Java Collections Framework

COMP 3111 tutorial

What is a collection?

- A collection — sometimes called a container — is simply an object that groups multiple elements into a single unit
- Collections are used to store, retrieve, manipulate, and communicate aggregate data
- Examples
  - A poker hand (a collection of cards)
  - A telephone directory (a mapping of names to phone numbers)

Collections Framework

- A collections framework is a unified architecture for representing and manipulating collections
- Well-know collections framework
  - Java Collections Framework
  - C++ Standard Template Library (STL)

Collections Framework

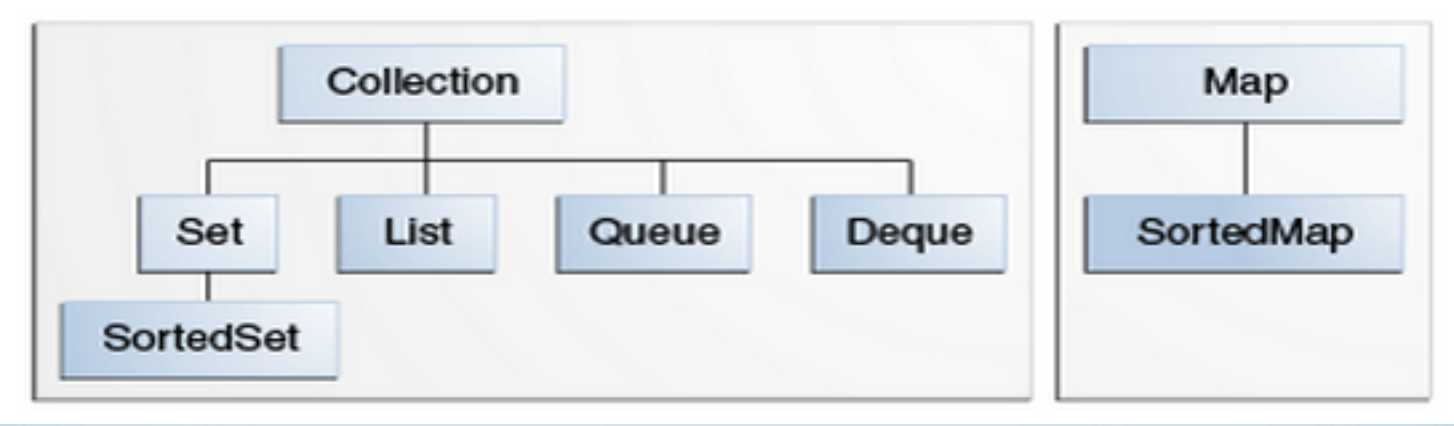
- All collections frameworks contain the following
  - Interfaces
    - Abstract Data Types that represent collections)
  - Implementations
    - Actual implementation of ADTs
    - May use different data structure to implement the same ADT
  - Algorithms
    - Methods that perform useful computations
    - Examples: Searching, Sorting…

Benefits of using Collections Framework

- Reduces programming effort
- Increases program speed and quality
- Allows interoperability among unrelated APIs
- Reduces effort to learn and to use new APIs
- Fosters software reuse
- ...

Collections and Maps (Interfaces)

- There are two kinds of interfaces
  - Collection: stores things
  - Map: stores key-value pairs
- Note: You can't create an interface object!



Implementations

- Implementations are the data objects used to store collections, which implement the interfaces

General-purpose Implementations

Interfaces	Hash table Implementations	Resizable array Implementations	Tree Implementations	Linked list Implementations	Hash table + Linked list Implementations
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Too many concepts! Let's start with an example using LinkedList

