Chapter 5
Java Collection

Wang Yang
wyang@njnet.edu.cn

Outline

- Last Chapter Review
- Array & Arrays Tourism
- Collection & Collections
- Map
- Performance benchmark

Last Chapter Review

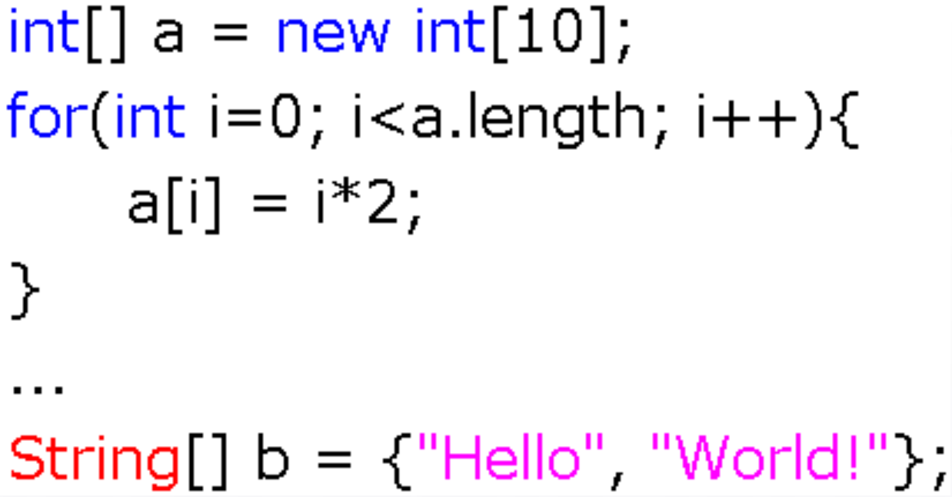
Exception

- Exception is an Object
- try/catch/finally
- throw/throws
- exception/error/normal problem
- catched exception/runtime exception

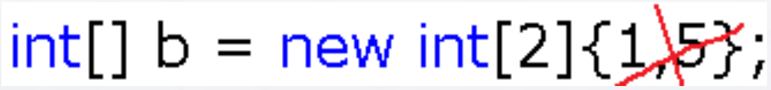
Array & Arrays Tourism

Recall of Array

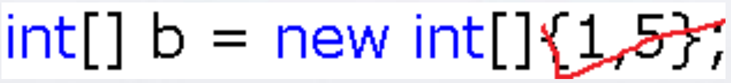
- Declaration and Assignment of an Array



```
int[] a = new int[10];
for(int i=0; i<a.length; i++){
    a[i] = i*2;
}
...
String[] b = {"Hello", "World!"};
```



```
int[] b = new int[2]{1,5};
```



```
int[] b = new int[]{1,5};
```

Use of Array

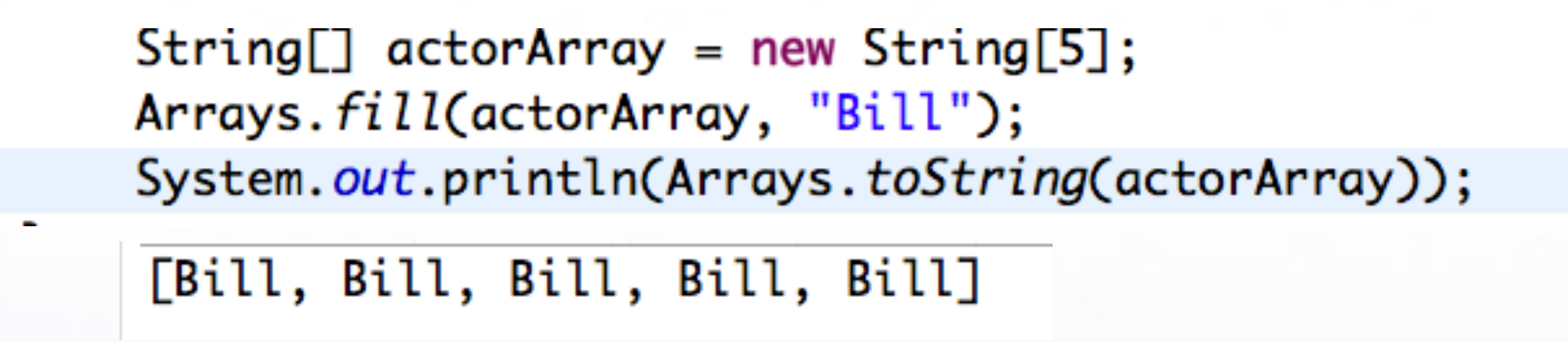
- Initialize
- Search & Sort
- toString / equal / hashcode

Use of Array

- Arrays
  - java.util.Arrays
  - all method are static
  - helper class

Initialize

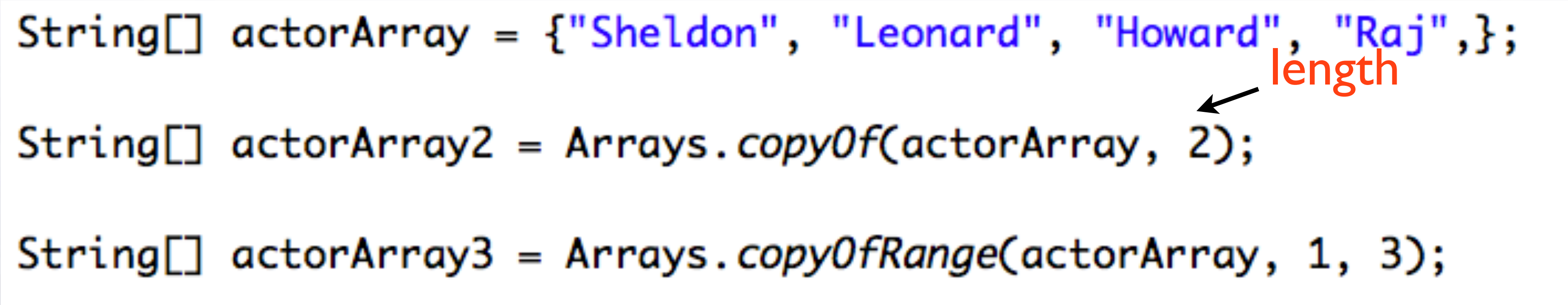
- fill



```
String[] actorArray = new String[5];
Arrays.fill(actorArray, "Bill");
System.out.println(Arrays.toString(actorArray));
```

```
[Bill, Bill, Bill, Bill, Bill]
```

- copyOf / copyOfRange

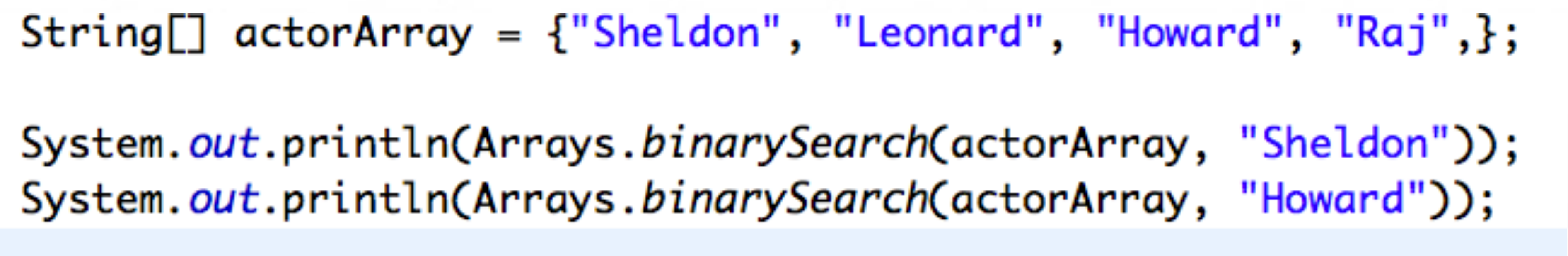


```
String[] actorArray = {"Sheldon", "Leonard", "Howard", "Raj",};

String[] actorArray2 = Arrays.copyOf(actorArray, 2);

String[] actorArray3 = Arrays.copyOfRange(actorArray, 1, 3);
```

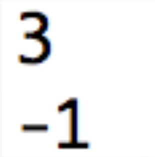
length

[startindex, endindex)

- binarySearch



```
String[] actorArray = {"Sheldon", "Leonard", "Howard", "Raj",};

System.out.println(Arrays.binarySearch(actorArray, "Sheldon"));
System.out.println(Arrays.binarySearch(actorArray, "Howard"));
```



