VIETNAM AUSTRALIA Vocational College

Slide-1.2: Set, HashSet, LinkedHashSet, TreeSet

Giảng viên: Nguyễn Bá Minh Đạo

Nội dung

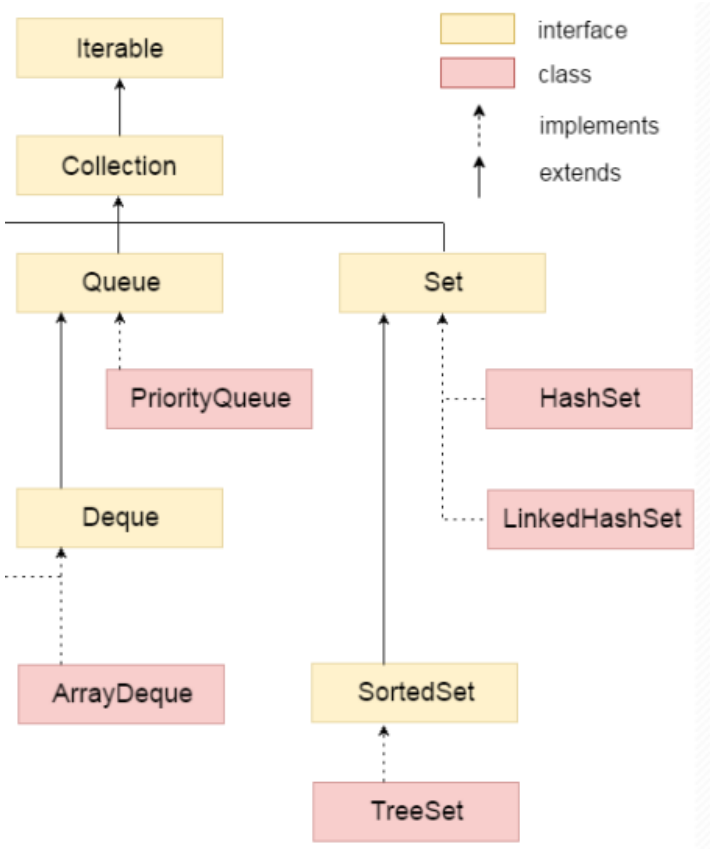
1. Set, HashSet, LinkedHashSet, TreeSet
2. So sánh HashSet, LinkedHashSet, TreeSet

Set, HashSet, LinkedHashSet, TreeSet

Giới thiệu Set Interface

- **Set** (tập hợp) trong Java là một Interface, được kế thừa từ Interface **Collection**.
- **Set** không chứa các phần tử trùng nhau.
- Interface **java.util.Set** được định nghĩa như sau:

```
public interface Set<E> extends Collection<E> {

}
```



Set, HashSet, LinkedHashSet, TreeSet

Các phương thức (method) của Set interface

Phương thức	Mô tả
public boolean add(Object element)	Thêm một phần tử vào collection.
public boolean addAll(Collection c)	Thêm các phần tử collection được chỉ định vào collection gọi phương thức này.
public boolean remove(Object element)	Xóa phần tử ra khỏi collection.
public boolean removeAll(Collection c)	Xóa tất cả các phần tử từ collection được chỉ định ra khỏi collection gọi phương thức này.
public boolean retainAll(Collection c)	Xóa tất cả các thành phần từ collection gọi phương thức này ngoại trừ collection được chỉ định.
public int size()	Trả về tổng số các phần tử trong collection.

Set, HashSet, LinkedHashSet, TreeSet

Các phương thức (method) của Set interface

Phương thức	Mô tả
public void clear()	Xóa tất cả các phần tử trong Collection, sau khi thực hiện phương thức này, Collection sẽ rỗng (empty)
public boolean contains(Object element)	Kiểm tra một phần tử có nằm trong Collection không
public boolean containsAll(Collection c)	Kiểm tra một Collection có chứa tất cả các phần tử của một Collection khác
public Iterator iterator()	Trả về một iterator.
public Object[] toArray()	Chuyển đổi collection thành mảng (array).
public boolean isEmpty()	Kiểm tra Collection có rỗng hay không.
public boolean equals(Object element)	So sánh 2 collection.
public int hashCode()	Trả về số hashcode của collection.

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng Set với lớp hiện thực là HashSet:

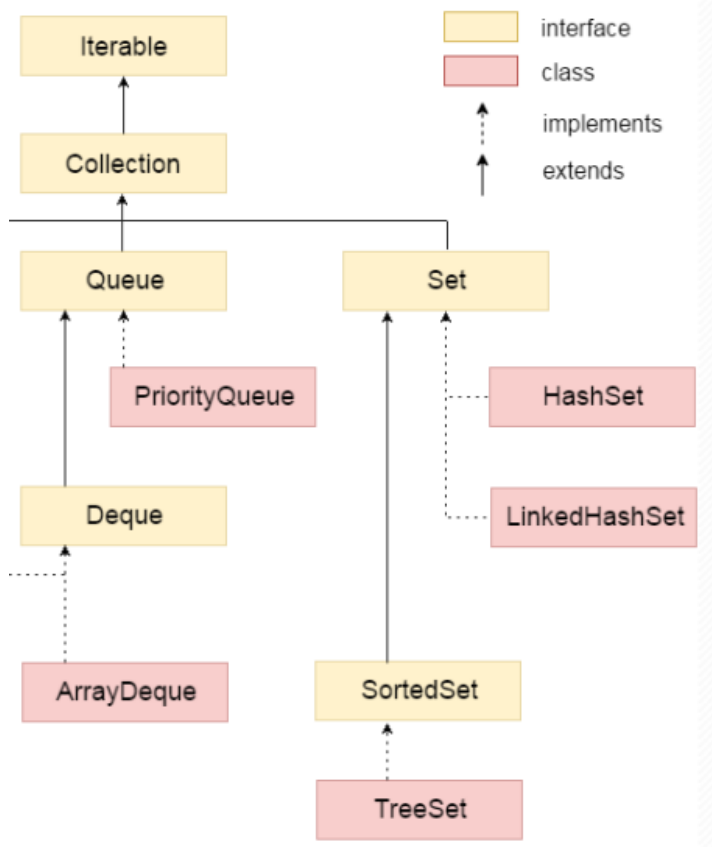
```
public class SetExample {
    public static void main(String[] args) {
        // Create set
        Set<String> items = new HashSet<>();
        items.add("A02"); // Add new item
        items.add("D03");
        items.add("D03"); // item is ignored
        items.add("01");
        items.add("04");

        // Show set through for-each
        for (String item: items) {
            System.out.print(item + " ");
        }
    }
}
```

- Trong ví dụ bên, Item *D03* thêm lần thứ 2 bị bỏ qua, bởi vì nó đã tồn tại trong *items*
- Thứ tự các phần tử được thêm vào không hiển thị đúng thứ tự, do chúng ta sử dụng **HashSet**.
- Để giữ đúng thứ tự thêm vào chúng ta có thể sử dụng **LinkedHashSet**,
- Hoặc sử dụng **TreeSet** để thêm các phần tử vào tập hợp được sắp xếp.

