Java – Collections API

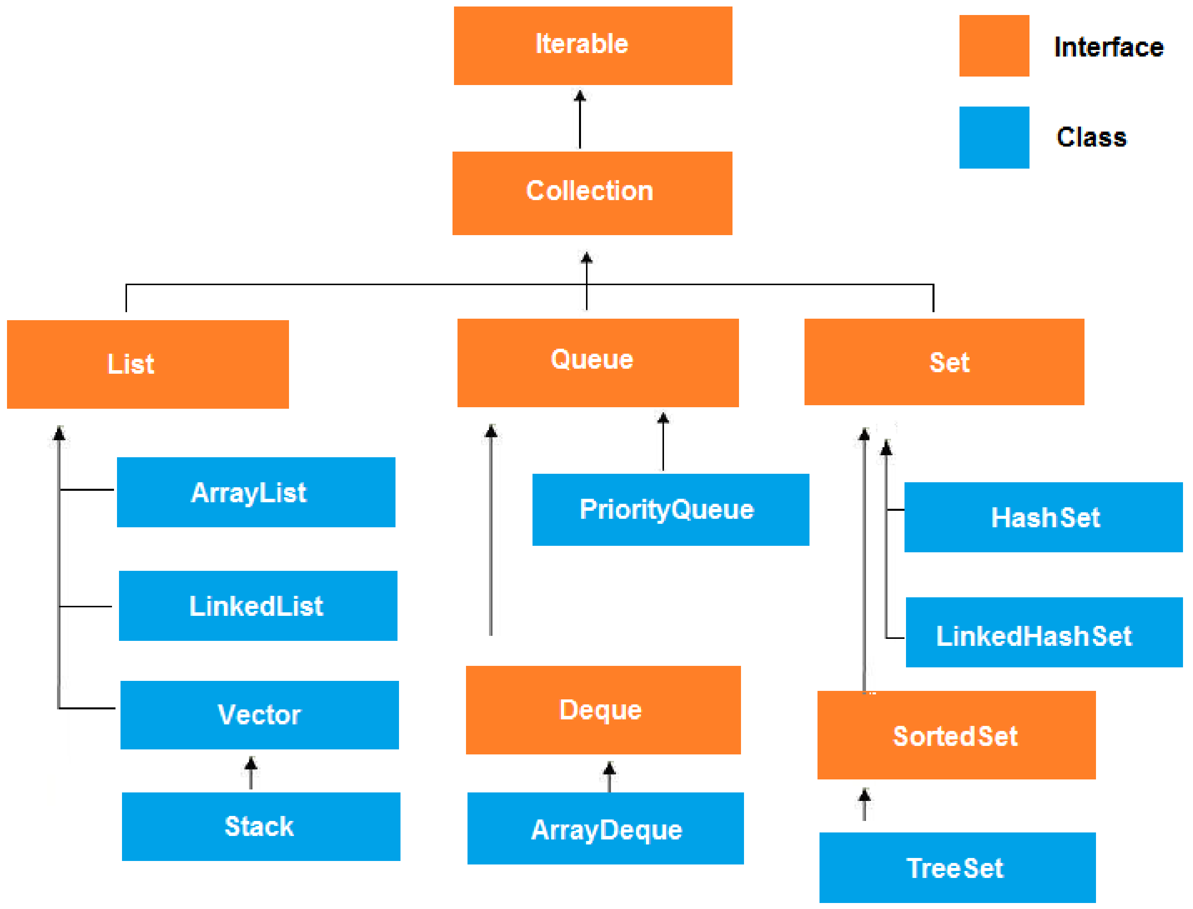
Topics

1) Collections
2) Collections Hierarchy
3) Collection Interface
4) List
5) Set
6) Queue
7) Collection classes
8) Generics
9) Non-Iterable Data Structures
10) Hashtable
11) HashMap