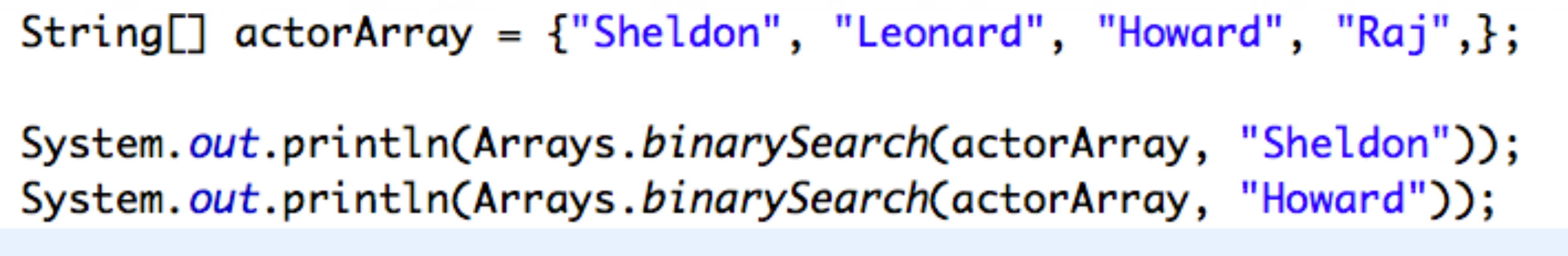
```
3
-1
```

Right

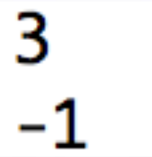
Wrong, Why ?

Search & Sort

- binarySearch



```
String[] actorArray = {"Sheldon", "Leonard", "Howard", "Raj",};

System.out.println(Arrays.binarySearch(actorArray, "Sheldon"));
System.out.println(Arrays.binarySearch(actorArray, "Howard"));
```



```
3
-1
```

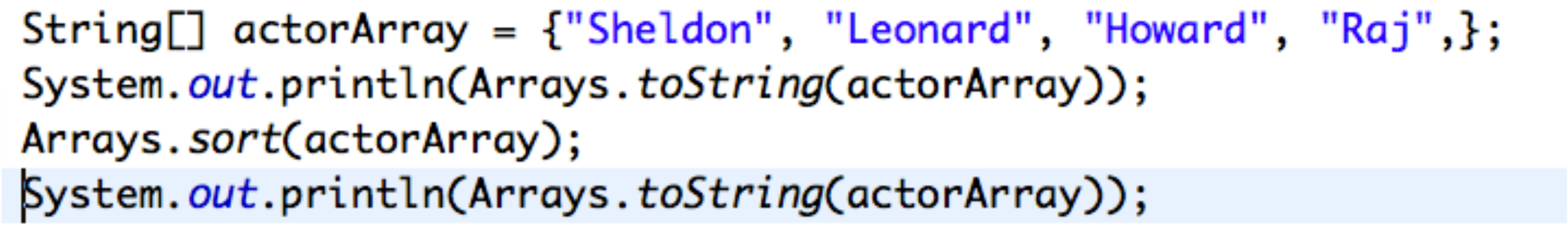
Right

Wrong, Why ?

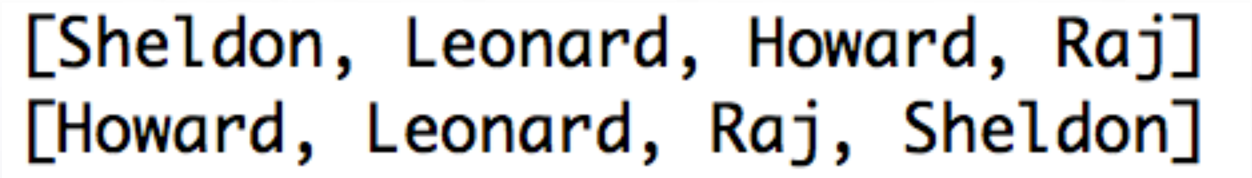
binarySearch only work for sorted array !!

Search & Sort

- Sort



```
String[] actorArray = {"Sheldon", "Leonard", "Howard", "Raj",};
System.out.println(Arrays.toString(actorArray));
Arrays.sort(actorArray);
System.out.println(Arrays.toString(actorArray));
```



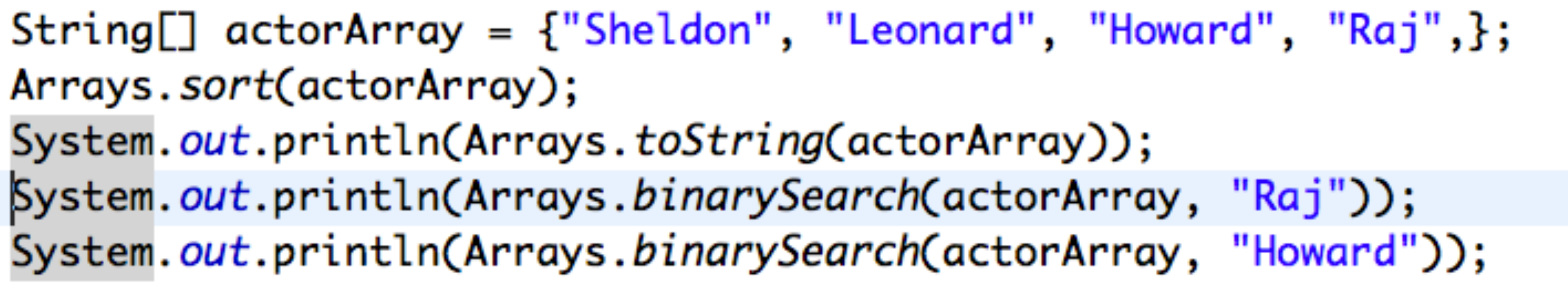
```
[Sheldon, Leonard, Howard, Raj]
[Howard, Leonard, Raj, Sheldon]
```

Unsorted

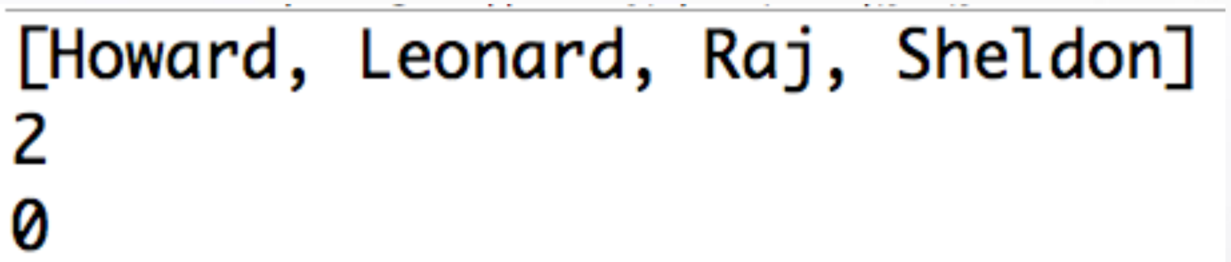
Sorted

Search & Sort

- Then Search



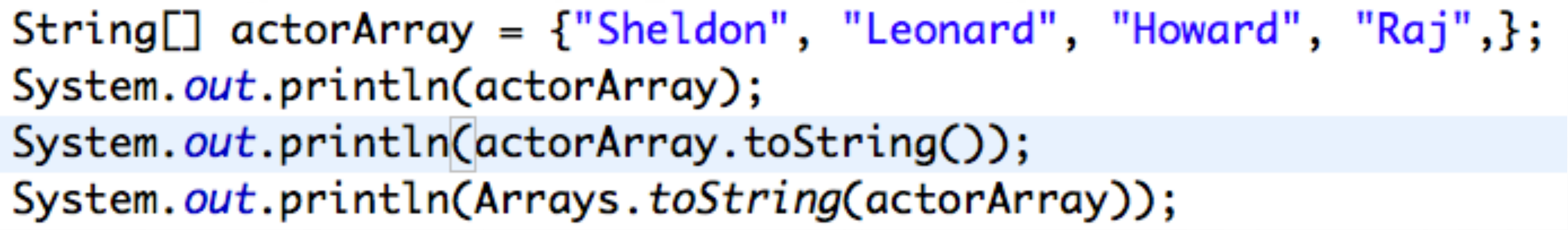
```
String[] actorArray = {"Sheldon", "Leonard", "Howard", "Raj",};
Arrays.sort(actorArray);
System.out.println(Arrays.toString(actorArray));
System.out.println(Arrays.binarySearch(actorArray, "Raj"));
System.out.println(Arrays.binarySearch(actorArray, "Howard"));
```