Set, HashSet, LinkedHashSet, TreeSet

Giới thiệu lớp HashSet

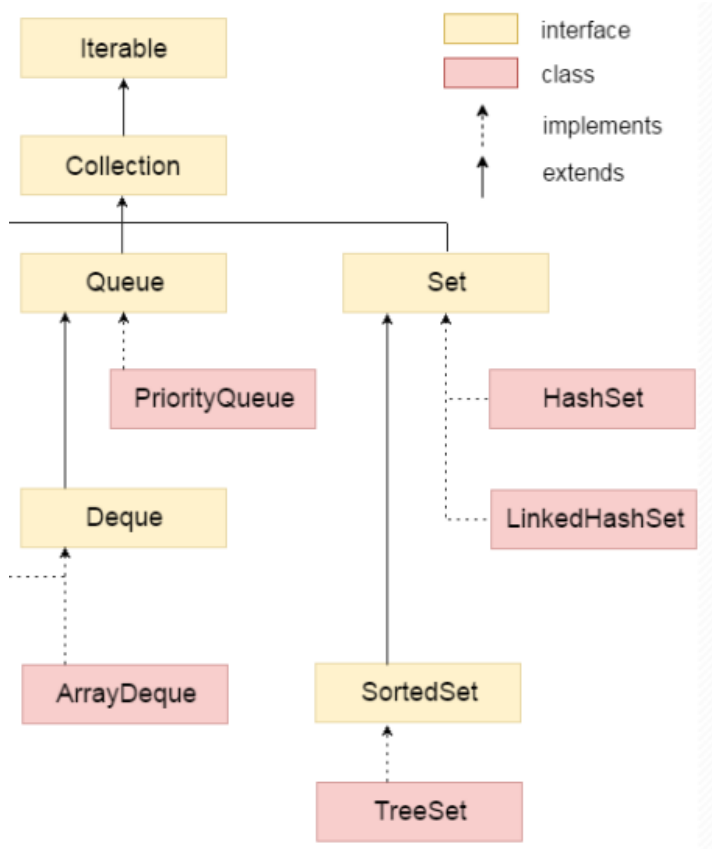
- Lớp **HashSet** trong Java kế thừa **AbstractSet** và triển khai **Set Interface**.
- **HashSet** tạo một Collection và sử dụng một **hash table** để lưu trữ.
- Một **hash table** lưu trữ thông tin bởi sử dụng một kỹ thuật được gọi là **hashing** (băm).



Set, HashSet, LinkedHashSet, TreeSet

Giới thiệu lớp HashSet

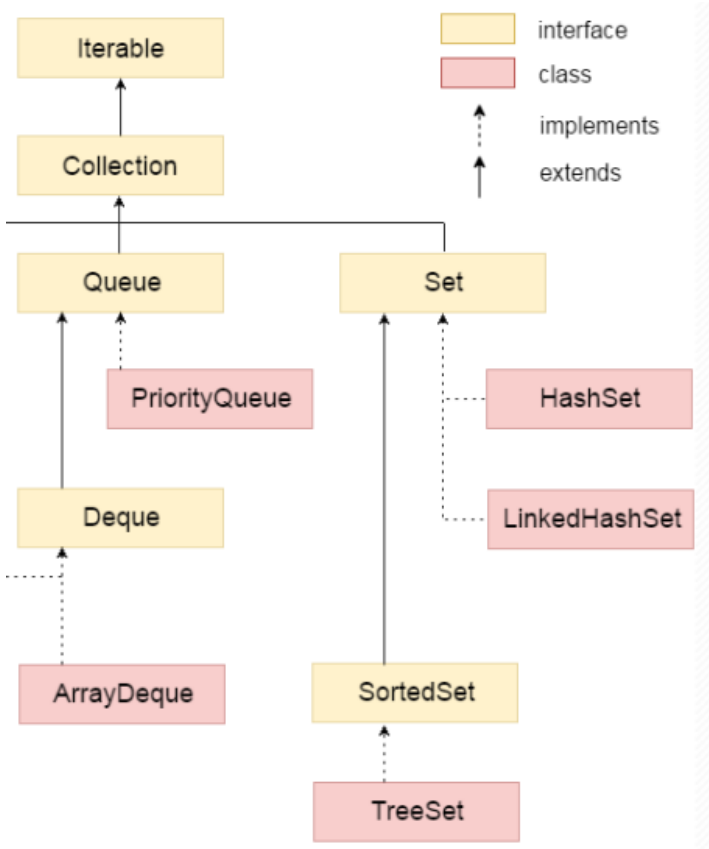
- Trong **hashing** (băm), nội dung mang tính chất thông tin của một **key** được sử dụng để quyết định một **value** duy nhất, được gọi là **hash code** của nó.
- **Hash code** sau đó được sử dụng như **index**, tại đó dữ liệu mà liên kết với **key** được lưu trữ.
- Phép biến đổi của **key** vào trong **hash code** của nó được thực hiện tự động.



Set, HashSet, LinkedHashSet, TreeSet

Các điểm quan trọng của lớp HashSet

- **HashSet** chỉ chứa các phần tử duy nhất, không chấp nhận 2 phần tử trùng nhau.
- **HashSet** lưu trữ các phần tử bằng cách sử dụng một cơ chế được gọi là **hashing** (băm).
- **HashSet** không đảm bảo thứ tự được thêm vào.
- **HashSet** cho phép chứa phần tử **null**.