Example: Using Collections

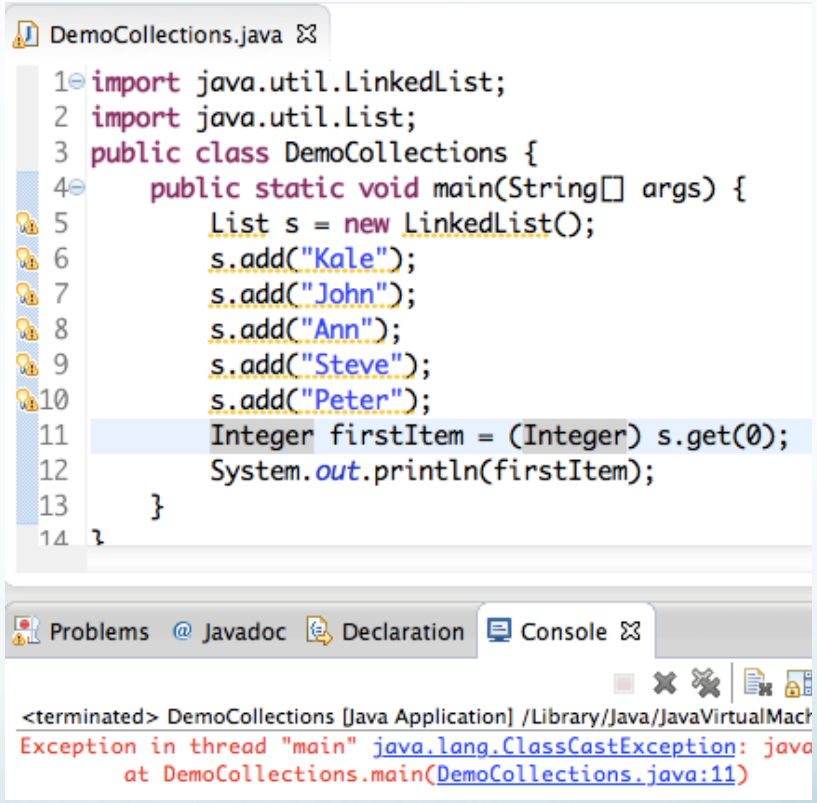
- Use Java Collections Framework to store the following 5 strings: {“Kale”, “John”, “Ann”, “Steve”, “Peter”} in a linked list
- Retrieve the first item from the linked list

Using Collections (before JDK 5)

```
import java.util.LinkedList;
import java.util.List;
public class DemoCollections {
    public static void main(String[] args) {
        List s = new LinkedList();
        s.add("Kale");
        s.add("John");
        s.add("Ann");
        s.add("Steve");
        s.add("Peter");
        String firstItem = (String) s.get(0);
        System.out.println(firstItem);
    }
}
```

Using Collections (before JDK 5)

- Problem: It will cause runtime error if an inappropriate type-casting is used to get the first item (e.g. casting to Integer instead of String)
- As the return type is "Object", which is the base class of both Integer and String, the error can't be resolved during the compile time



Using Collections with Generics

- Revised example:

```
import java.util.LinkedList;
import java.util.List;
public class DemoCollections {
    public static void main(String[] args) {
        List<String> s = new LinkedList<String>();
        s.add("Kale");
        s.add("John");
        s.add("Ann");
        s.add("Steve");
        s.add("Peter");
        String firstItem = s.get(0);
        System.out.println(firstItem);
    }
}
```