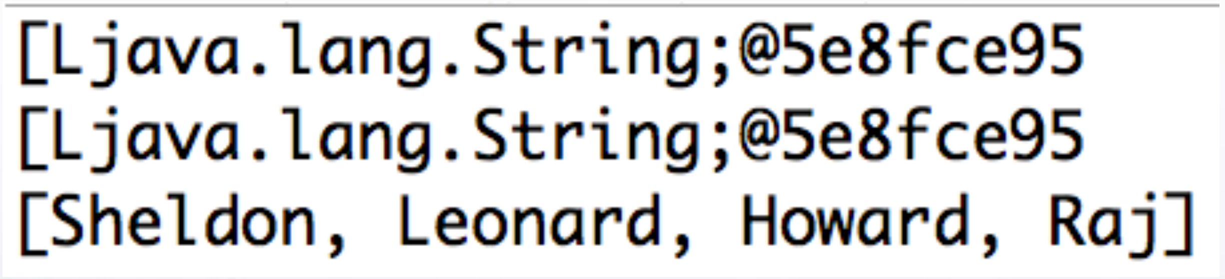
```
[Howard, Leonard, Raj, Sheldon]
2
0
```

toString

- not same toString



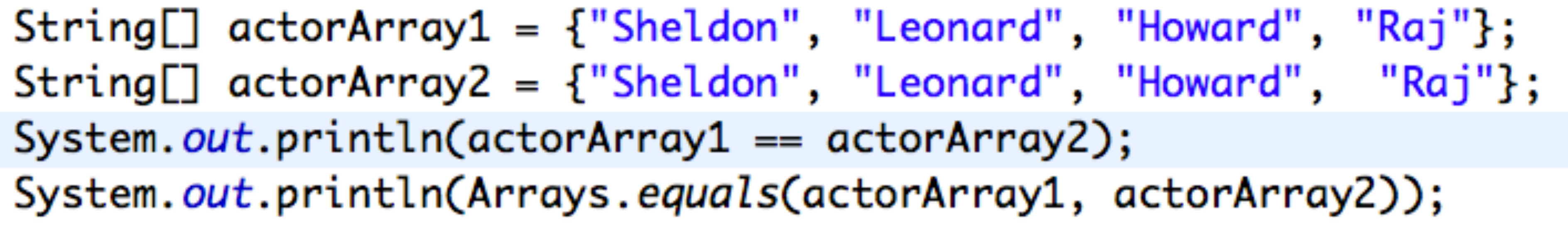
```java
String[] actorArray = {"Sheldon", "Leonard", "Howard", "Raj",};
System.out.println(actorArray);
System.out.println(actorArray.toString());
System.out.println(Arrays.toString(actorArray));
```



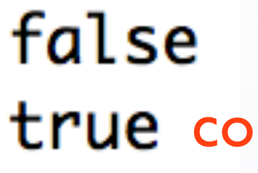
```
[Ljava.lang.String;@5e8fce95
[Ljava.lang.String;@5e8fce95
[Sheldon, Leonard, Howard, Raj]
```

Equal

- equal



```
String[] actorArray1 = {"Sheldon", "Leonard", "Howard", "Raj"};
String[] actorArray2 = {"Sheldon", "Leonard", "Howard",  "Raj"};
System.out.println(actorArray1 == actorArray2);
System.out.println(Arrays.equals(actorArray1, actorArray2));
```

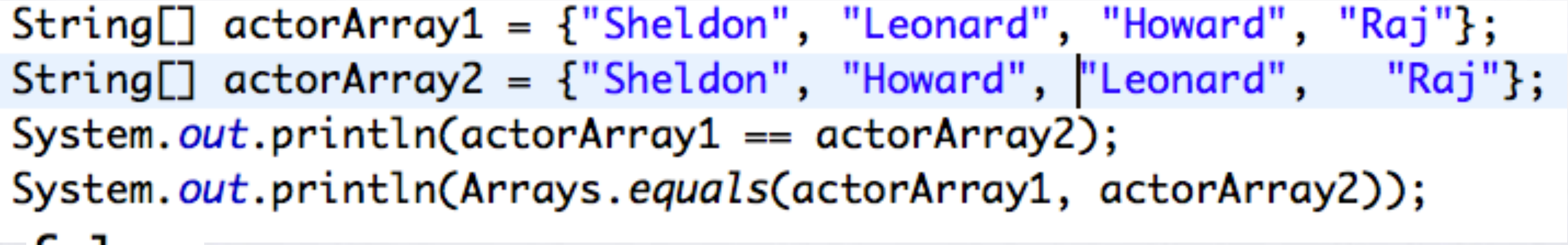


```
false
true
```

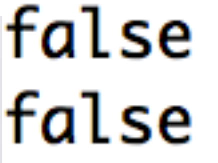
refer_1 != refer_2

content_1 == content_2

- the order of element will be considered



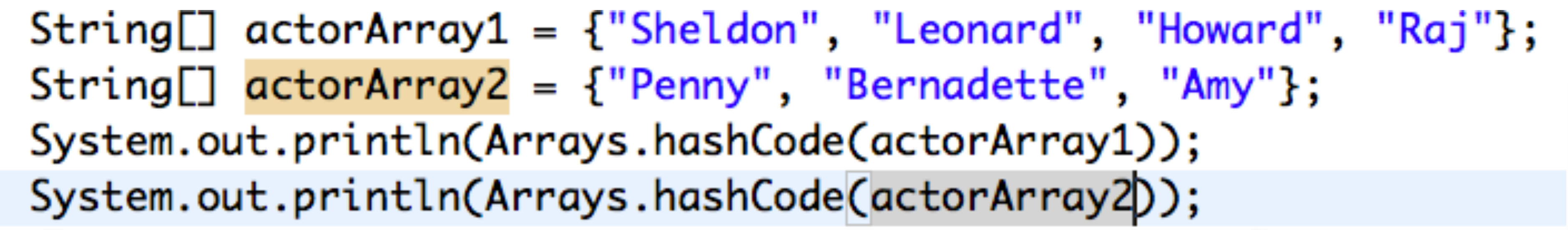
```
String[] actorArray1 = {"Sheldon", "Leonard", "Howard", "Raj"};
String[] actorArray2 = {"Sheldon", "Howard", "Leonard",    "Raj"};
System.out.println(actorArray1 == actorArray2);
System.out.println(Arrays.equals(actorArray1, actorArray2));
```



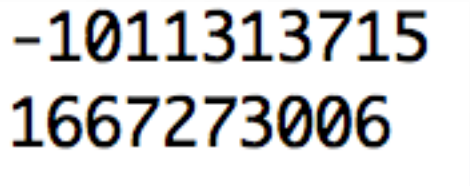
```
false
false
```

Hashcode

- hashCode



```
String[] actorArray1 = {"Sheldon", "Leonard", "Howard", "Raj"};
String[] actorArray2 = {"Penny", "Bernadette", "Amy"};
System.out.println(Arrays.hashCode(actorArray1));
System.out.println(Arrays.hashCode(actorArray2));
```

