Collections (more…)

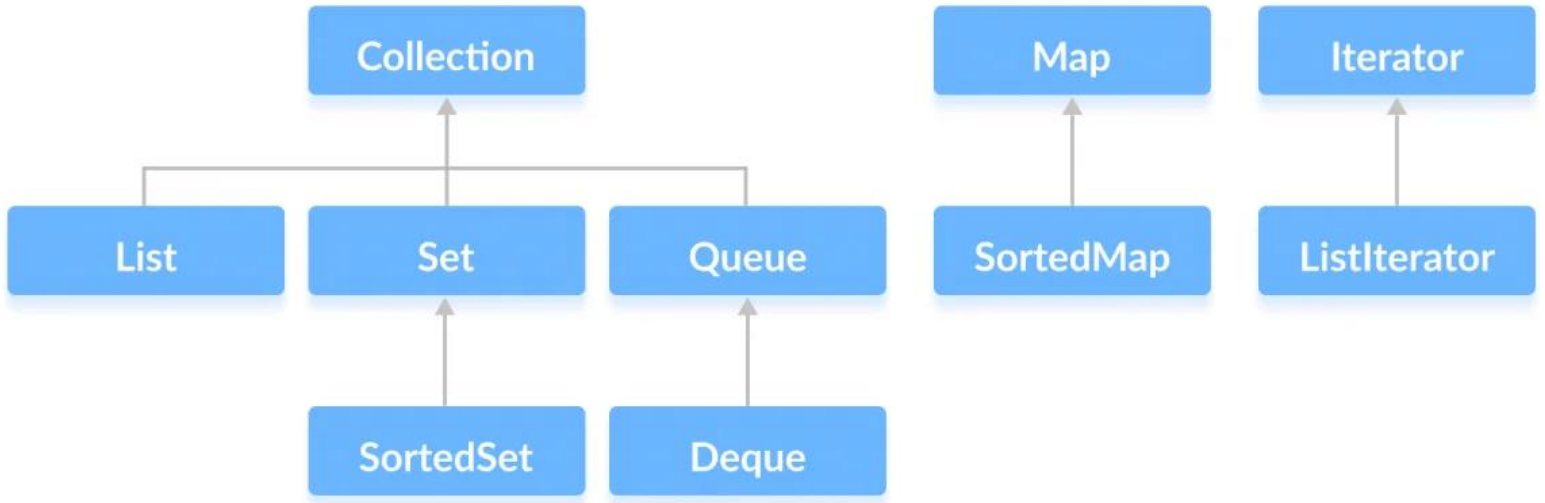
Dr. Krishnendu Guha (kguha@ucc.ie)