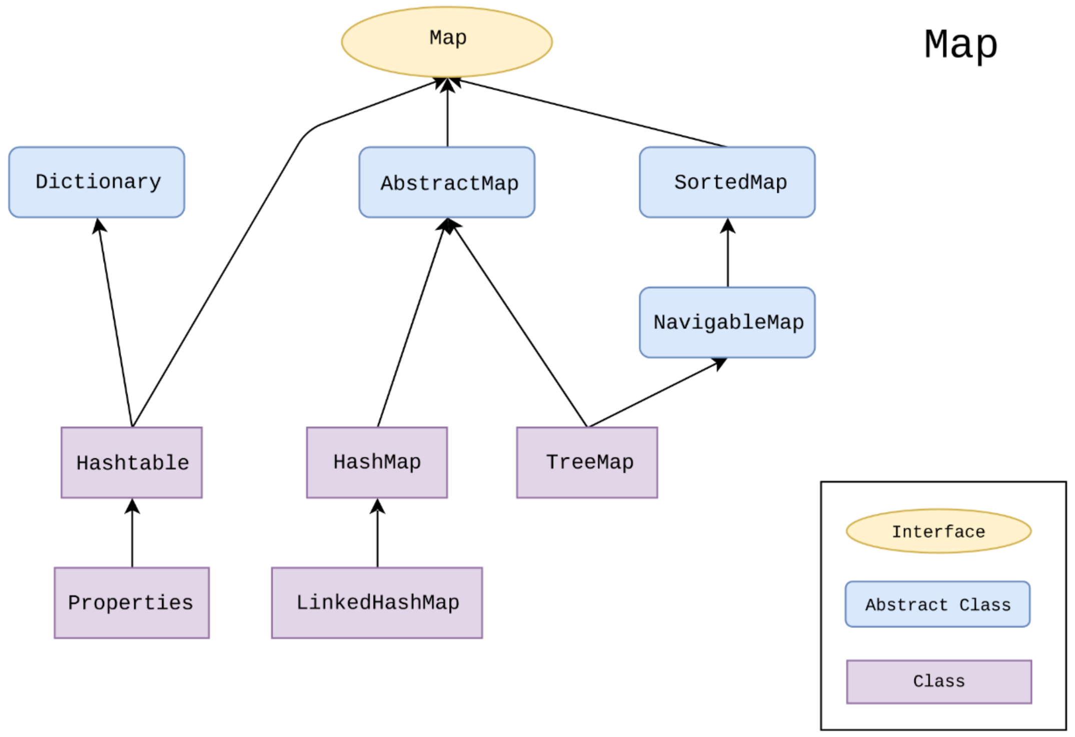
Collections

- A collection is an object that groups multiple elements into a single unit
- Used to store, retrieve, transform, and manipulate data
- Reduces programming effort by providing useful data structures and algorithms
- Increases program speed and quality
- Collections “collect” things

Collections Hierarchy



A Map is NOT an Iterable!



Collection Interface

- add(Object obj)
- addAll(Collection c)
- clear()
- contains(Object obj)
- equals (Object obj)
- isEmpty()
- iterator()
- remove(Object obj)
- removeAll(Object obj)
- size()

List Interface

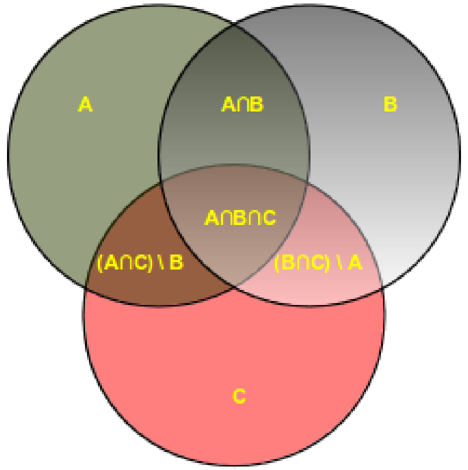
- Elements can be inserted or accessed by their position in the list
- Like array, List uses a zero-based index
- May contain duplicate elements
- Methods include:
  - add (int index, Object obj)
  - get (int index)
  - remove (int index)