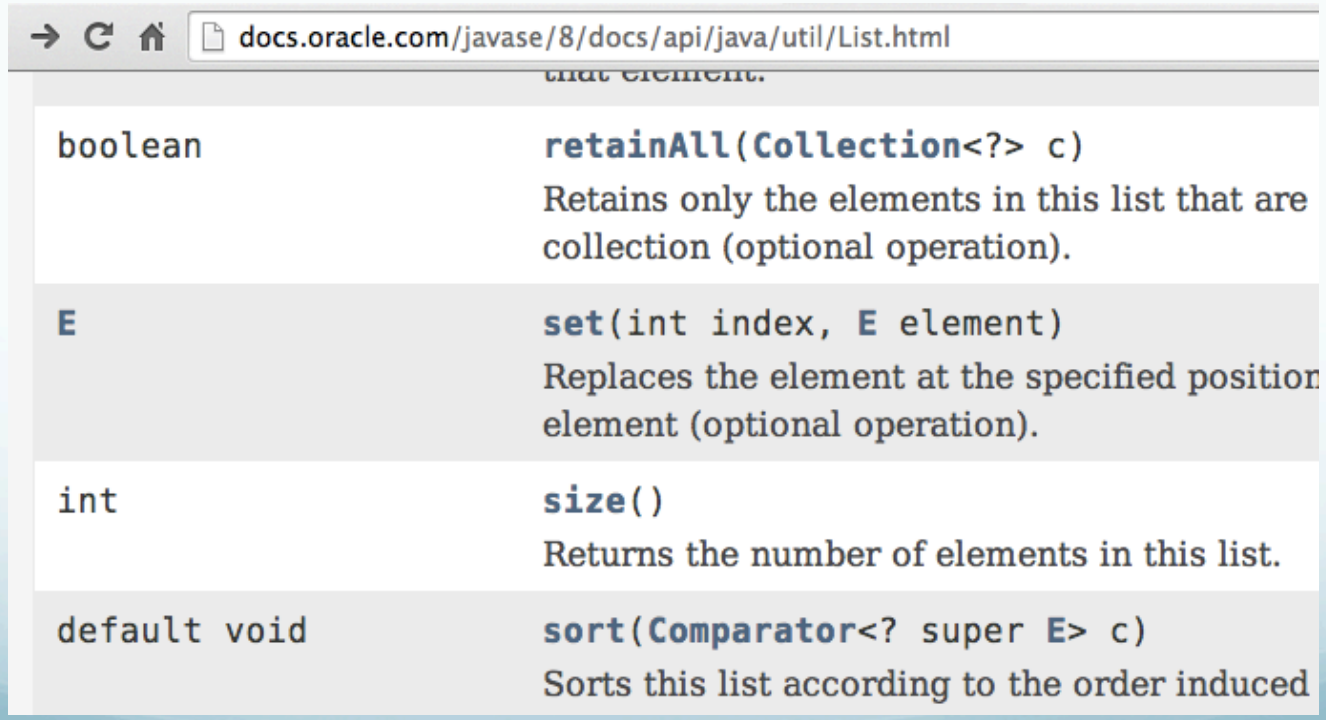
Generic (Java)

```
public class GenericsDemo {
    public static void main(String[] args) {
        IntPair p1 = new IntPair(1,2);
        System.out.println( p1.first() + " " + p1.second());

        Pair<Integer> p2 = new Pair<Integer>(1,2);
        System.out.println( p2.first() + " " + p2.second());
    }
}
class IntPair {
    public IntPair(int f, int s) {first=f; second=s;}
    public int first() {return first;}
    public int second() {return second;}
    private int first, second;
}
class Pair<T> {
    public Pair(T f, T s) {first=f; second=s;}
    public T first() {return first;}
    public T second() {return second;}
    private T first, second;
}
```

What is ?

- You may see lots of confusing things (e.g. ?, ? Super, ? extends...) when you read the online documentation, for example:



? = wildcards

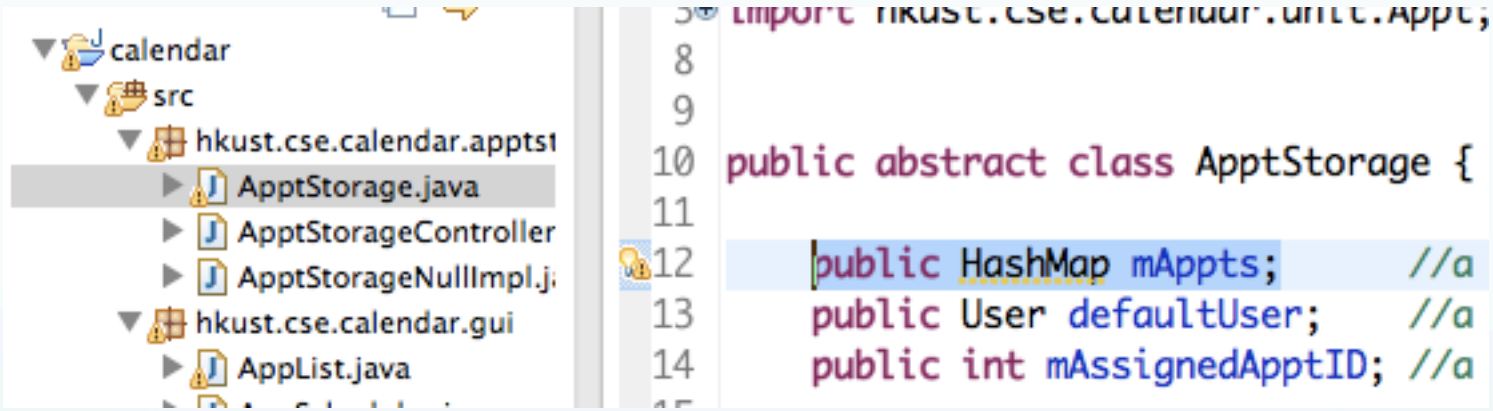
- References:
  - http://docs.oracle.com/javase/tutorial/java/generics/wildcards.html
- Unbounded Wildcards (?)
  - http://docs.oracle.com/javase/tutorial/java/generics/unboundedWildcards.html
- Lower Bounded Wildcards (? super ClassName)
  - http://docs.oracle.com/javase/tutorial/java/generics/lowerBounded.html
- Upper Bounded Wildcards (? extends ClassName)
  - http://docs.oracle.com/javase/tutorial/java/generics/upperBounded.html

