CS112
Sets (Part 1)
Chapters 21

Lecture 17

الفصل الدراسي الثاني 1443- Spring 2022

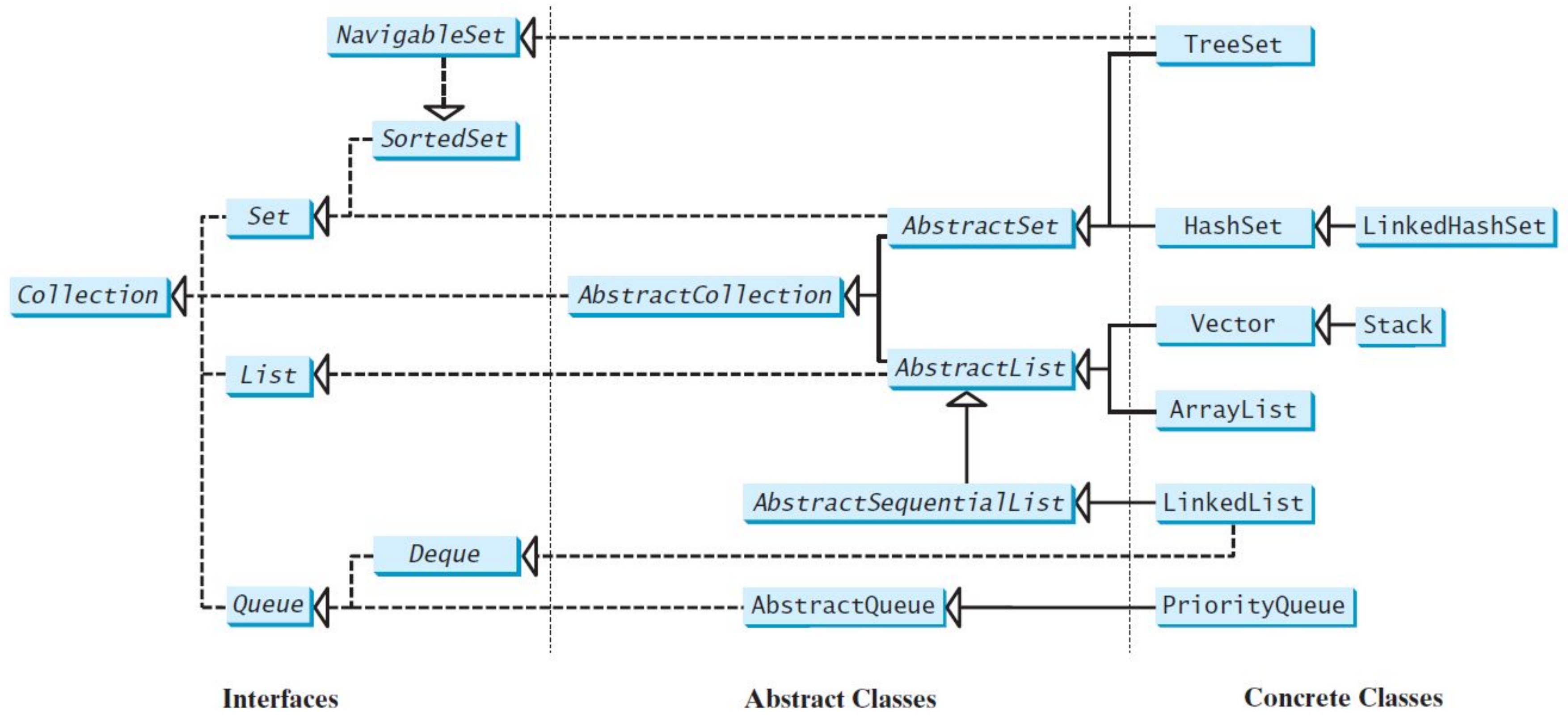
College of Computer Science and Engineering

Motivations

- The “**No-Fly**” **list** is a list, created and maintained by some government, of people who are not permitted to board a commercial aircraft for travel in or out of the corresponding countries.
- Suppose we need to write a program that checks whether a person is on the No-Fly list.
- You can use a list to store names in the No-Fly list. However, a more efficient data structure for this application is a ***set***.

What is a set in Java programming?