List Implementations

- ArrayList
  - Array-backed list
  - Dynamic size – starts out by default at size 10, increases capacity 50% when limit reached
  - Faster retrieval – by index
  - Slower insertion & deletion – elements must be moved around within the array
- LinkedList
  - Implements List and Queue interfaces
  - Backed by a doubly-linked list
  - Consists of nodes with references to previous, next nodes
  - Faster insertion & deletion – simply change the references to prev/next node
  - Slower retrieval – must iterate through list to get to specific index
- Vector
  - Synchronized version of ArrayList

Set Interface

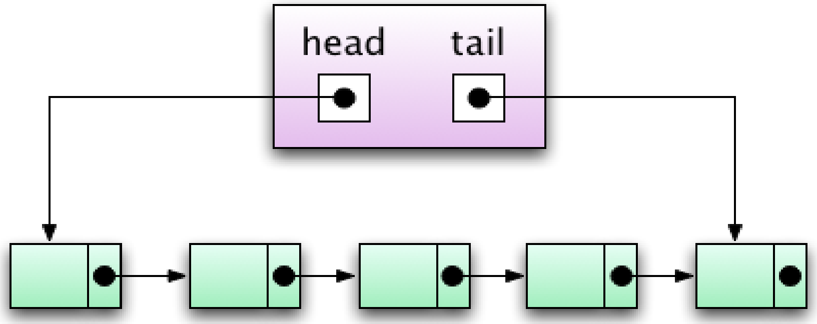
- The Set interface defines a collection of <u>distinct</u> elements
- Set does NOT allow duplicate elements
- Elements are accessed by iterating over the whole set
- Methods include:
  - add(Object obj)
  - clear()
  - remove(Object obj)
  - size()
  - toArray()



Set Implementations

- HashSet
  - Backed by a HashMap
  - **No guarantees of iteration order**
- TreeSet
  - Elements **ordered based on natural ordering** (or, alternatively, a Comparator)
- LinkedHashSet
  - Backed by a LinkedList which defines **iteration order, which is the same as the insertion order**

Queue Interface

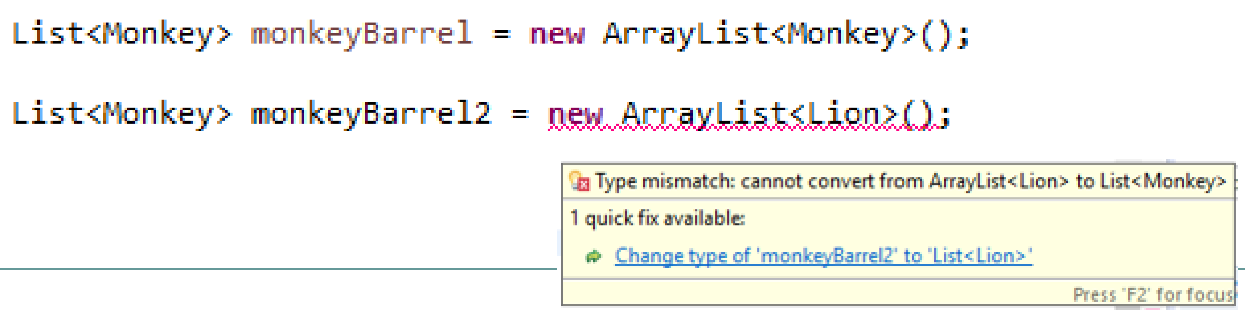


- Places objects on a “waiting list”, typically based on First-In-First-Out (FIFO)
- Elements are added to the tail of the queue
- Elements can be popped off the head of the queue
- Methods include:
  - add(Object obj)
  - element() : Returns the element at the front of the queue without removing it. If the queue is empty, it throws an exception
  - peek() : Returns the element at the front of the queue without removing it. If the queue is empty, it returns null.
  - poll() : Removes and returns the element at the front of the queue. If the queue is empty, it returns null.
  - remove() : Removes and returns the element at the front of the queue. If the queue is empty, it throws an exception.