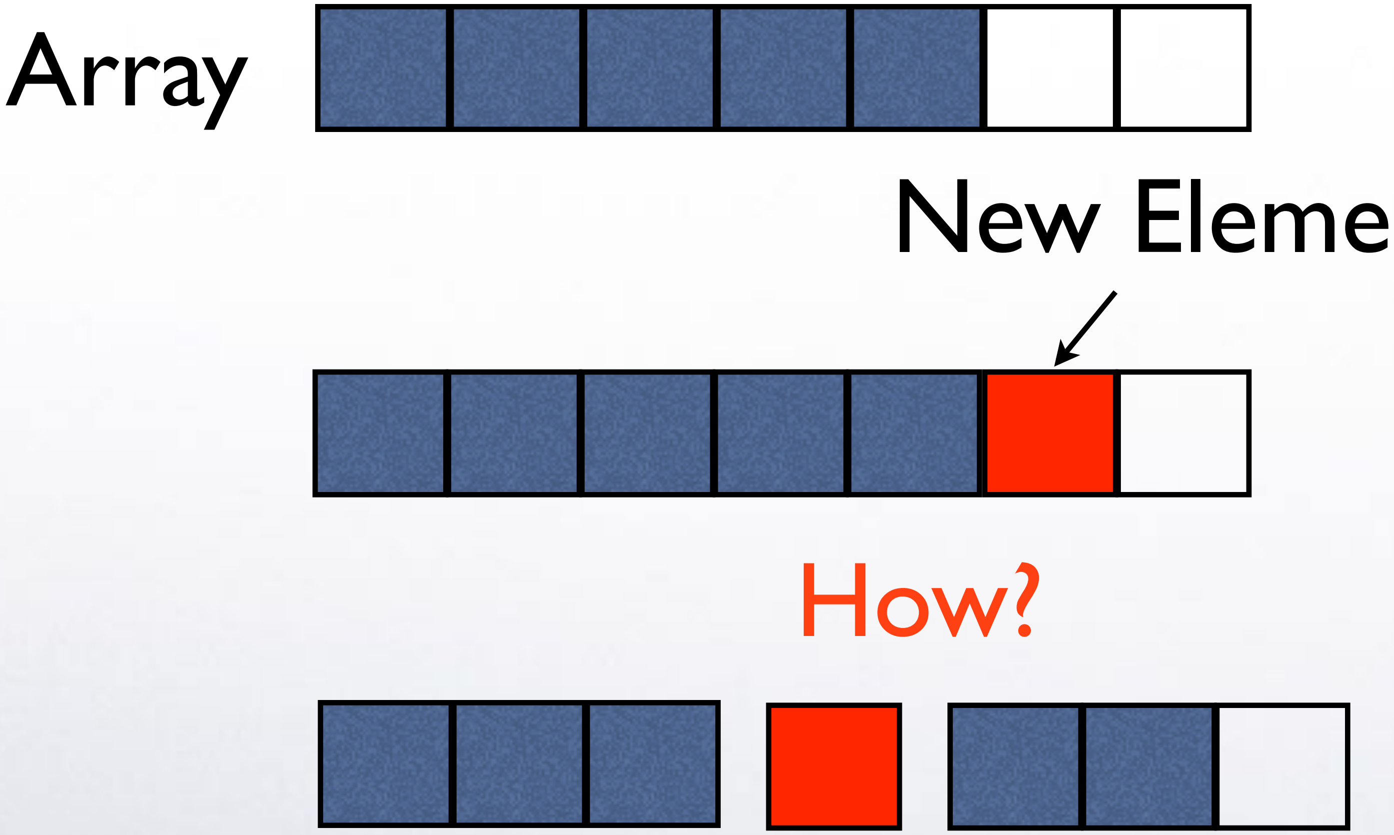


```
-1011313715
1667273006
```

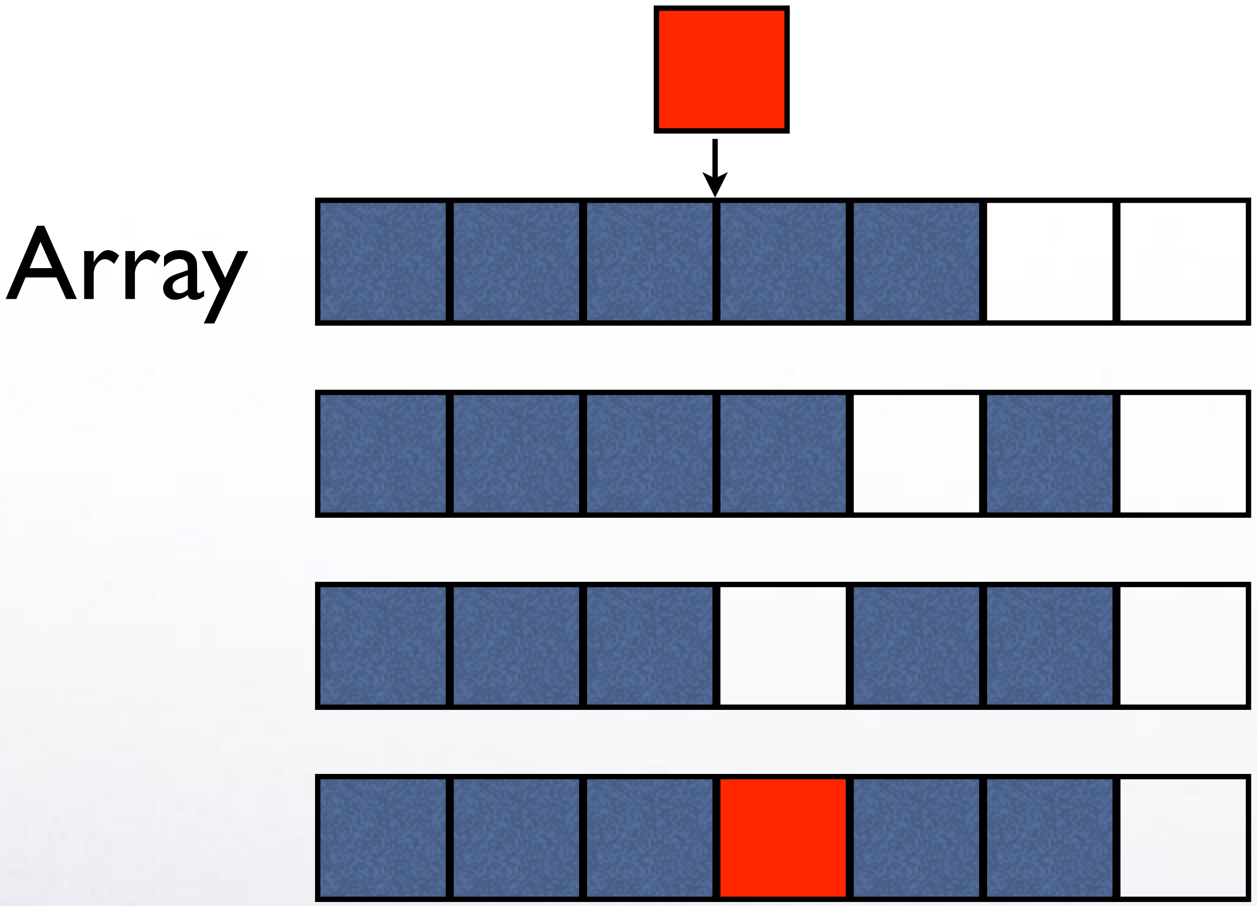
Why not just Array

- Shortage of Array
  - Fixed-length
    - Space penalty
  - Complex for insert and delete
    - not All the time
    - Time penalty