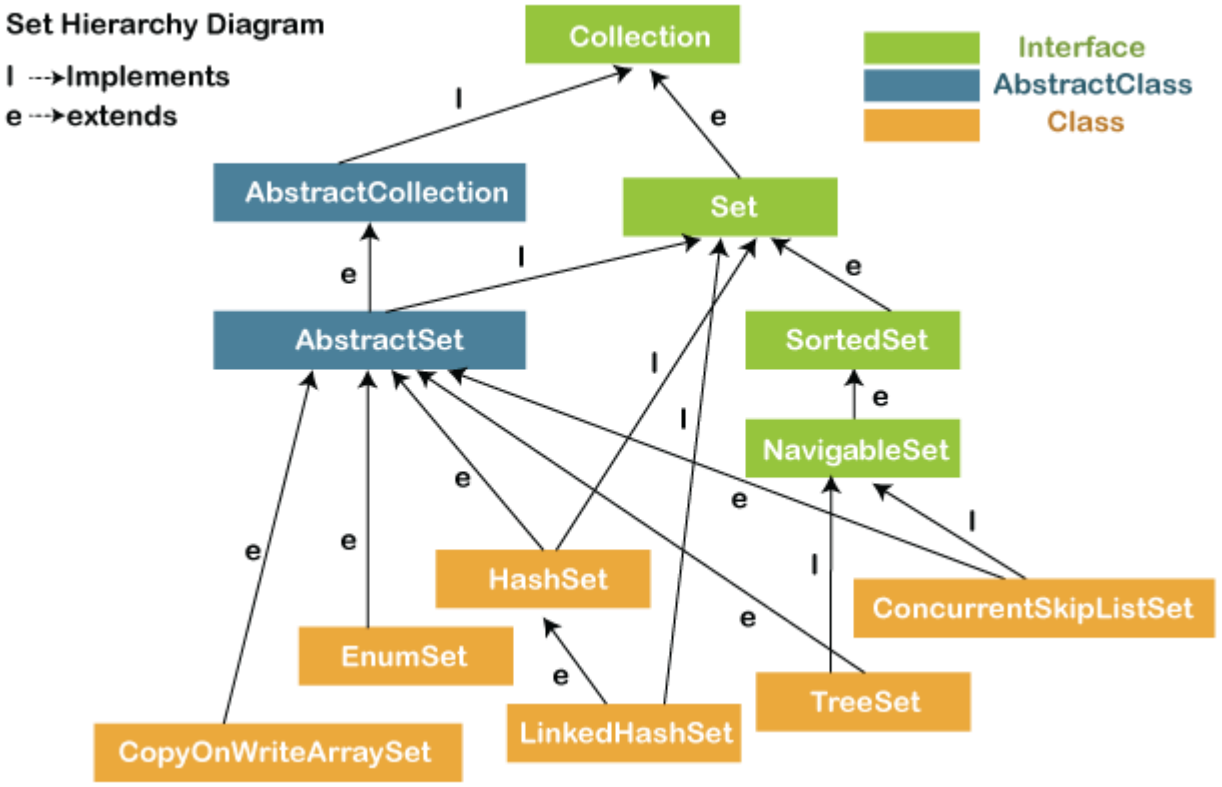
Teaching Assistants

Zarrar Haider (123120583@umail.ucc.ie)

Rheena Blas (120347046@umail.ucc.ie)

Java Collections Framework




Set Hierarchy Diagram
I --->Implements
e --->extends
Collection
Interface
AbstractClass
Class
AbstractCollection
Set
AbstractSet
SortedSet
NavigableSet
HashSet
ConcurrentSkipListSet
EnumSet
TreeSet
LinkedHashSet
CopyOnWriteArraySet

Sets

- A set is a collection of things that **does not allow duplicates** and **imposes no ordering** of items
- Two sets are said to be **equal if and only if they contain the same values**

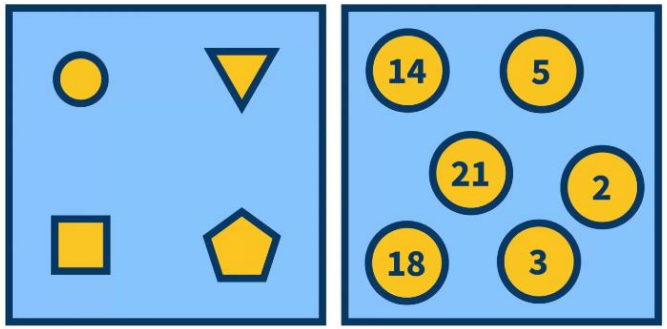
- Java collections provide **three classes that implement a set**:

  TreeSet, HashSet and LinkedHashSet

NOTE:

i **TreeSet** *orders the elements in the set*, whereas **HashSet does not**.

ii **LinkedHashSet orders its elements based on the order in which they were inserted** into the set