Deque Interface

- Extends the Queue interface
- Short for “double-ended queue”
- Pronounced “deck”
- Supports element insertion and removal from **both ends of the queue**
- Can be used to implement a stack, with Last-In-First-Out (LIFO) behavior

Generics



```
List<Monkey> monkeyBarrel = new ArrayList<Monkey>();

List<Monkey> monkeyBarrel2 = new ArrayList<Lion>();
```

Type mismatch: cannot convert from ArrayList<Lion> to List<Monkey>

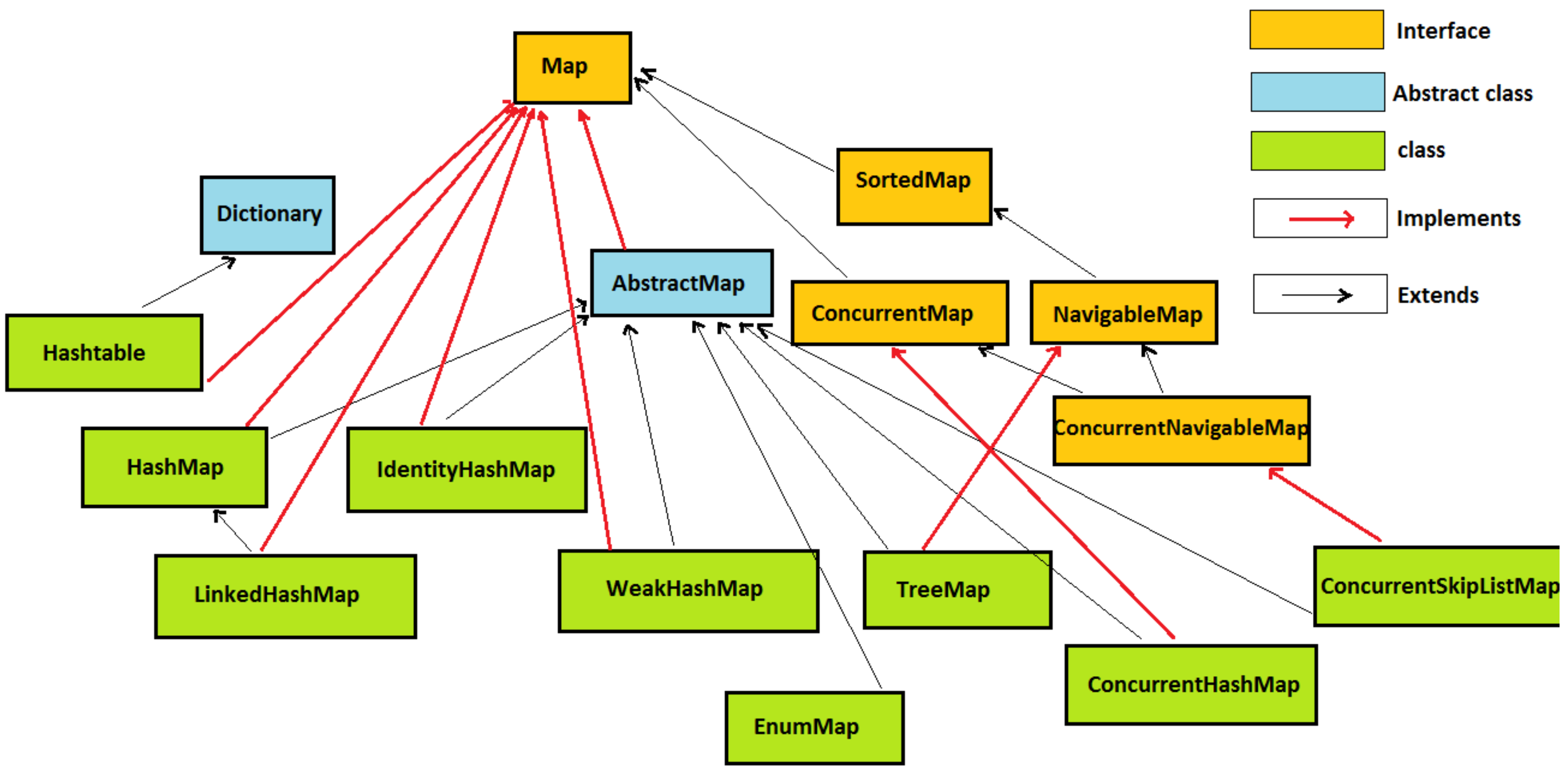
1 quick fix available:

Change type of 'monkeyBarrel2' to 'List<Lion>'

Press 'F2' for focus

- Generics enforce the type of object allowed in a Collection
- Uses the Diamond operator < >
- Insert the Class in the Diamond: <Employee>
- Generics provide compile-time safety

Non-Iterable Data Structures

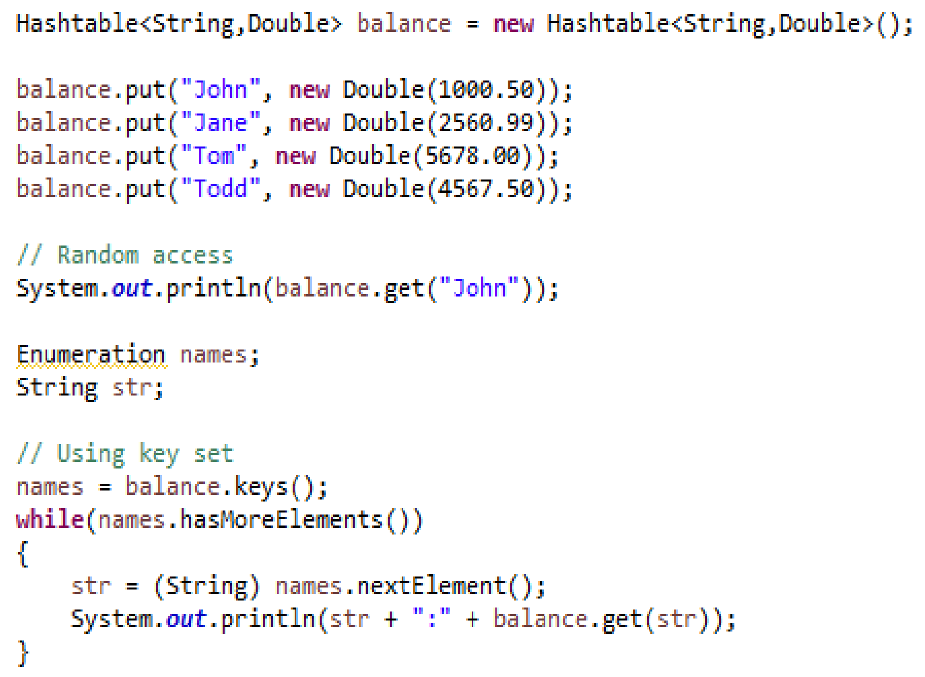


Map Interface

- The Map interface defines a data structure of **key – value pairs**
- Items are added and retrieved by their keys
- Map does NOT extend the Iterable interface, therefore it cannot be **directly** iterated over
- Instead, you can iterate over the Set of keys, a Collection of values, or a Set of key-value mappings
  - .keySet(), .entrySet(), or .values() methods allow iteration
- Other important methods:
  - .get()
  - .put()
  - .remove()
  - .replace()