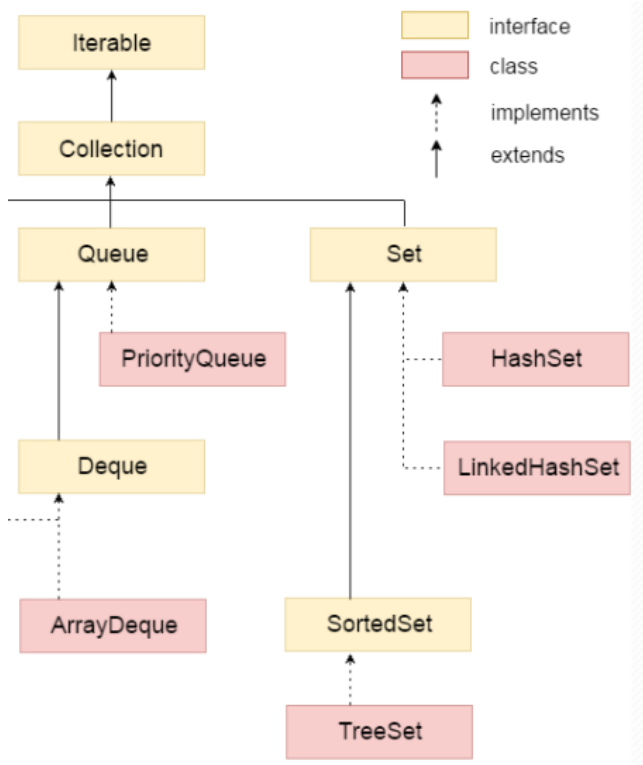


Set, HashSet, LinkedHashSet, TreeSet

Các điểm quan trọng của lớp HashSet

- Lớp **java.util.HashSet** được định nghĩa như sau:

```
public class HashSet<E> extends AbstractSet<E>
        implements Set<E>, Cloneable, java.io.Serializable {
    static final long serialVersionUID = -5024744406713321676L;

    private transient HashMap<E,Object> map;

    private static final Object PRESENT = new Object();

    public HashSet() {
        map = new HashMap<>();
    }
}
```



Set, HashSet, LinkedHashSet, TreeSet

Các phương thức khởi tạo (constructor) của lớp HashSet

Phương thức	Mô tả
HashSet()	khởi tạo một danh sách mảng trống
HashSet(Collection c)	khởi tạo một danh sách mảng với các phần tử của collection c

Các phương thức (method) của lớp HashSet

- Tương tự các phương thức đã được giới thiệu ở bài viết về **Set Interface**

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng HashSet với kiểu dữ liệu cơ bản (Wrapper)

```
public class HashSetExample01 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create set
        Set<String> set = new HashSet<>();
        set.add("Item01");
        set.add("Item02");
        set.add("Item03");
        set.add("Item04");
        set.add("Item05");
        set.add("Item02");
        set.add("Item03");

        // Show set through for-each
        for (String item : set) {
            System.out.print(item + " ");
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng HashSet với kiểu dữ liệu người dùng tự định nghĩa (Object)

```java
import java.util.HashSet;
import java.util.Set;

class Student04 {
    private int id;
    private String name;

    public Student04(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + "]";
    }
}
```

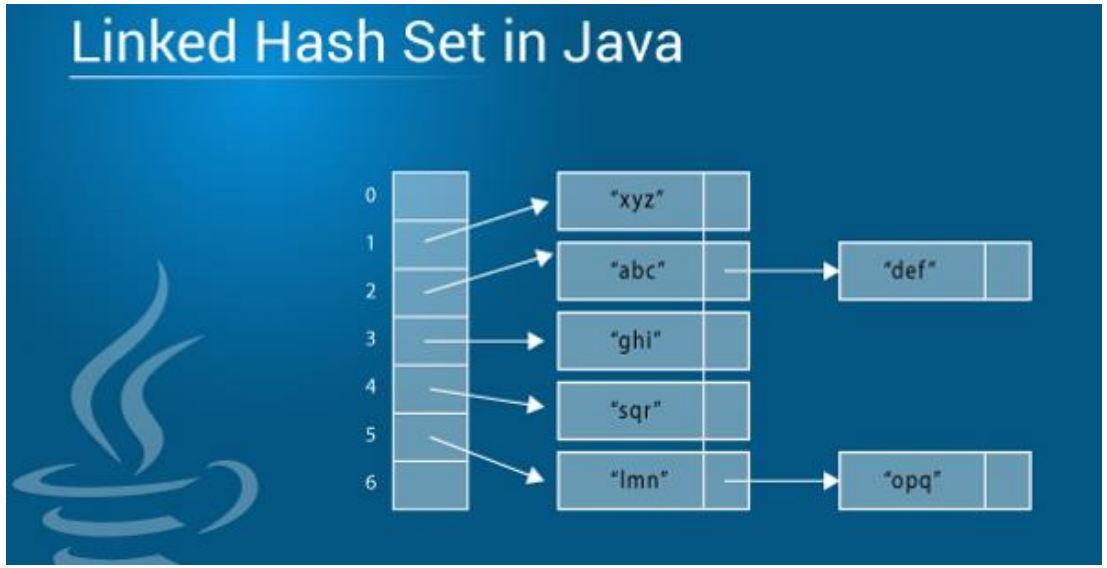
```
Student [id=5, name=myname5]
Student [id=3, name=myname3]
Student [id=4, name=myname4]
Student [id=2, name=myname2]
Student [id=1, name=myname1]
```

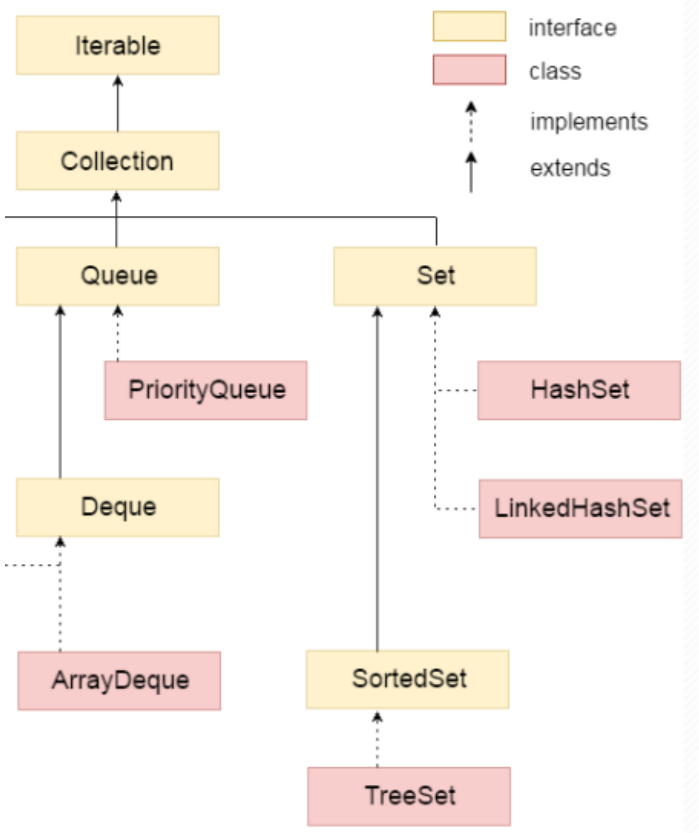
```java
public class HashSetExample02 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create list
        Set<Student04> students = new HashSet<>();
        Student04 student1 = new Student04(1, "myname1");
        Student04 student2 = new Student04(2, "myname2");
        Student04 student3 = new Student04(3, "myname3");
        Student04 student4 = new Student04(4, "myname4");
        Student04 student5 = new Student04(5, "myname5");
        students.add(student1);
        students.add(student2);
        students.add(student3);
        students.add(student4);
        students.add(student5);
        students.add(student2);
        students.add(student3);