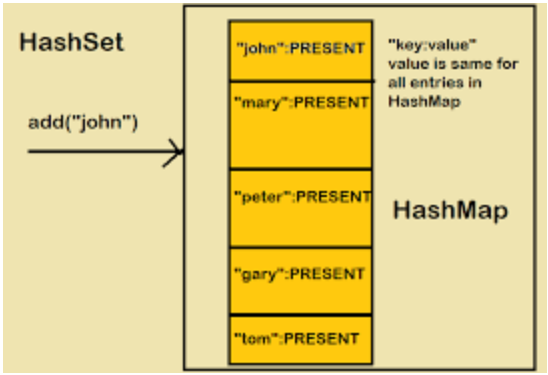
Set Interface

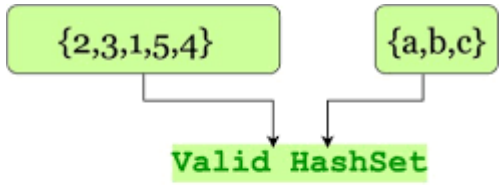
```
//Constructors:
TreeSet<E>()
TreeSet<E>(Collection)
HashSet<E>()
HashSet<E>(Collection)
//Methods:
add(E x)
remove(E x)
boolean contains(E x)
int size()
toString()
boolean isEmpty()
addAll(Collection) // set union
retainAll(Collection) // set intersection
removeAll(Collection) // set difference
boolean containsAll(Collection) // subset
clear()
Object[] toArray()
Iterator<E> iterator()
```

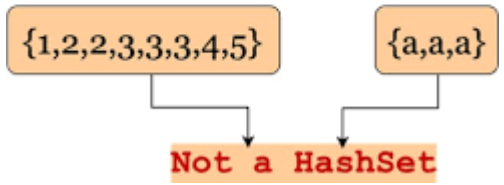
Example (Hash Set)

What are the characteristics of the output?

```
HashSet<Integer> set = new HashSet<Integer>();
for(int j = 0; j < 10; j++) {
        int x = (int)(Math.random() * 10);
        set.add(new Integer(x));
}
System.out.println();
System.out.println(set.toString());
```







Example (Hash Set)

What are the characteristics of the output?

```
HashSet<Integer> set = new HashSet<Integer>();
for(int j = 0; j < 10; j++) {
        int x = (int)(Math.random() * 10);
        set.add(new Integer(x));
}
System.out.println();
System.out.println(set.toString());
```

- Random numbers between [0,9] are generated.
- **Duplicate elements might be generated** but will be ***ignored by the add method***.
- Sample output: [2, 4, 9, 6, 3, 7, 0]

Hash Set Example: Search

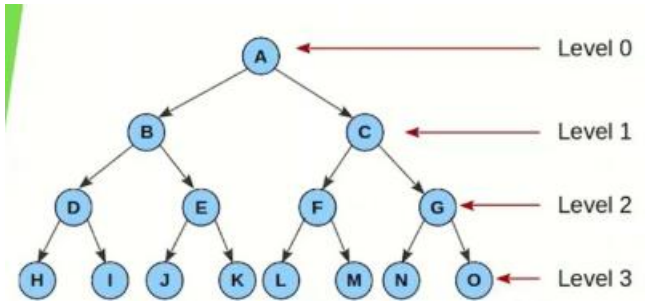
Find if 5 is an element of the set

```
if(set.contains(5)) {
        System.out.println("5 is an element of" + set.toString());
}
else {
        System.out.println("5 is not an element of" + set.toString());
}
```

Tree Set Example

What are the characteristics of the output?

```
TreeSet<Integer> t1 = new TreeSet<Integer>();
while(t1.size() < 10){
        int x = (int)(Math.random()*20);
        t1.add(new Integer(x));
}
System.out.println();
System.out.println(t1.toString());
```



Tree Set Example

What are the characteristics of the output?

```
TreeSet<Integer> t1 = new TreeSet<Integer>();
while(t1.size() < 10){
        int x = (int)(Math.random()*20);
        t1.add(new Integer(x));
}
System.out.println();
System.out.println(t1.toString());
```

- Random numbers between [0,19] are generated.
- ***Duplicate elements might be generated but will be ignored by the add method.***
- Elements will be sorted in ascending order.
- Sample output: [0, 2, 4, 5, 6, 7, 8, 9, 15, 19]

How to Traverse a Collection

The elements of a Collection can be visited by:

- using a **for-each loop**
- using what are called **Iterators**.

```
for(Integer x : setA){
        System.out.println(x);
}
//print only the even values in set
System.out.print('[');
for(Integer x : set){
        if(x % 2 == 0) {
                System.out.print(x) + " ";
        }
}
System.out.println(']');
```

Iterator Interface

An **Iterator** is an **object** that enables you to ***traverse*** *through a collection* and to ***remove*** *elements from the collection selectively*, if desired.

```
public interface Iterator<E> {
        boolean hasNext();
        E next();
        void remove(); //optional
}
```