Add Element in Array



Add Element in Array

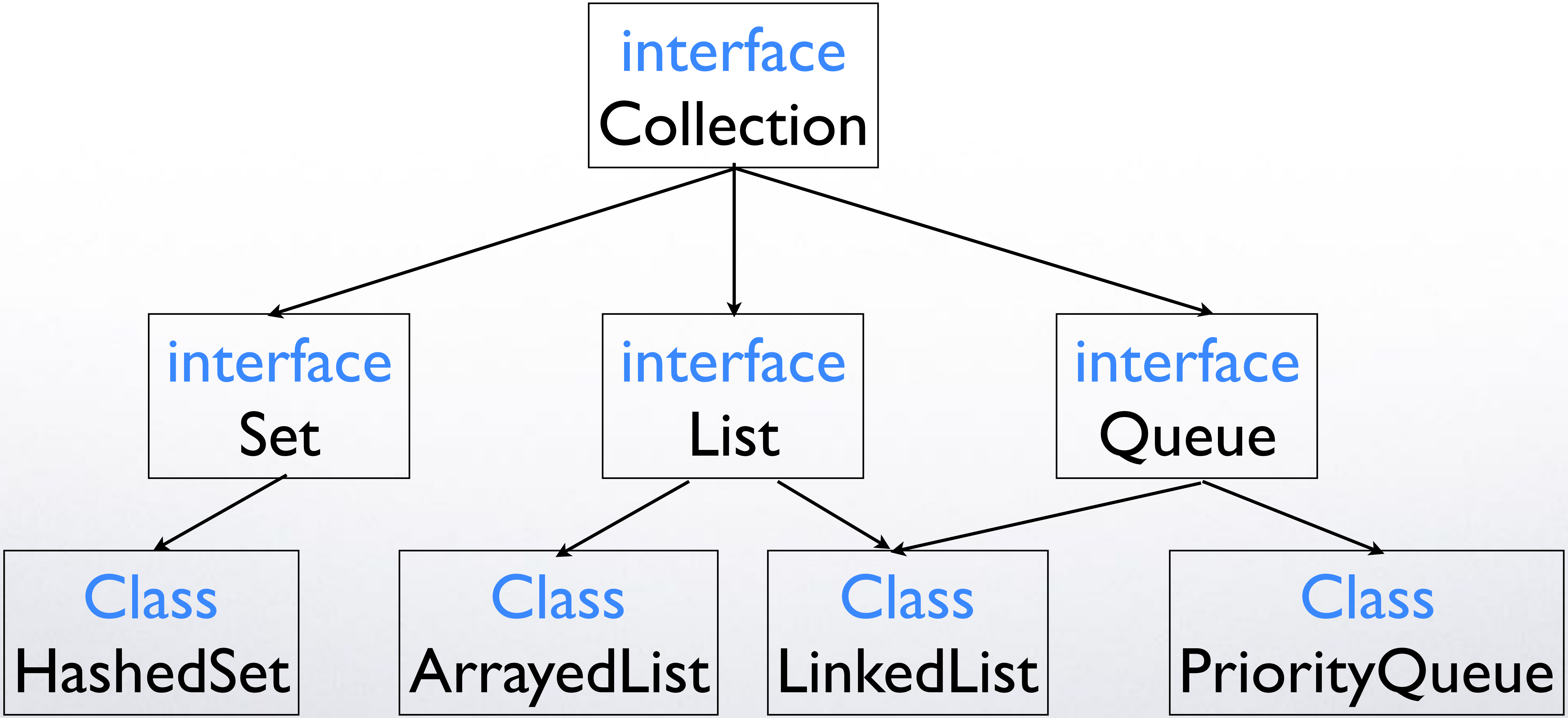


Collection

Collection

- Variable Length
- More ways to visit values
- Collection in java.util

Java Collection Framework



interface Collection

interface Set

interface List

interface Queue

Class HashedSet

Class ArrayedList

Class LinkedList

Class PriorityQueue

Collection

- Collection<E>
- root interface
- a group of objects -- elements
- allow duplicated or not dulp
- allow ordered or unordered
- add/remove/clear/contain/size
- toArray

List

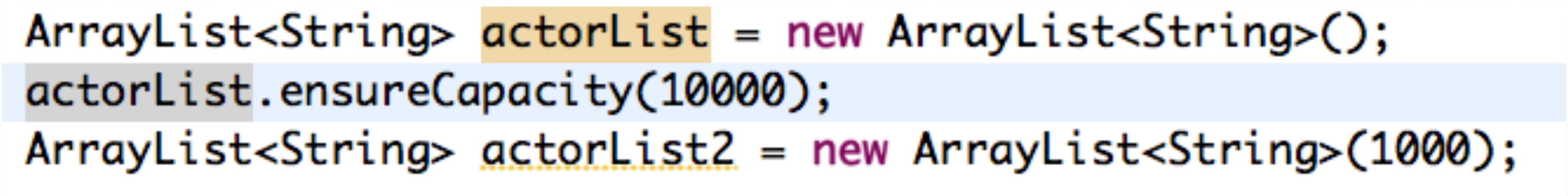
- List<E>
- An ordered collection (also known as a sequence).
- precise control over where in the list each element is inserted.
- access elements by their integer index

ArrayList

- the under is Array...
- have capacity
  - when the new element make size > capacity, the under array will extends itself

ArrayList

- Create an ArrayList

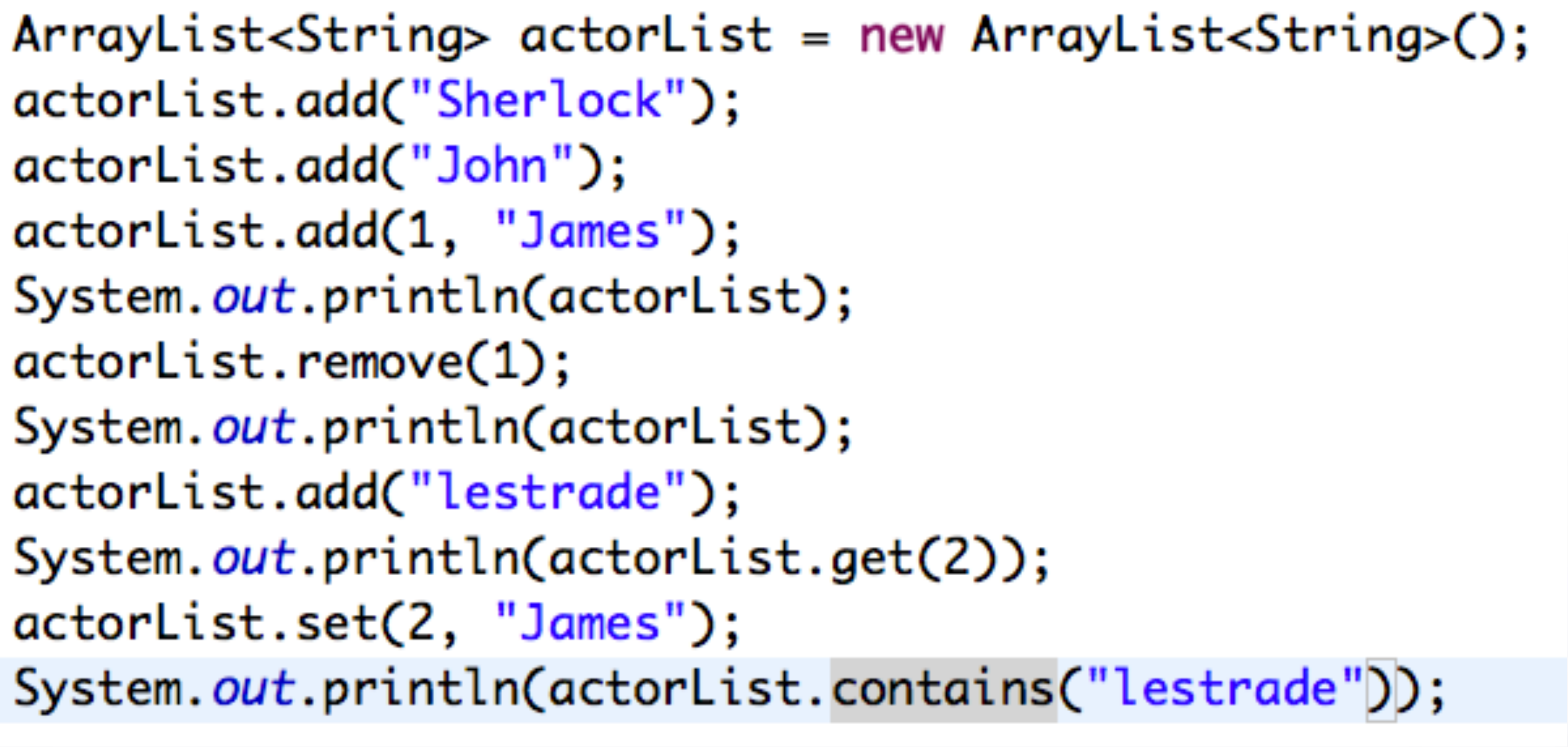


```java
ArrayList<String> actorList = new ArrayList<String>();
actorList.ensureCapacity(10000);
ArrayList<String> actorList2 = new ArrayList<String>(1000);
```

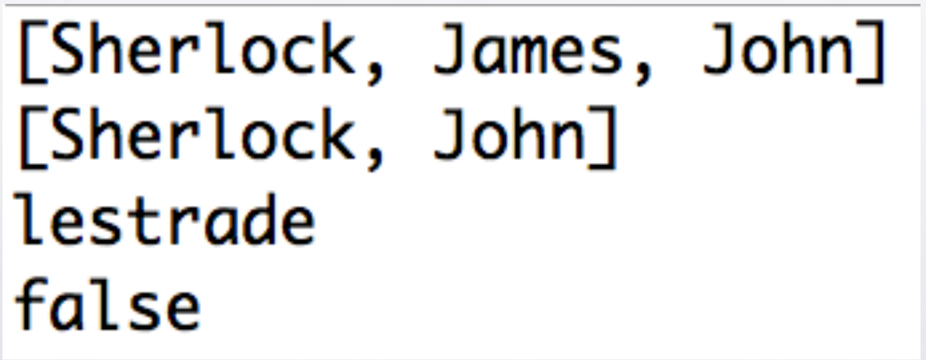
ArrayList

- Operator
  - add(E) : add the E at the end of list
  - add(index, E) : add the E at the index of list
  - remove(E) : remove the last E
  - remove(index, E) : remote the E at the index pos
  - get(index) : get E from index pos
  - set(index) : change E's value at the index pos
  - contain(E) : if E in the list