        // Show set student
        for (Student04 student : students) {
            System.out.println(student);
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

Giới thiệu lớp LinkedHashSet

- Lớp **LinkedHashSet** trong Java kế thừa **HashSet** và triển khai **Set Interface**.
- **LinkedHashSet** tạo một Collection và sử dụng một **LinkedList** để lưu trữ các phần tử theo thứ tự được chèn.



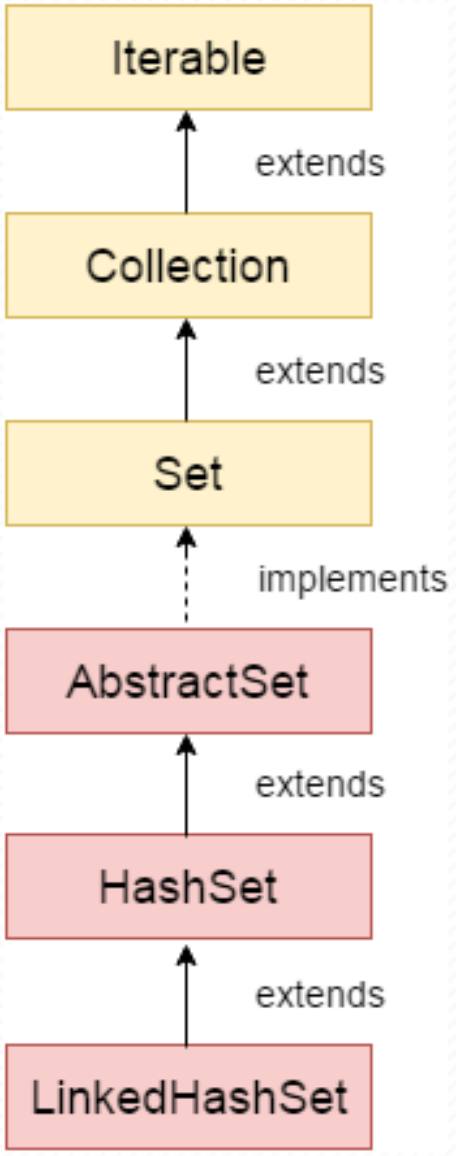


Set, HashSet, LinkedHashSet, TreeSet

Giới thiệu lớp LinkedHashSet

❑ Lớp **java.util.LinkedHashSet** được định nghĩa như sau:

```
public class LinkedHashSet<E> extends HashSet<E>
        implements Set<E>, Cloneable, java.io.Serializable {
    private static final long serialVersionUID = -5024744406713321676L;

    public LinkedHashSet(int initialCapacity, float loadFactor) {
        super(initialCapacity, loadFactor, true);
    }
}
```

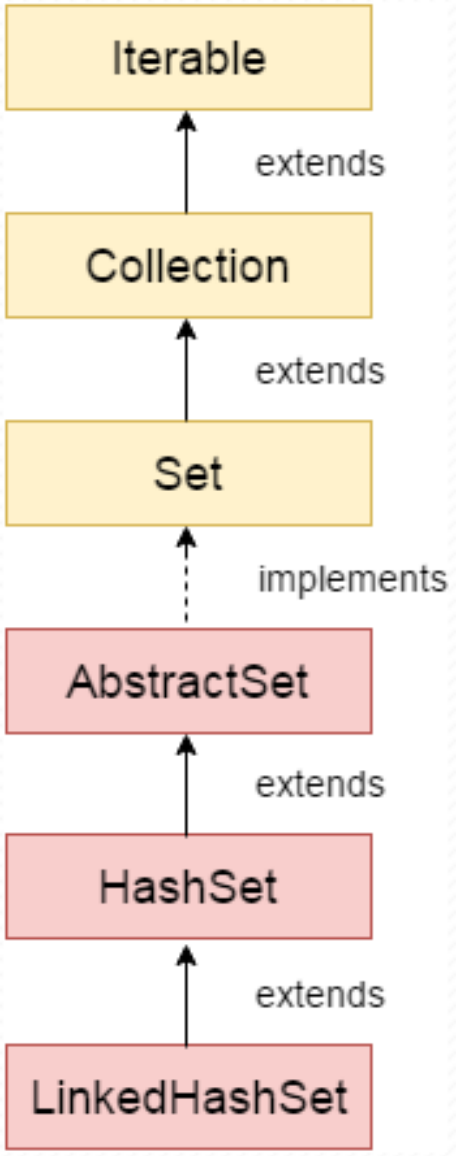


Set, HashSet, LinkedHashSet, TreeSet

Giới thiệu lớp LinkedHashSet

- Bên cạch đó, lớp **java.util.HashSet** được định nghĩa như sau:

```
public class HashSet<E>
   extends AbstractSet<E>
   implements Set<E>, Cloneable, java.io.Serializable {
   static final long serialVersionUID = -5024744406713321676L;

   private transient HashMap<E,Object> map;

   private static final Object PRESENT = new Object();

   HashSet(int initialCapacity, float loadFactor, boolean dummy) {
      map = new LinkedHashMap<>(initialCapacity, loadFactor);
   }
}
```