Implementations on Collections and Maps

- List – ArrayList, LinkedList
- Set – HashSet, TreeSet, LinkedHashSet
- Map – HashMap, TreeMap, LinkedHashMap
- Queue – PriorityQueue
- Deque – ArrayQueue, LinkedList

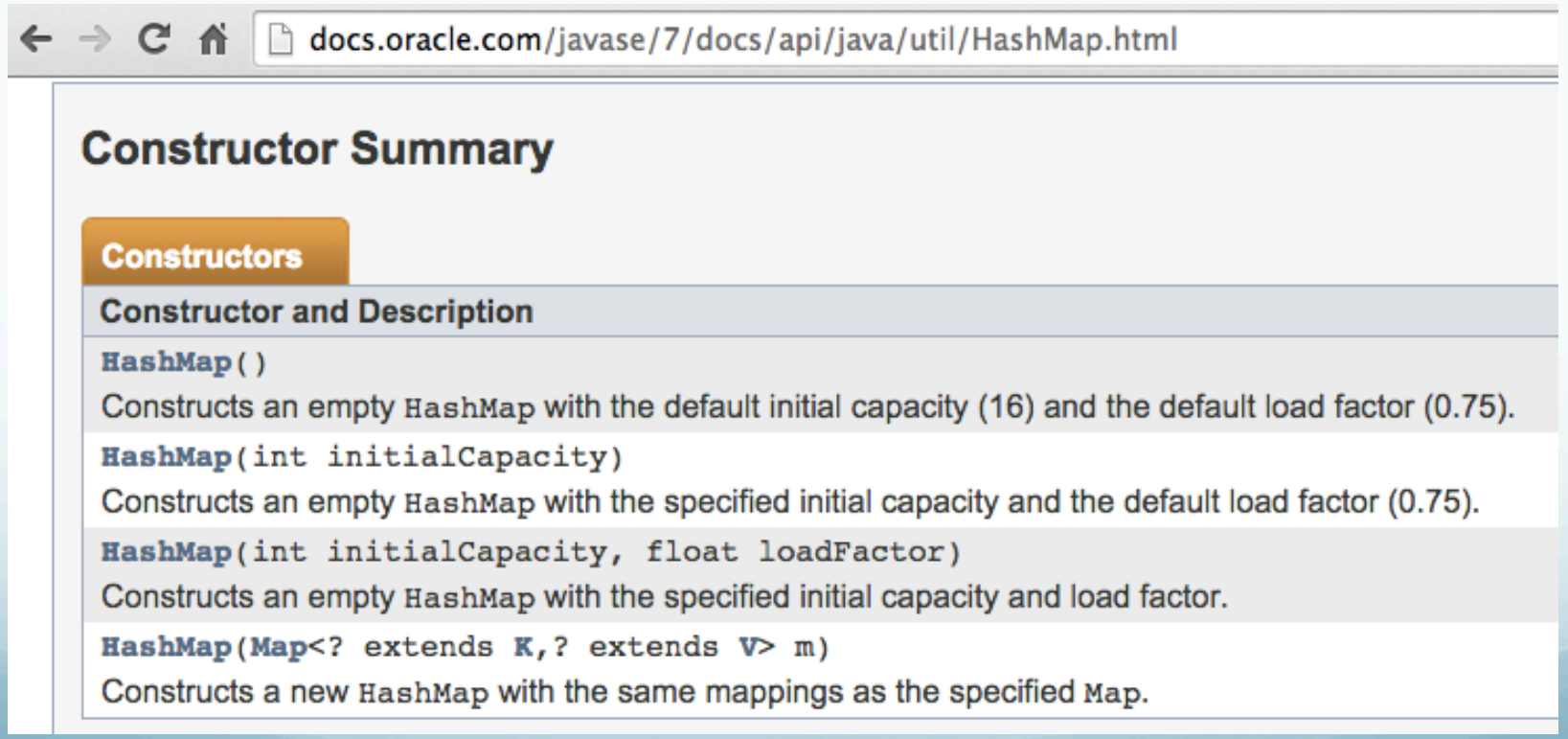
Example: HashMap

- HashMap is used in the calendar project, let's try to have some examples on HashMap



Example: HashMap

- How to construct a HashMap object? Read the documentation of HashMap constructor



What is the Map Interface?