ArrayList



```java
ArrayList<String> actorList = new ArrayList<String>();
actorList.add("Sherlock");
actorList.add("John");
actorList.add(1, "James");
System.out.println(actorList);
actorList.remove(1);
System.out.println(actorList);
actorList.add("lestrade");
System.out.println(actorList.get(2));
actorList.set(2, "James");
System.out.println(actorList.contains("lestrade"));
```



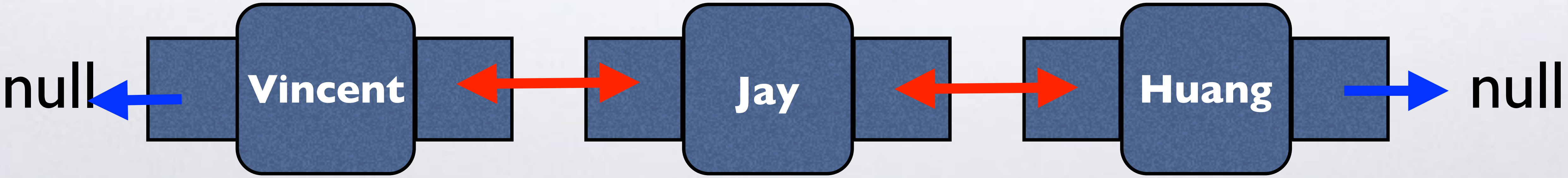
```
[Sherlock, James, John]
[Sherlock, John]
lestrade
false
```

ArrayList

- Feature
    - Efficient in random access of elements
    - May enlarge backend array when append new elements (can be partly solved by setting initial capacity)
    - Not efficient for insertion (may cause the movement of elements)
    - Waste of space (solved by trimToSize)

LinkedList

- Implemented by co-reference of neighbors
- No capacity
- Each Element stores:
  - A reference to the previous element
  - A reference to the succeeded element
  - The value



LinkedList

- Operator
  - addFirst/addLast
  - removeFirst/removeLast
  - peek/poll

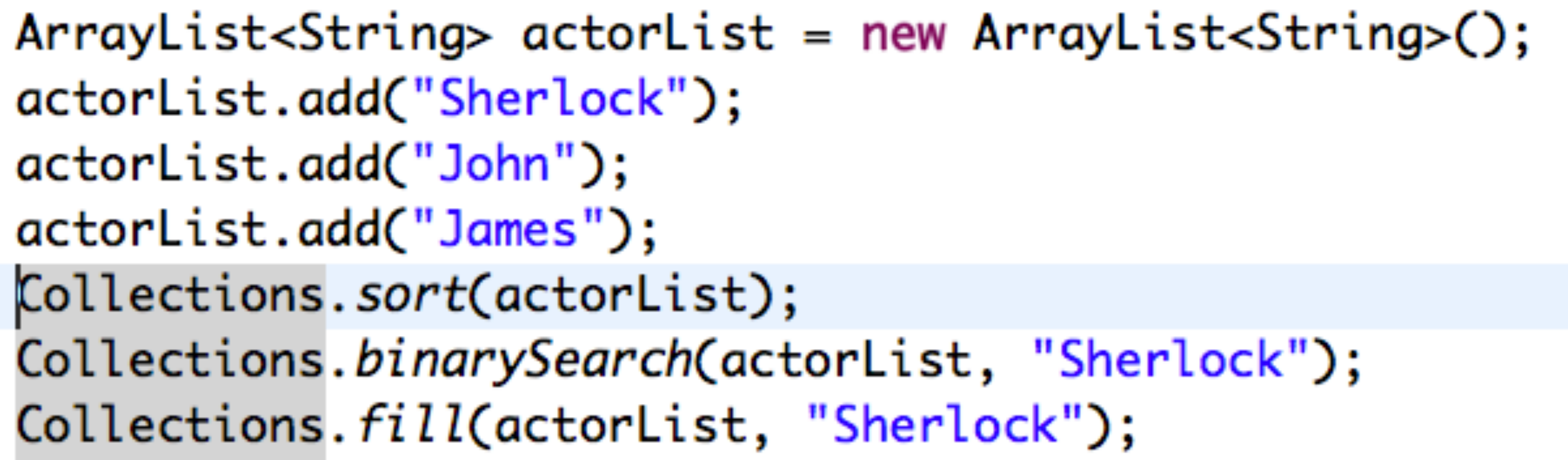
LinkedList

- Feature
  - Do not cause the reassignment of memory
  - Efficient for add / delete / insert
  - Not efficient for random access (need traverse from head)

Collections

- Collections
  - java.util. Collections
  - all method are static
  - helper class

Collections



```
ArrayList<String> actorList = new ArrayList<String>();
actorList.add("Sherlock");
actorList.add("John");
actorList.add("James");
Collections.sort(actorList);
Collections.binarySearch(actorList, "Sherlock");
Collections.fill(actorList, "Sherlock");
```

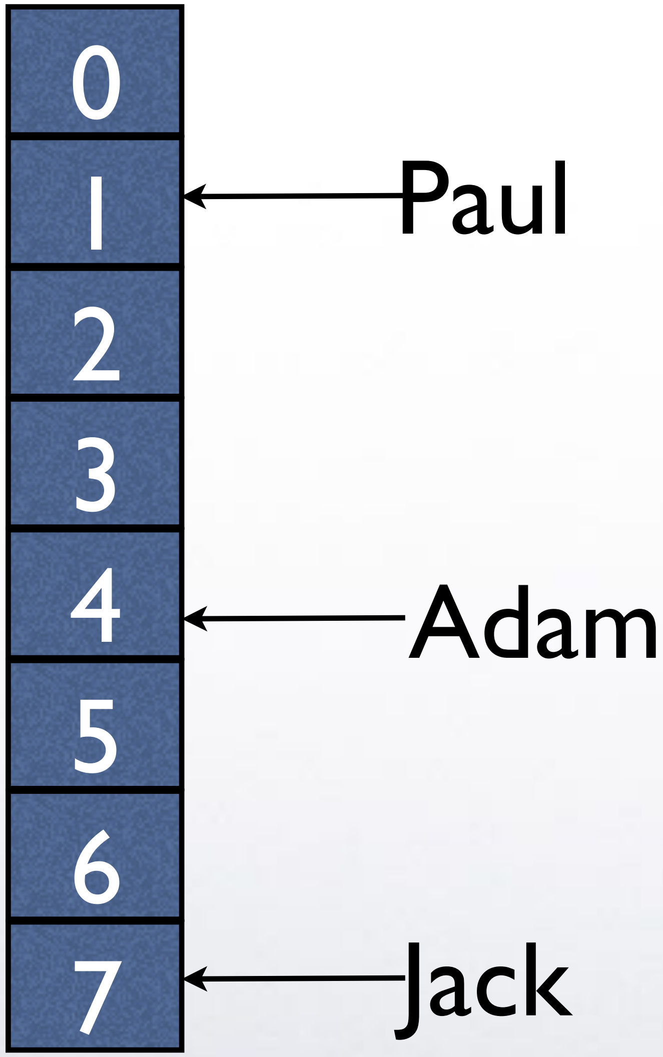
Map

- An object that maps keys to values
- Dictionary
- Key, Value

  Key（张三）:Value（19,男，计算机）

  - must be object
  - cannot contain duplicate keys
  - each key can map to at most one value
- HashMap, TreeMap

HashMap



O(1)

HashCode

- Object : hashCode()
- Integer
- Each Class can Define its own Hash Algorithm
- Requirement:
  - the same integer for the same object more than once during an execution of a Java application
  - If two objects are equal according to equals(Object) method, hashCode() return the same integer result
  - If two objects are not equal, not require hashCode() return the different integer result