Set, HashSet, LinkedHashSet, TreeSet

Các phương thức khởi tạo (constructor) của lớp LinkedHashSet

Phương thức	Mô tả
LinkedHashSet()	khởi tạo một danh sách mảng trống
LinkedHashSet(Collection c)	khởi tạo một danh sách mảng với các phần tử của collection c

Các phương thức (method) của lớp LinkedHashSet

- Tương tự các phương thức đã được giới thiệu ở bài viết về **Set Interface**

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng LinkedHashSet với kiểu dữ liệu cơ bản (Wrapper)

```
public class LinkedHashSetExample01 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create set
        Set<String> set = new LinkedHashSet<>();
        set.add("Item01");
        set.add("Item02");
        set.add("Item03");
        set.add("Item04");
        set.add("Item05");
        set.add("Item02");
        set.add("Item03");

        // Show set through for-each
        for (String item : set) {
            System.out.print(item + " ");
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ LinkedHashSet với kiểu dữ liệu người dùng tự định nghĩa (Object)

```java
import java.util.LinkedHashSet;
import java.util.Set;

class Student05 {
    private int id;
    private String name;

    public Student05(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + "]";
    }
}
```

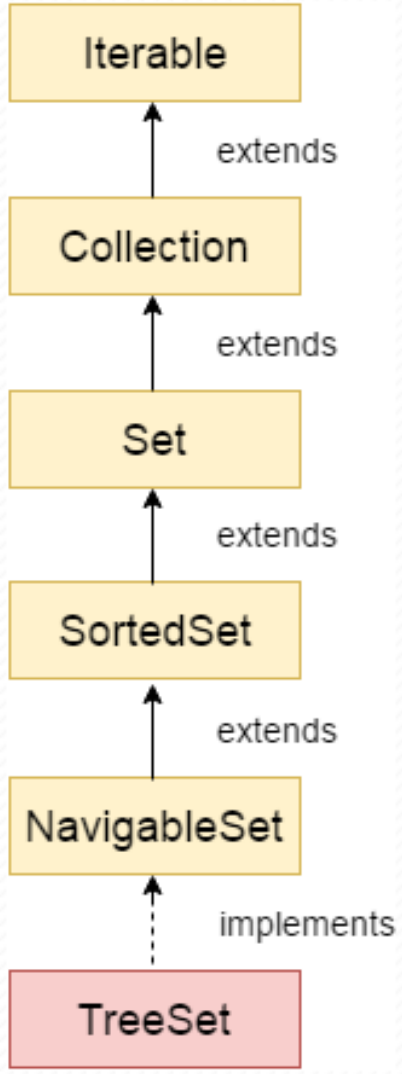
```
Student [id=1, name=myname1]
Student [id=3, name=myname3]
Student [id=2, name=myname2]
Student [id=5, name=myname5]
Student [id=4, name=myname4]
```

```java
public class LinkedHashSetExample02 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create list
        Set<Student05> students = new LinkedHashSet<>();
        Student05 student1 = new Student05(1, "myname1");
        Student05 student2 = new Student05(2, "myname2");
        Student05 student3 = new Student05(3, "myname3");
        Student05 student4 = new Student05(4, "myname4");
        Student05 student5 = new Student05(5, "myname5");
        students.add(student1);
        students.add(student3);
        students.add(student2);
        students.add(student5);
        students.add(student4);
        students.add(student2);
        students.add(student3);

        // Show set student
        for (Student05 student : students) {
            System.out.println(student);
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

Giới thiệu lớp TreeSet

- Lớp **TreeSet** trong Java cài đặt (implements) **Set Interface**.
- **TreeSet** sử dụng một cây (tree) để lưu trữ các phần tử.
- Kế thừa (extends) lớp **AbstractSet** và cài đặt **NavigableSet** interface
- Các đối tượng của lớp **TreeSet** được lưu trữ theo thứ tự tăng dần.

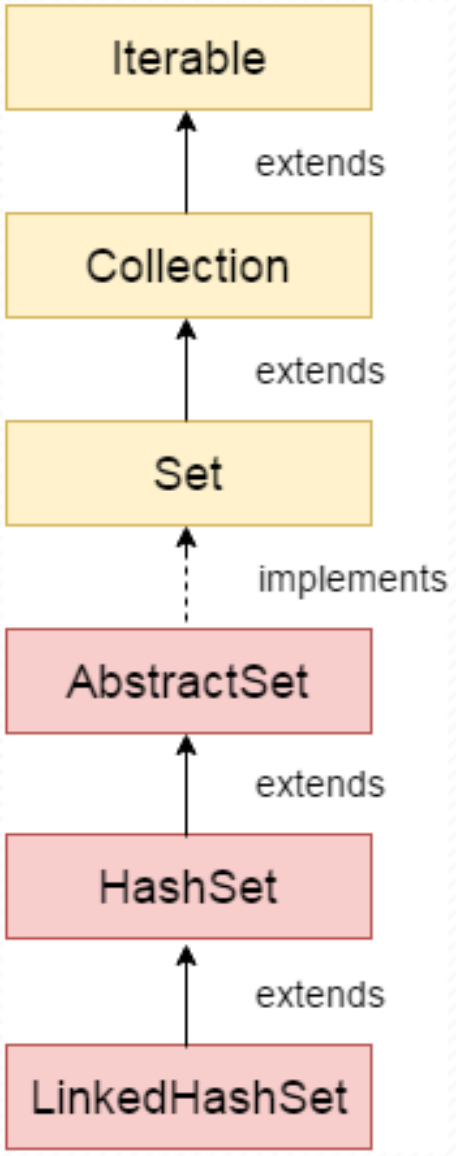


Set, HashSet, LinkedHashSet, TreeSet

Giới thiệu lớp TreeSet

- Lớp **java.util.TreeSet** được định nghĩa như sau:

```
public class TreeSet<E> extends AbstractSet<E>
        implements NavigableSet<E>, Cloneable, java.io.Serializable {
    private transient NavigableMap<E,Object> m;

    private static final Object PRESENT = new Object();

    TreeSet(NavigableMap<E,Object> m) {
        this.m = m;
    }
    public TreeSet() {
        this(new TreeMap<E,Object>());
    }
}
```



Set, HashSet, LinkedHashSet, TreeSet

Các phương thức khởi tạo (constructor) của lớp TreeSet

Phương thức	Mô tả
TreeSet()	khởi tạo một danh sách mảng trống
TreeSet(Collection c)	khởi tạo một danh sách mảng với các phần tử của collection c
TreeSet(Comparator comparator)	khởi tạo một tập hợp rỗng mà các phần tử được xếp thứ tự theo bộ so sánh được xác định bởi comparator.