Hashtable

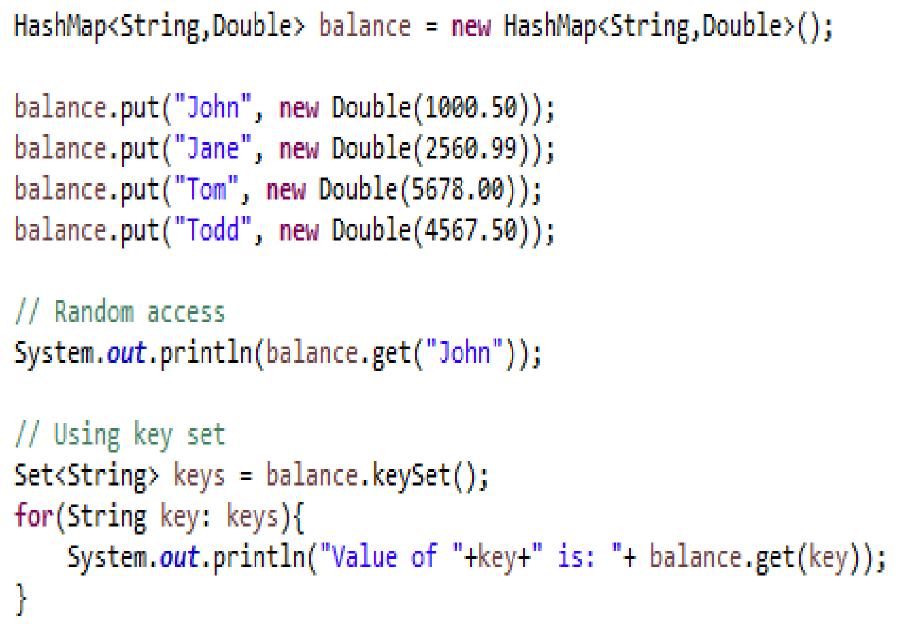
- Hashtable stores key/value pairs
- When using a Hashtable, you must specify:
  - An object that is used as a key
  - The value that you want linked to that key
- Allows random access by key
- Iterate over key set



```
Hashtable<String,Double> balance = new Hashtable<String,Double>();

balance.put("John", new Double(1000.50));
balance.put("Jane", new Double(2560.99));
balance.put("Tom", new Double(5678.00));
balance.put("Todd", new Double(4567.50));

// Random access
System.out.println(balance.get("John"));

Enumeration names;
String str;

// Using key set
names = balance.keys();
while(names.hasMoreElements())
{
    str = (String) names.nextElement();
    System.out.println(str + ":" + balance.get(str));
}
```

HashMap

- HashMap stores key/value pairs
- When using a HashMap, you must specify:
  - An object that is used as a key
  - The value that you want linked to that key
- Allows random access by key
- Iterate over key set and get values



```java
HashMap<String,Double> balance = new HashMap<String,Double>();

balance.put("John", new Double(1000.50));
balance.put("Jane", new Double(2560.99));
balance.put("Tom", new Double(5678.00));
balance.put("Todd", new Double(4567.50));

// Random access
System.out.println(balance.get("John"));

// Using key set
Set<String> keys = balance.keySet();
for(String key: keys){
    System.out.println("Value of "+key+" is: "+ balance.get(key));
}
```

Hashtable VS HashMap

Hashtable

- Thread-safe, **synchronized**
- Does not allow **null** keys and **null** values
- Uses Enumeration to iterate key set
- Legacy class

hashmap

- Not thread-safe
- Allows one **null** key and any number of **null** values
- Uses iterator or for:each loop to iterate over key set
- Better performance

TreeMap

- Map is **sorted** based on natural ordering (or a Comparator)
- Guaranteed O(log(n)) time for get, put, and remove operations
- Not synchronized

Common Concrete Collection Classes

Class	Synchronized?	Unique?	Sorted?	Retrieval	Insertion
ArrayList	✗	✗	✗	O(1)	O(n)
LinkedList	✗	✗	✗	O(n)	O(1)
Vector	✓	✗	✗	O(1)	O(n)
HashSet	✗	✓	✗	O(1)	O(1)
TreeSet	✗	✓	✓	O(log(n))	O(log(n))
ArrayDeque	✗	✗	✗	O(1)	O(n)
ArrayBlockingQueue	✓	✗	✗	O(1)	O(n)
PriorityQueue	✗	✗	✗	O(1)	O(log(n))
HashMap	✗	✗	✗	O(1)	O(1)
TreeMap	✗	✗	✓	O(log(n))	O(log(n))

Assignment

- Create an ArrayList and a HashSet. Insert 3 objects into each.
- Iterate over each collection and print each object.
- Review Vector and other collections online.
- Review Comparator and Comparable in your book or online.
- Review java.util.Collections methods (sort, reverseOrder, shuffle, etc.)