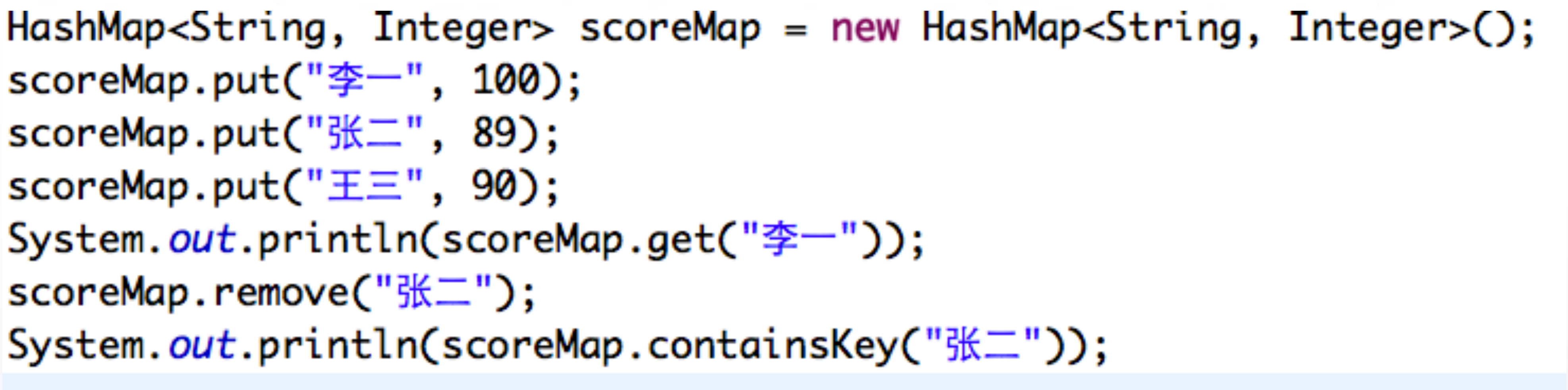
HashCode

- Object
  - converting the internal address of the object into an integer
- String
  - s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
- Integer
  - itself

Map

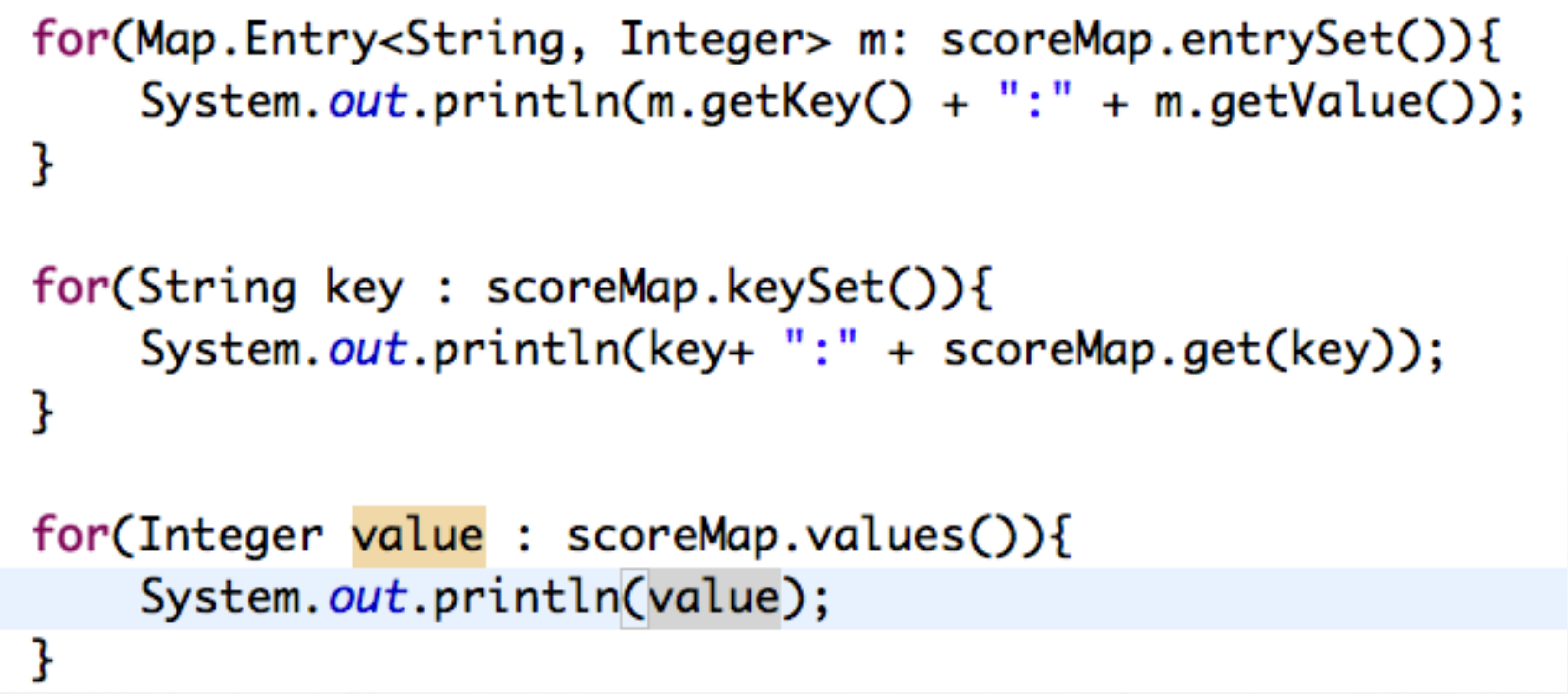
- Operator
  - add an k,v Pair : put(k, v)
  - get a v by a k : get(k)
  - remove a k : remove(k)
  - find if key exists : contain(key)
  - get all key : keySet()
  - get all values : values()
  - get all k,v pairs : entrySet()
    - Map.Entry<K,V>

HashMap



```java
HashMap<String, Integer> scoreMap = new HashMap<String, Integer>();
scoreMap.put("李一", 100);
scoreMap.put("张二", 89);
scoreMap.put("王三", 90);
System.out.println(scoreMap.get("李一"));
scoreMap.remove("张二");
System.out.println(scoreMap.containsKey("张二"));
```

HashMap

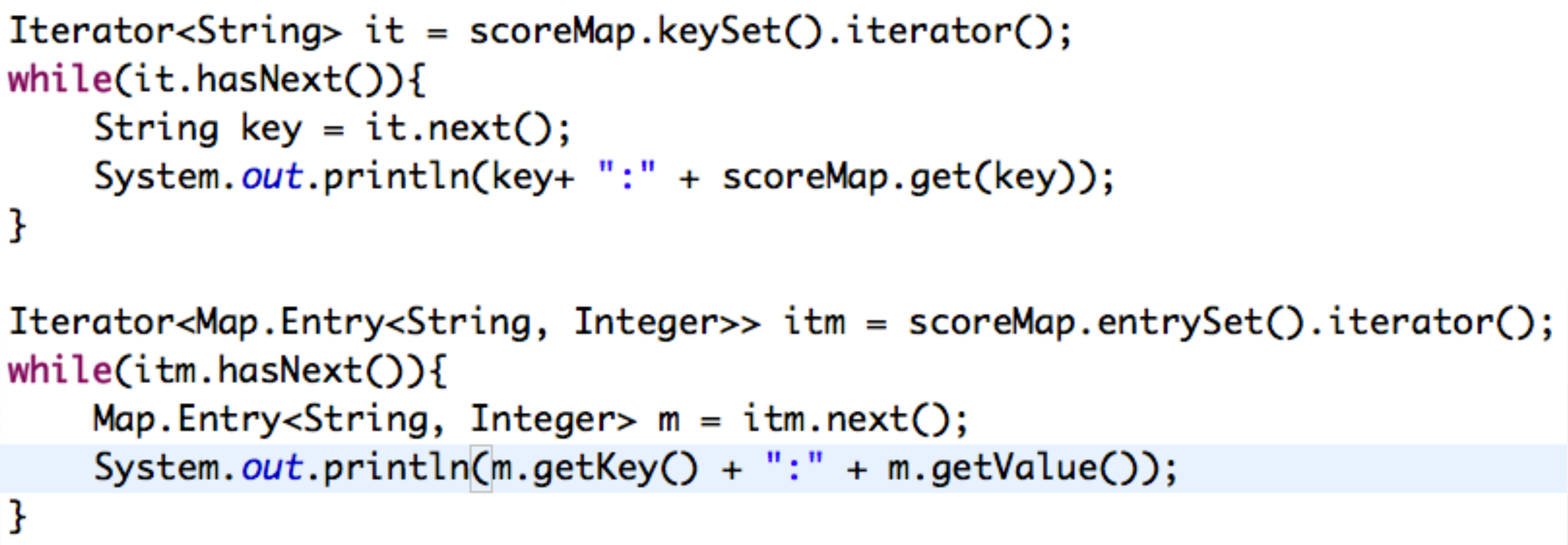


```
for(Map.Entry<String, Integer> m: scoreMap.entrySet()){
    System.out.println(m.getKey() + ":" + m.getValue());
}

for(String key : scoreMap.keySet()){
    System.out.println(key+ ":" + scoreMap.get(key));
}

for(Integer value : scoreMap.values()){
    System.out.println(value);
}
```

