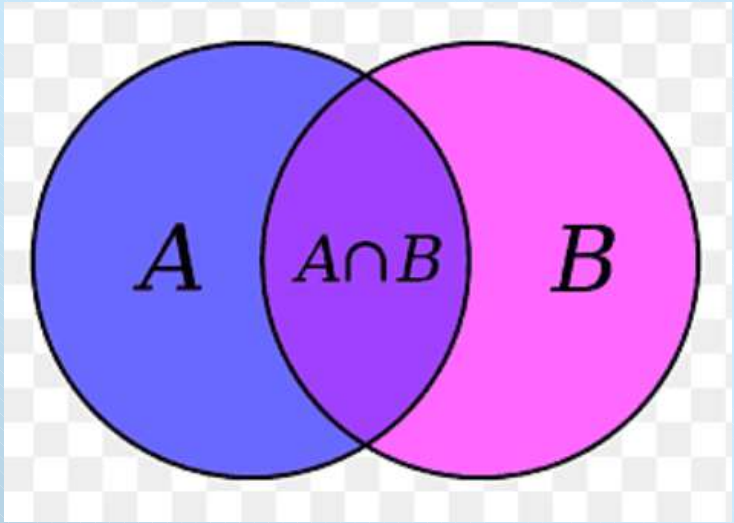
Collections

Set



Objectives

- At the end of this topic, you should be able to
  - Understand the concept of Set.
  - Explain Set and its common operations.
  - Use the concrete implementations of Set in the Java collection framework .

What is a Set?

- A Set is a collection that stores unique elements.
- Elements are not stored in any particular order, unlike a List.
- Sets do not allow duplicates.

Use Cases of Set

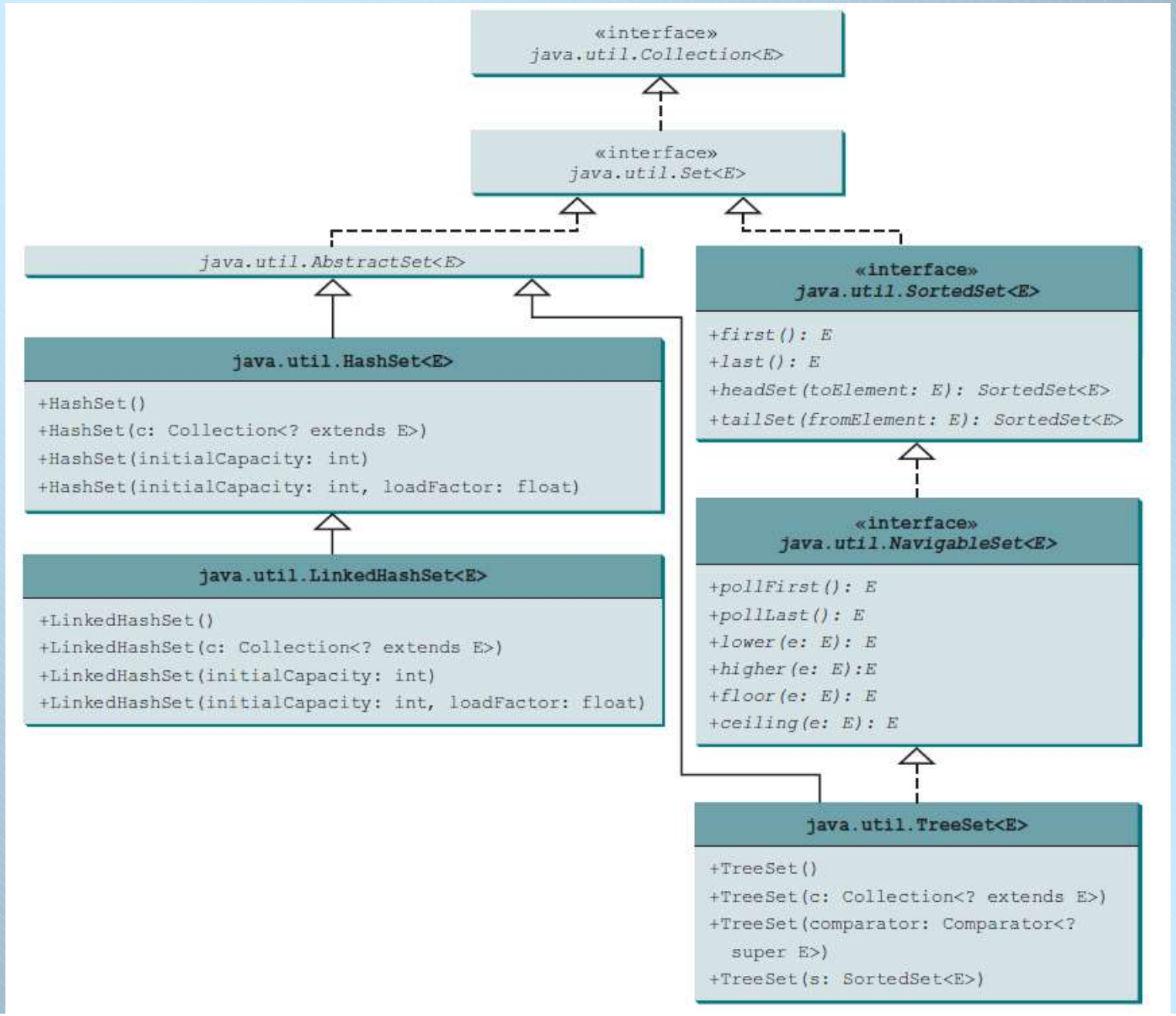
- In a social network, a Set can be used to store the unique set of friends of a user, since each friend should only be added once.
- In a shopping cart, a Set can be used to store the unique set of items added to the cart, since each item should only be added once
- In a game, a Set can be used to store the unique set of players who have joined the game, since each player should only be added once.