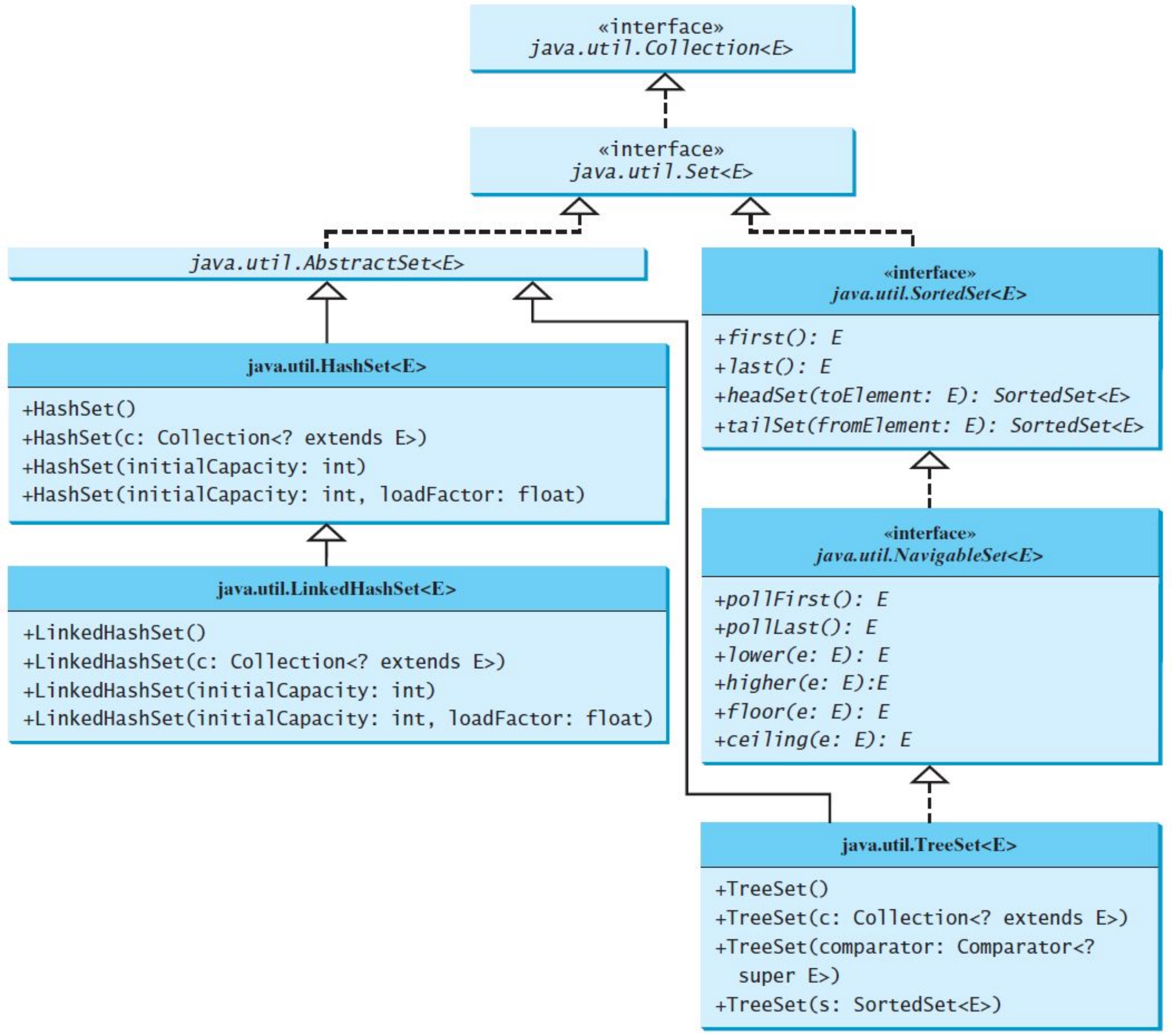
- A set is an efficient data structure for storing and processing nonduplicate elements.



The Set Interface

- The Set interface extends the Collection interface.
- It does not introduce new methods or constants, but it stipulates that an instance of Set contains no duplicate elements.
- The concrete classes that implement Set must ensure that no duplicate elements can be added to the set:
    - That is no two elements e1 and e2 can be in the set such that e1.equals(e2) is true.

The Set Interface Hierarchy


«interface»
java.util.Collection<E>
«interface»
java.util.Set<E>
java.util.AbstractSet<E>
«interface»
java.util.SortedSet<E>
+first(): E
+last(): E
+headSet(toElement: E): SortedSet<E>
+tailSet(fromElement: E): SortedSet<E>
java.util.HashSet<E>
+HashSet()
+HashSet(c: Collection<? extends E>)
+HashSet(initialCapacity: int)
+HashSet(initialCapacity: int, loadFactor: float)
java.util.LinkedHashSet<E>
+LinkedHashSet()
+LinkedHashSet(c: Collection<? extends E>)
+LinkedHashSet(initialCapacity: int)
+LinkedHashSet(initialCapacity: int, loadFactor: float)
«interface»
java.util.NavigableSet<E>
+pollFirst(): E
+pollLast(): E
+lower(e: E): E
+higher(e: E):E
+floor(e: E): E
+ceiling(e: E): E
java.util.TreeSet<E>
+TreeSet()
+TreeSet(c: Collection<? extends E>)
+TreeSet(comparator: Comparator<? super E>)
+TreeSet(s: SortedSet<E>)

The AbstractSet Class

- The AbstractSet class is a convenience class that extends AbstractCollection and implements Set.
- The AbstractSet class provides concrete implementations for the equals method and the hashCode method.
- The hash code of a set is the sum of the hash code of all the elements in the set.
- Since the size method and iterator method are not implemented in the AbstractSet class, AbstractSet is an abstract class.

Concrete Classes

- There are three concrete classes that uses sets:
  1. HashSet
  2. LinkedHashSet
  3. TreeSet

The HashSet Class

- The HashSet class is a concrete class that implements Set.
- It can be used to store duplicate-free elements.
- For efficiency, objects added to a hash set need to implement the hashCode method in a manner that properly disperses the hash code.

How to use the HashSet Class

- You can create an empty hash set using its no-arg constructor.
- You can also create a hash set from an existing collection.
- By default, the initial capacity is 16 and the load factor is 0.75.
- If you know the size of your set, you can specify the initial capacity and load factor in the constructor. Otherwise, use the default setting. The load factor is a value between 0.0 and 1.0.

What is a load factor?

- The load factor measures how full the set is allowed to be before its capacity is increased.
- When the number of elements exceeds the product of the capacity and load factor, the capacity is automatically doubled.
- For example, if the capacity is 16 and load factor is 0.75, the capacity will be doubled to 32 when the size reaches 12 (16*0.75 = 12).
- A higher load factor decreases the space costs but increases the search time.
- Generally, the default load factor 0.75 is a good tradeoff between time and space costs.
- Recommended reading: Chapter 27

The hashCode() method

- For efficiency, objects added to a hash set need to implement the hashCode method in a manner that properly disperses the hash code.
- The hash codes of two objects must be the same if the two objects are equal.
- Two unequal objects may have the same hash code, but you should implement the hashCode method to avoid too many such cases.
- See TestHashSet.java