.
- `map.containsKey(key)` -- returns a boolean value that is `true` if the map associates some value to the specified `key`.
- `map.containsValue(value)` -- returns a boolean value that is `true` if the map associates the specified `value` to some key.
- `map.size()` -- returns an int that gives the number of key/value associations in the map.
- `map.isEmpty()` -- returns a boolean value that is `true` if the map is empty, that is if it contains no associations.
- `map.clear()` -- removes all associations from the map, leaving it empty.

Address book – Map method

```
public class AddressBookMap {
private HashMap<String,String> addressBook=
 new HashMap<String,String>();

public void ListAll(){
  Set<String> keys =  addressBook.keySet();
     // The set of keys in the map.
  Iterator<String> keyIter = keys.iterator(
);
  while (keyIter.hasNext()) {
     String key = keyIter.next();  // Get t
he next key.
     String value =  addressBook.get(key);
 // Get the value for that key.
     System.out.println( " Name:\t" + key +
 "PhoneNumber" + value + "" );
  }
}
private void addEntry(Entry entry){
  addressBook.put(entry.entryName,entry.pho
neNumber);
  System.out.println(" Entry successful!");
}
```

```
private Entry findEntry(String name){
  String entry=addressBook.get(name);
  if(entry!=null){
    System.out.println(" Found! ");
    System.out.print(" Name:\t"+name);
    System.out.println("Phone: "+entry);
    return new Entry(name,entry);
  }
  System.out.println(" Not found! ";
  return null;
}
private void deleteEntry(String name){
    addressBook.remove(entry);
    System.out.println(" Deleted! ");
    return;
}

}
```

Summary of collections

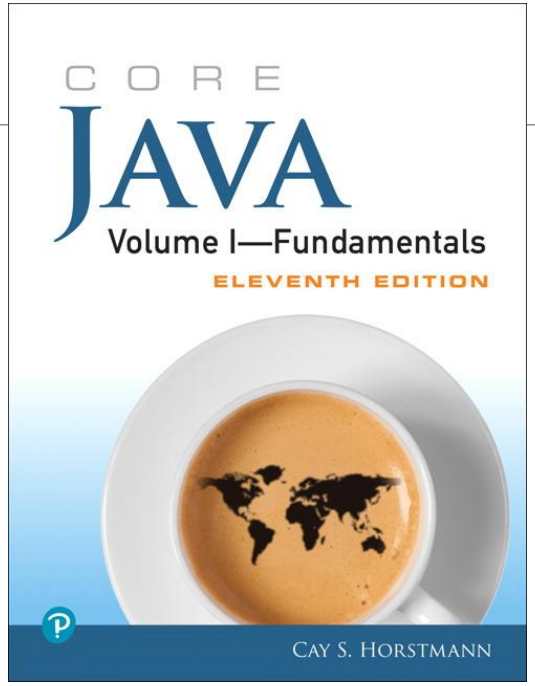
- In a list we are dealing with one type i.e. List<T>. In a map however, we are dealing with two types Map<K,V>, a key and a value.
- HashMap does not allow duplicate values the way a list would allow while adding items to the list.
- The collections we have seen allow dynamic addition of items which is superior to static allocation done in arrays.
- Arrays have fixed length while collections have dynamic lengths.
- Collections provide methods that allow dealing with lists and maps a lot more quicker and convenient than ordinary arrays

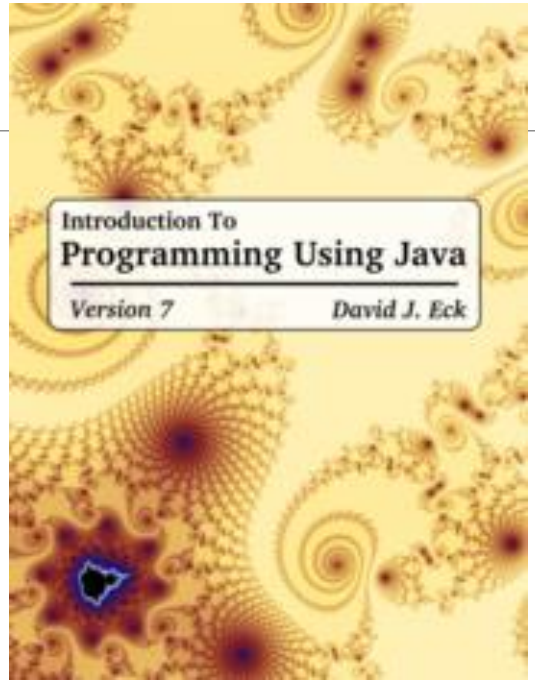
Any Questions?

Exercises

1. Implement the Address book using arrays
2. Implement an address book using Lists
3. Implement address book using Map
4. (advanced) Implement a private sort method called by the List entries method of the address book that will display entries in alphabetical order

Supplementary material





- The Java Tutorial
- Java API documentation
- Link to today's Session screencast
- Link to John's Group Padlet
- Link to Kelly's Group Padlet