- Read the documentation of Map Interface

Now, we should know how to create a HashMap

Example: HashMap

- Suppose I would like to create a telephone directory
- The telephone directory store key-value pairs, where keys are phone numbers and values are description
- Examples:
  - Key = 2358 8999
  - Description = HKUST Emergency Line (24 hours)
  - Key = 999
  - Description = Police, fire, ambulance (24 hours)

Example: HashMap

- A HashMap object is created as follows:

```
import java.util.HashMap;
import java.util.Map;
public class DemoHashMap {

    public static void main(String[] args) {
        Map<Integer, String> myPhoneDir = new HashMap<Integer, String>();
        myPhoneDir.put(999, "Police, fire, ambulance(24 hours)");
        myPhoneDir.put(23588999, "HKUST Emergency Line (24 hours)");

        Integer keyToSearch = 999;
        if ( myPhoneDir.containsKey(keyToSearch) )
            System.out.println( myPhoneDir.get(keyToSearch) );
        else
            System.out.println( "Key " + keyToSearch + " is not found!" );
    }
}
```

Interface V.S. Implementation

- What is the differences between:

```
Map<Integer, String> myPhoneDir = new HashMap<Integer, String>();

HashMap<Integer, String> myPhoneDir2 =  new HashMap<Integer, String>();
```

- Both lines won't cause compilation error
- Why it is better to use the first instead of the second?

Easy to change the underlying implementation

- For a specific problem, we may need to change the implementation to improve its performance
- By using the interface (instead of implementation), only change one line (e.g. HashMap => TreeMap)

```
// Map<Integer, String> myPhoneDir = new HashMap<Integer, String>();
Map<Integer, String> myPhoneDir = new TreeMap<Integer, String>();


myPhoneDir.put(999, "Police, fire, ambulance(24 hours)");
myPhoneDir.put(23588999, "HKUST Emergency Line (24 hours)");

Integer keyToSearch = 999;
if ( myPhoneDir.containsKey(keyToSearch) )
    System.out.println( myPhoneDir.get(keyToSearch) );
else
    System.out.println( "Key " + keyToSearch + " is not found!" );
```

Without using interface (Hard to change implementation)

- No problem – firstKey() is defined in TreeMap

```
TreeMap<Integer, String> myPhoneDir = new TreeMap<Integer, String>();

myPhoneDir.put(999, "Police, fire, ambulance(24 hours)");
myPhoneDir.put(23588999, "HKUST Emergency Line (24 hours)");

System.out.println("First key: " + myPhoneDir.firstKey());
```

- Error – firstKey() is not defined in HashMap

```
HashMap<Integer, String> myPhoneDir = new HashMap<Integer, String>();
//TreeMap<Integer, String> myPhoneDir = new TreeMap<Integer, String>();

myPhoneDir.put(999, "Police, fire, ambulance(24 hours)");
myPhoneDir.put(23588999, "HKUST Emergency Line (24 hours)");

System.out.println("First key: " + myPhoneDir.firstKey());
```

Synchronization (thread-safety)

- For multi-threaded application, synchronization must be considered
- Adding synchronization features in Java Collections Framework. For example:

```
// Now, myPhoneDir is thread-safe
Map<Integer, String> myPhoneDir =
        Collections.synchronizedMap(new HashMap<Integer, String>());
```

- References:
  - http://docs.oracle.com/javase/tutorial/collections/implementations/wrapper.html
  - http://docs.oracle.com/javase/tutorial/essential/concurrency/

Algorithm

- Many useful algorithms are also defined in the Collections class. For example:

```
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
public class DemoAlgorithm {
    public static void main(String[] args) {
        List<Integer> myList = Arrays.asList(1,9,2,8,9,9,9);
        System.out.println( "Frequency of 9 : "
          + Collections.frequency(myList, 9));
        System.out.println(myList);
        Collections.sort(myList);
        System.out.println(myList);
        Collections.reverse(myList);
        System.out.println(myList);
    }
}
```