Các phương thức (method) của lớp TreeSet

- Tương tự các phương thức đã được giới thiệu ở bài viết về **Set Interface**

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng TreeSet với kiểu dữ liệu cơ bản (Wrapper)

```
public class TreeSetExample01 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create set
        Set<String> set = new TreeSet<>();
        set.add("Item01");
        set.add("Item02");
        set.add("Item03");
        set.add("Item04");
        set.add("Item05");
        set.add("Item02");
        set.add("Item03");

        // Show set through for-each
        for (String item : set) {
            System.out.print(item + " ");
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng TreeSet với kiểu dữ liệu người dùng tự định nghĩa (Object)

```java
import java.util.Set;
import java.util.TreeSet;

class Student06 {
    private int id;
    private String name;

    public Student06(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name="
                                + name + "]";
    }

    public String getName() {
        return name;
    }
}
```

Đối với kiểu **Object** nếu bạn không cung cấp bộ so sánh (**Comaparator**) cho **TreeSet** thì bạn sẽ gặp lỗi như dưới.
=> Sao giờ???

```
Exception in thread "main" java.lang.ClassCastException: class com.vu.Student06
cannot be cast to class java.lang.Comparable (com.vu.Student06 is in unnamed module
of loader 'app'; java.lang.Comparable is in module java.base of loader 'bootstrap')
at java.base/java.util.TreeMap.compare(TreeMap.java:1563)
at java.base/java.util.TreeMap.addEntryToEmptyMap(TreeMap.java:768)
at java.base/java.util.TreeMap.put(TreeMap.java:777)
at java.base/java.util.TreeMap.put(TreeMap.java:534)
at java.base/java.util.TreeSet.add(TreeSet.java:255)
at com.vu.TreeSetExample02.main(TreeSetExample02.java:36)
```

```java
public class TreeSetExample02 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create list with no comparator
        Set<Student06> students = new TreeSet<>();
        Student06 student1 = new Student06(1, "myname1");
        Student06 student2 = new Student06(2, "myname2");
        Student06 student3 = new Student06(3, "myname3");
        Student06 student4 = new Student06(4, "myname4");
        Student06 student5 = new Student06(5, "myname5");
        students.add(student1);
        students.add(student3);
        students.add(student2);
        students.add(student5);
        students.add(student4);
        students.add(student2);
        students.add(student3);

        // Show set student
        for (Student06 student : students) {
            System.out.println(student);
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng TreeSet với kiểu dữ liệu người dùng tự định nghĩa (Object)

```java
import java.util.Set;
import java.util.TreeSet;

class Student06 {
    private int id;
    private String name;

    public Student06(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name="
                                + name + "]";
    }

    public String getName() {
        return name;
    }
}
```

Có **2** cách để cung cấp bộ **Comparator**:

1. Implement **Comparator<T>** và override phương thức **compare(T obj1, T obj2)**

2. Implement **Comparable<T>** và override phương thức **compareTo(T obj)**.

```
Exception in thread "main" java.lang.ClassCastException: class com.vu.Student06
cannot be cast to class java.lang.Comparable (com.vu.Student06 is in unnamed module
of loader 'app'; java.lang.Comparable is in module java.base of loader 'bootstrap')
at java.base/java.util.TreeMap.compare(TreeMap.java:1563)
at java.base/java.util.TreeMap.addEntryToEmptyMap(TreeMap.java:768)
at java.base/java.util.TreeMap.put(TreeMap.java:777)
at java.base/java.util.TreeMap.put(TreeMap.java:534)
at java.base/java.util.TreeSet.add(TreeSet.java:255)
at com.vu.TreeSetExample02.main(TreeSetExample02.java:36)
```

```java
public class TreeSetExample02 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create list with no comparator
        Set<Student06> students = new TreeSet<>();
        Student06 student1 = new Student06(1, "myname1");
        Student06 student2 = new Student06(2, "myname2");
        Student06 student3 = new Student06(3, "myname3");
        Student06 student4 = new Student06(4, "myname4");
        Student06 student5 = new Student06(5, "myname5");
        students.add(student1);
        students.add(student3);
        students.add(student2);
        students.add(student5);
        students.add(student4);
        students.add(student2);
        students.add(student3);

        // Show set student
        for (Student06 student : students) {
            System.out.println(student);
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

Implement Comparator<T>, override phương thức compare(T obj1, T obj2)

```java
public class Student {
    private int id;
    private String name;

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + "]";
    }

    public String getName() {
        return name;
    }
}
```

```java
import java.util.Comparator;

public class StudentComparator implements Comparator<Student> {
    @Override
    public int compare(Student o1, Student o2) {
        // sort student's name by ASC
        return o1.getName().compareTo(o2.getName());
    }
}
```

```java
public class TreeSetExample01 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create list with StudentComparator
        Set<Student> students = new TreeSet<>(new StudentComparator());
        Student student1 = new Student(1, "myname1");
        Student student2 = new Student(2, "myname2");
        Student student3 = new Student(3, "myname3");
        Student student4 = new Student(4, "myname4");
        Student student5 = new Student(5, "myname5");
        students.add(student1);
        students.add(student3);
        students.add(student2);
        students.add(student5);
        students.add(student4);
        students.add(student2);
        students.add(student3);

        // Show set student
        for (Student student : students) {
            System.out.println(student);
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