Iterator

- Iterator for the Traverse of Collection
- There is an iterator() Method in Collection
  - Each implemented class of Collection should implemented iterator()
  - Each implemented class of Collection can be traversed using iterator()
- Methods in Iterator:
  - hasNext()
  - next()

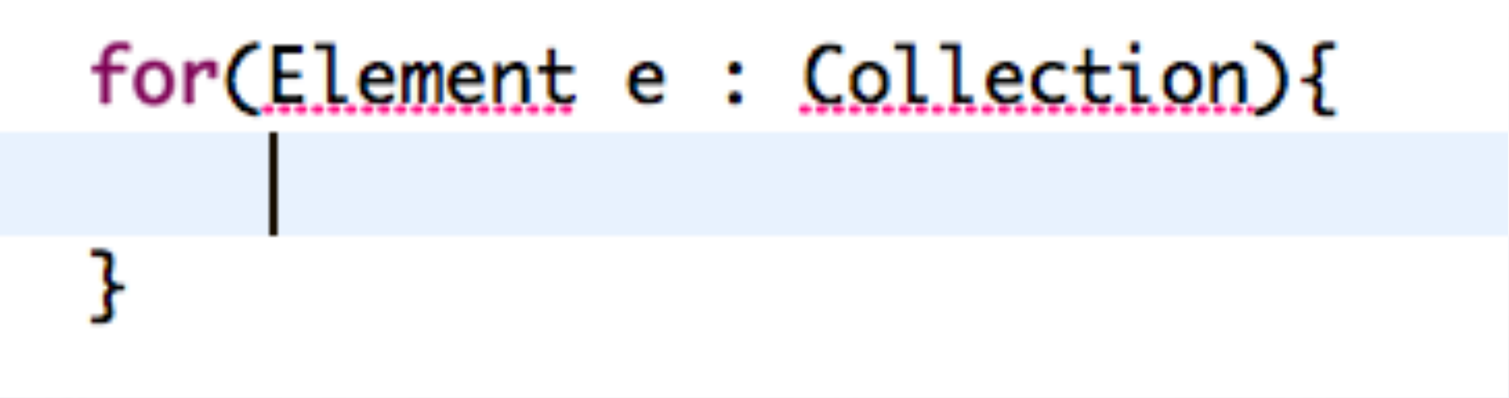
Iterator



```
Iterator<String> it = scoreMap.keySet().iterator();
while(it.hasNext()){
    String key = it.next();
    System.out.println(key+ ":" + scoreMap.get(key));
}

Iterator<Map.Entry<String, Integer>> itm = scoreMap.entrySet().iterator();
while(itm.hasNext()){
    Map.Entry<String, Integer> m = itm.next();
    System.out.println(m.getKey() + ":" + m.getValue());
}
```

For-each Loop

- For-each Loop
  - for each element in a collection



```
for(Element e : Collection){
    
}
```

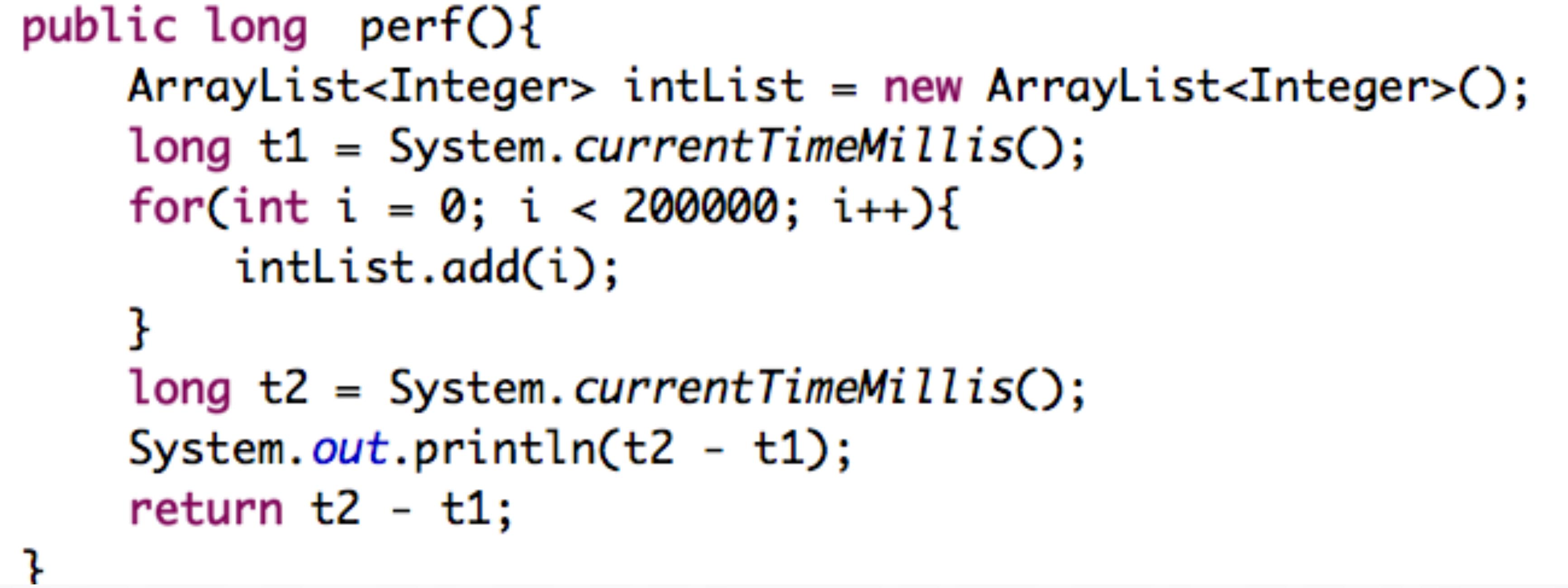
Iterator

- What’s the difference between for-loop traverse and iterator traverse

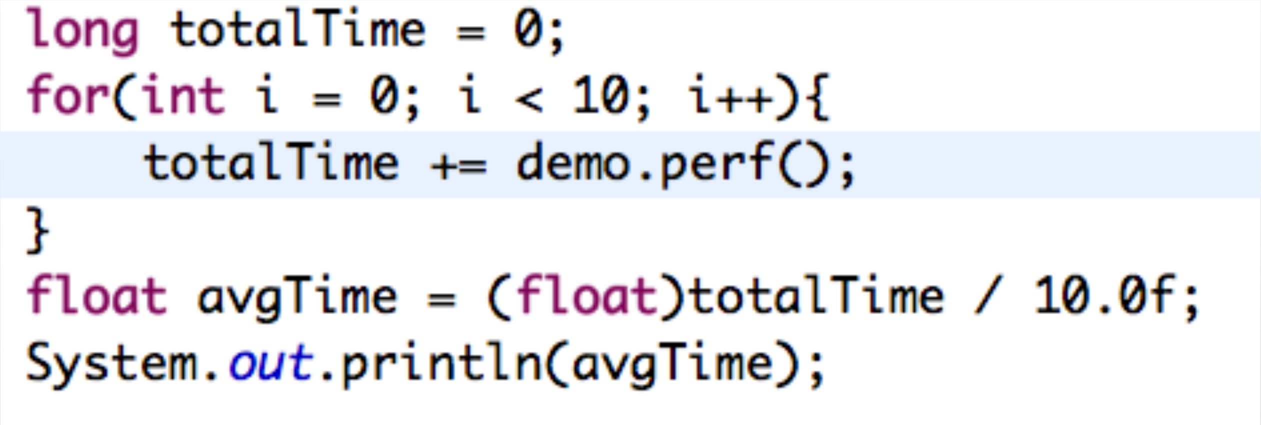
Performance

- How to do performance benchmark
  - get time
  - repeat action
  - multiple times and get average

Performance



```java
public long  perf(){
    ArrayList<Integer> intList = new ArrayList<Integer>();
    long t1 = System.currentTimeMillis();
    for(int i = 0; i < 200000; i++){
        intList.add(i);
    }
    long t2 = System.currentTimeMillis();
    System.out.println(t2 - t1);
    return t2 - t1;
}
```



```java
long totalTime = 0;
for(int i = 0; i < 10; i++){
    totalTime += demo.perf();
}
float avgTime = (float)totalTime / 10.0f;
System.out.println(avgTime);
```