Suppose k is a collection of integers

```
Iterator<Integer> t = k.iterator();
while(t.hasNext()){
        Integer x = t.next();
        // process x
}
```

Print using Iterator

```
//print elements of collection setA
Iterator<Integer> it = setA.iterator();
System.out.print('[');
while(it.hasNext()){
        Integer x = it.next();
        System.out.print(x.intValue()+" ");
}
System.out.println(']');
```

Remove Values using Iterator

```
//remove even values from set t1
it = t1.iterator();
while(it.hasNext()){
        Integer x = it.next();
        if(x % 2 == 0) {
                it.remove();
        }
}
System.out.println(t1.toString());
```

Comparable Interface

- Java Comparable interface is ***used to order the objects of the user-defined class.***
- This **interface is found in java.lang package** and **contains only one method** named ***compareTo(Object)***

```
public interface Comparable<T>
{
        public int compareTo(T o);
}
```

Using Set using User Defined Classes

```
class Book implements Comparable<Book>{
        private String title;
        Book(String t) { title = t; }
        public String get() { return title; }
        public String toString() { return title; }
        public boolean equals(Object p){
                Book b = (Book)p;
                return(title.equals(b.title));
        }
        public int compareTo(Book b){
                return title.compareTo(b.title);
        }
        public int hashCode(){ return title.hashCode(); }
}
```

Using Set with User Defined Classes

```
import java.util.*;
public static void main(String[] args) {
        HashSet<Book> d = new HashSet<Book>();
        d.add(new Book("Ulysses"));
        d.add(new Book("Dubliners"));
        d.add(new Book("Ulysses"));
        System.out.println(d.toString());
        System.out.println(d.contains(new Book("Ulysses")));
        }
}
```

It is important to note that if **we did not implement the equals method the rule that sets do not allow duplicates would no longer apply** and the **contains method would no longer work**.
In fact the program would give the following output:
[Ulysses, Ulysses, Dubliners]
false

Map Interface in Java

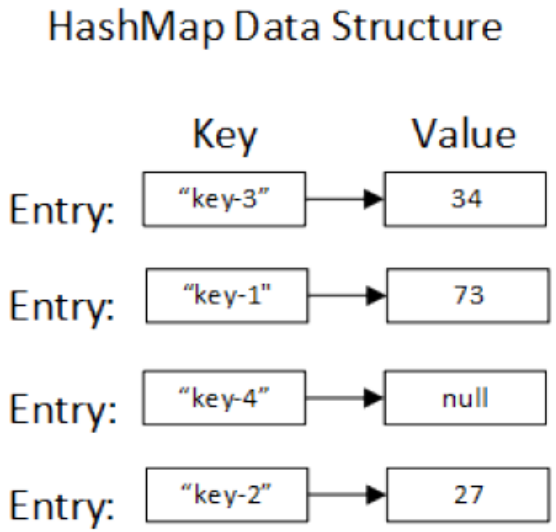
Map interface is present in java.util package **represents a mapping between a key and a value.**

- The Map interface is **not a subtype of the Collection interface**.
- A map **contains unique keys**.
- **Maps are perfect to use for key-value association** mapping such as **dictionaries**.
- The maps are used to **perform lookups by keys** or when someone **wants to retrieve and update elements by keys.**
- There are **two interfaces** for implementing Map in java.

They are Map and SortedMap, and three classes: HashMap, TreeMap, and LinkedHashMap.

Hash Map

- A HashMap in Java is an implementation of the Map interface which works on a hashing principle.



Some useful methods:

- V put(Object key, Object value)
- V remove(Object key)
- Set keySet()
- void clear()
- boolean containsKey(Object key)
- V get(Object key)
- boolean containsValue(Object value)
- etc.

- Once you've decided which Java interface to use, you choose an implementation (or implement a new one of your own)

Interface	Implementations	
Set	HashSet & its subclasses	A hash table
List	ArrayList	Resizable array
	LinkedList	Doubly linked list
Map	HashMap & its subclasses	Hash table
SortedSet	TreeSet	Balanced binary tree
SortedMap	TreeMap	Balanced binary tree

- (Where Java needs to test for equality of objects in these data structures, it will use your equals method. For the hash tables, it will use your hashCode method)


Thank you