Implement Comparable<T> và override phương thức compareTo(T obj)

```java
public class Student02 implements Comparable<Student> {
    private int id;
    private String name;

    public Student02(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + "]";
    }

    @Override
    public int compareTo(Student student) {
        // sort student's name by ASC
        return this.getName().compareTo(student.getName());
    }

    public String getName() {
        return name;
    }
}
```

```java
public class TreeSetExample02 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create list
        Set<Student02> students = new TreeSet<>();
        Student02 student1 = new Student02(1, "myname1");
        Student02 student2 = new Student02(2, "myname2");
        Student02 student3 = new Student02(3, "myname3");
        Student02 student4 = new Student02(4, "myname4");
        Student02 student5 = new Student02(5, "myname5");
        students.add(student1);
        students.add(student3);
        students.add(student2);
        students.add(student5);
        students.add(student4);
        students.add(student2);
        students.add(student3);

        // Show set student
        for (Student02 student : students) {
            System.out.println(student);
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

So sánh Comparable và Comparator

Comparable	Comparator
Comparable chỉ cung cấp một cách so sánh duy nhất. Tức là chúng ta chỉ có thể so sánh theo id hoặc name, hoặc age, …	Comparable cung cấp nhiều cách so sánh. Tức là chúng ta có thể sắp xếp dựa trên nhiều yếu tố như id, name, age, …
Comparable làm thay đổi lớp gốc (original class), tức là lớp của đối tượng so sánh phải chỉnh sửa và implement Comparable Interface để cài đặt bộ so sánh.	Comparator không làm thay đổi lớp gốc (original class). Chúng ta có thể tạo một class mới, implement Comparator Interface để cài đặt bộ so sánh.

Set, HashSet, LinkedHashSet, TreeSet

So sánh Comparable và Comparator

Comparable	Comparator
Comparable cung cấp phương thức **compareTo()** để so sánh 2 phần tử.	Comparable cung cấp phương thức **compare()** để so sánh 2 phần tử.
Comparable nằm trong package **java.lang**.	Comparator nằm trong package **java.util**.
Có thể sắp xếp một danh sách bất kỳ bằng phương thức **Collections.sort(List)**.	Có thể sắp xếp một danh sách bất kỳ bằng phương thức **Collections.sort(List, Comparator)**.

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng Comparator để sắp xếp danh sách List

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class SortListExample {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {

    public static void printData(List<Student> students) {
        for (Student student : students) {
            System.out.println(student);
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng Comparator để sắp xếp danh sách List (trong hàm main)

```
// Create list
List<Student> students = new ArrayList<>();
Student student1 = new Student(1, "myname1");
Student student2 = new Student(2, "myname2");
Student student3 = new Student(3, "myname3");
Student student4 = new Student(4, "myname4");
Student student5 = new Student(5, "myname5");
students.add(student1);
students.add(student3);
students.add(student2);
students.add(student5);
students.add(student4);
students.add(student2);

// Show list student
System.out.println("List without sorted: ");
printData(students);
System.out.println("--- ");

System.out.println("List sorted using StudentNameComparator: ");
List<Student> students2 = new ArrayList<>(students);
Collections.sort(students2, new StudentNameComparator());
printData(students2);
System.out.println("--- ");

System.out.println("List sorted using StudentIdComparator: ");
List<Student> students3 = new ArrayList<>(students);
Collections.sort(students3, new StudentIdComparator());
printData(students3);
System.out.println("--- ");
```

```
List without sorted:
Student [id=1, name=myname1]
Student [id=3, name=myname3]
Student [id=2, name=myname2]
Student [id=5, name=myname5]
Student [id=4, name=myname4]
Student [id=2, name=myname2]
---
List sorted using StudentNameComparator:
Student [id=1, name=myname1]
Student [id=2, name=myname2]
Student [id=2, name=myname2]
Student [id=3, name=myname3]
Student [id=4, name=myname4]
Student [id=5, name=myname5]
---
List sorted using StudentIdComparator:
Student [id=5, name=myname5]
Student [id=4, name=myname4]
Student [id=3, name=myname3]
Student [id=2, name=myname2]
Student [id=2, name=myname2]
Student [id=1, name=myname1]
---
```

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng Anonymous Class: Comparable, Comparator để sắp xếp danh sách List (trong hàm main)

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class SortListExample02 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {

    public static void printData(List<Student> students) {
        for (Student student : students) {
            System.out.println(student);
        }
    }
}
```

Set, HashSet, LinkedHashSet, TreeSet

Ví dụ sử dụng Anonymous Class: Comparable, Comparator để sắp xếp danh sách List (trong hàm main)

```
// Create list
List<Student> students = new ArrayList<>();
Student student1 = new Student(1, "myname1");
Student student2 = new Student(2, "myname2");
Student student3 = new Student(3, "myname3");
Student student4 = new Student(4, "myname4");
Student student5 = new Student(5, "myname5");
students.add(student1);
students.add(student3);
students.add(student2);
students.add(student5);
students.add(student4);
students.add(student2);

// Show list student
System.out.println("List without sorted: ");
printData(students);
System.out.println("--- ");
```

```
// Show list student
System.out.println("List without sorted: ");
printData(students);
System.out.println("--- ");

System.out.println("List sorted using Comparable: ");
List<Student> students2 = new ArrayList<>(students);
Collections.sort(students2, new Comparator<Student>() {
    @Override
    public int compare(Student o1, Student o2) {
        // sort student's name by ASC
        return o1.getName().compareTo(o2.getName());
    }
});
printData(students2);
System.out.println("--- ");

System.out.println("List sorted using Comparable: ");
List<Student> students3 = new ArrayList<>(students);
Collections.sort(students3, new Comparator<Student>() {
    @Override
    public int compare(Student o1, Student o2) {
        // sort student's id by DESC
        return o2.getName().compareTo(o1.getName());
    }
});
printData(students3);
System.out.println("--- ");
```

So sánh HashSet, LinkedHashSet, TreeSet

Giống nhau giữa HashSet, LinkedHashSet, TreeSet

- ❑ Cả ba không cho phép các phần tử trùng lặp.
- ❑ Cả ba không được đồng bộ (non-synchronized).
- ❑ Cả ba đều cài đặt interface **Cloneable** và **Serializable**.

So sánh HashSet, LinkedHashSet, TreeSet

Giống nhau giữa HashSet, LinkedHashSet, TreeSet

- Cả ba đều là fail-fast. Bạn sẽ nhận được **ConcurrentModificationException** nếu chúng được sửa đổi sau khi tạo đối tượng Iterator.

```java
public class LinkedHashSetExample03 {
    public static void main(String[] args) {
        // Create list
        Set<Student> students = new LinkedHashSet<>();
        // Add element to list
        Student student1 = new Student(1, "myname1");
        Student student2 = new Student(2, "myname2");
        Student student3 = new Student(3, "myname3");
        students.add(student1);
        students.add(student2);
        students.add(student3);

        // Show list student
        Iterator<Student> it = students.iterator();
        students.remove(student2); // You will get ConcurrentModificationException
                                   // if they are modified after the creation of Iterator object.
        while (it.hasNext()) {
            Student s = (Student) it.next();
            System.out.println(s);
        }
    }
}
```

```java
public class Student {
    private int id;
    private String name;

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", "
                            + "name=" + name + "]";
    }
}
```

```
Exception in thread "main" java.util.ConcurrentModificationException
at java.base/java.util.LinkedHashMap$LinkedHashIterator.nextNode(LinkedHashMap.java:758)
at java.base/java.util.LinkedHashMap$LinkedKeyIterator.next(LinkedHashMap.java:780)
at com.vu.compare.LinkedHashSetExample03.main(LinkedHashSetExample03.java:24)
```

So sánh HashSet, LinkedHashSet, TreeSet

Khác nhau giữa HashSet, LinkedHashSet, TreeSet

	HashSet	LinkedHashSet	TreeSet
Cách thức làm việc?	HashSet sử dụng HashMap nội bộ để lưu trữ các phần tử.	LinkedHashSet sử dụng LinkedHashMap nội bộ để lưu trữ các phần tử.	TreeSet sử dụng TreeMap nội bộ để lưu trữ các phần tử.
Thứ tự của các phần tử (Order of Elements)	HashSet không duy trì bất kỳ thứ tự các phần tử được thêm vào.	LinkedHashSet duy trì thứ tự chèn của các phần tử. Các phần tử được lưu trữ đúng như thứ tự chúng được chèn vào.	TreeSet duy trì thứ tự các phần tử theo bộ so sánh được cung cấp (Comparator). Nếu không có bộ so sánh được cung cấp, các phần tử sẽ được đặt theo thứ tự tăng dần tự nhiên của chúng.

So sánh HashSet, LinkedHashSet, TreeSet

Khác nhau giữa HashSet, LinkedHashSet, TreeSet

	HashSet	LinkedHashSet	TreeSet
Hiệu suất (Performance)	HashSet cho hiệu suất tốt hơn so với LinkedHashSet và TreeSet.	Hiệu suất của LinkedHashSet nằm giữa HashSet và TreeSet. Hiệu suất của nó hầu như tương tự như HashSet. Nhưng hơi chậm hơn vì nó cũng duy trì LinkedList nội bộ để duy trì trình tự chèn các phần tử.	TreeSet cho hiệu suất thấp hơn HashSet và LinkedHashSet vì nó phải sắp xếp các phần tử sau mỗi lần chèn và loại bỏ.

So sánh HashSet, LinkedHashSet, TreeSet

Khác nhau giữa HashSet, LinkedHashSet, TreeSet

	HashSet	LinkedHashSet	TreeSet
Thao tác thêm (Insertion), xóa (Removal), truy xuất (Retrieval) phần tử	HashSet cho hiệu suất là O(1) để chèn, loại bỏ, truy xuất phần tử.	LinkedHashSet cũng cho hiệu suất là O(1) để chèn, loại bỏ và truy xuất phần tử.	TreeSet cho hiệu suất là O(log (n)) cho các thao tác chèn, loại bỏ và truy xuất phần tử.
So sánh các phần tử	HashSet sử dụng các equals(), hashCode() để so sánh các phần tử => loại bỏ các phần tử có thể trùng lặp.	LinkedHashSet cũng sử dụng phương thức equals() và hashCode() để so sánh các phần tử.	TreeSet sử dụng compare() hoặc compareTo() để so sánh các phần tử => loại bỏ các phần tử có thể trùng lặp. Nó không sử dụng các phương thức equals (), hashCode () để so sánh các phần tử.

So sánh HashSet, LinkedHashSet, TreeSet

Khác nhau giữa HashSet, LinkedHashSet, TreeSet

	HashSet	LinkedHashSet	TreeSet
Phần tử Null	HashSet cho phép tối đa một phần tử null.	LinkedHashSet cũng cho phép tối đa một phần tử null.	TreeSet không cho phép chứa phần tử null. Nếu bạn cố gắng để chèn null thành phần TreeSet, nó ném NullPointerException.
Sử dụng bộ nhớ	HashSet đòi hỏi ít bộ nhớ hơn LinkedHashSet và TreeSet vì nó chỉ sử dụng HashMap nội bộ để lưu trữ các phần tử của nó.	LinkedHashSet yêu cầu bộ nhớ nhiều hơn HashSet vì nó cũng duy trì LinkedList cùng với HashMap để lưu trữ các phần tử của nó.	TreeSet cũng yêu cầu bộ nhớ nhiều hơn HashSet vì nó cũng duy trì bộ so sánh để sắp xếp các phần tử cùng với TreeMap.

So sánh HashSet, LinkedHashSet, TreeSet

Khác nhau giữa HashSet, LinkedHashSet, TreeSet

	HashSet	LinkedHashSet	TreeSet
Khi nào sử dụng?	Sử dụng HashSet nếu bạn muốn danh sách không chứa phần tử trùng và không cần duy trì bất kỳ thứ tự các phần tử được chèn vào.	Sử dụng LinkedHashSet nếu bạn muốn danh sách không chứa phần tử trùng và muốn duy trì thứ tự chèn của các phần tử.	Sử dụng TreeSet nếu bạn muốn danh sách không chứa phần tử trùng và muốn sắp xếp các phần tử theo một số so sánh.

Tổng kết nội dung bài học

- Set, HashSet, LinkedHashSet, TreeSet
- So sánh HashSet, LinkedHashSet, TreeSet