Set in Java

- Set is part of the Java Collection Framework and is implemented by several classes.
- The Set interface provides several methods for adding, removing, and accessing elements.
- Since Set is an interface, so it can be implemented by several different classes.
- Set implementations in Java:
  - HashSet, TreeSet, and LinkedHashSet.

Set in Java

- 3 concrete Set classes:



Common Set Implementations in Java

- **HashSet**: Implements Set using a hash table.
- **LinkedHashSet**: Implements Set using a linked list and hash table.
- **TreeSet**: Implements Set using a tree.

HashSet in Java Set

- Stores elements in a hash table.
- Has very fast (constant-time performance) for the basic operations (add, remove, contains) on average.
- Does not maintain any order of elements.

Creating and Initializing a Set in Java

- A Set can be created using the "new" keyword and one of the Set implementation classes.
- A Set can also be initialized using another Collection object.
- Elements can be added to a Set during initialization.

```
Set<String> set = new HashSet<>();          // create a HashSet of strings

Set<Person> setP = new HashSet<>();     // create a HashSet of Person objects

List<String> destList = new ArrayList<>(); destList.add("Tokyo");
Set<String> destSet = new HashSet<>(destList);      // create a HashSet from a List
```

Adding Elements to a Set in Java

- Elements can be added to a Set using the "add" method.
- If the Set already contains the element, the "add" method has no effect.

```
destSet.add("Paris");
destSet.add("New York");
destSet.add("Kuala Lumpur");
destSet.add("Paris");

 // print the size of the HashSet
 System.out.println("Size of set: " + destSet.size());

 // iterate over the elements in the HashSet
 for (String element : destSet) {
        System.out.println(element);
 }
```

Output:

```
Size of set: 4
New York
Tokyo
Kuala Lumpur
Paris
```

Adding Elements to a Set in Java

- When an element is added to a HashSet, its hash code is computed using the hashCode() method
- To check if an element already exists in hashtable, its hash code is first compared to hash codes of all elements. If the hash codes do not match, the element is not in the bucket and added to hashtable.
- If the hash codes match, the equals() method is used to determine if the element is equal to any elements. If the equals() method returns true, the element is considered a duplicate and is not added to hashtable.

Adding Elements to a Set in Java

- For a user-defined class, the hashCode() and equals() MUST be overridden
  - setP.add(new Person("John Doe", 35));
  - setP.add(new Person("John Doe", 35)); //both can be added
- Overriding the hashCode() method:
- Example (for a Person class):
  - Use the Objects hash() method to compute hashcode based on the object attributes e.g.name, age

```
@Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
```

Adding Elements to a Set in Java

- Overriding the equals() method:
- Example (for a Person class):

```
@Override
 public boolean equals(Object o) {
     if (this == o) return true;
     if (o == null || getClass() != o.getClass())
         return false;
     Person person = (Person) o;
     return age == person.age &&
             name.equalsIgnoreCase(person.name);
 }
```

Removing Elements from a Set in Java

- Elements can be removed from a Set using the "remove" method.
- If the Set does not contain the element, the "remove" method has no effect.
- The "clear" method can be used to remove all elements from a Set.

```
destSet.remove("Bangkok");  //removing from set of String
Person alice = new Person("Alice", 25);
setP.remove(alice);              //removing from set of Person

destSet.clear();                 //removing all elements in Set
```

Accessing Elements in a Set in Java

- The "contains" method can be used to check if a Set contains a specific element.
- The "size" method can be used to get the number of elements in a Set.
- The "isEmpty" method can be used to check if a Set is empty.

Set Operations in Java (Union, Intersection, Difference)

- The "addAll" method can be used to perform a union operation between two Sets.
- The "retainAll" method can be used to perform an intersection operation between two Sets.
- The "removeAll" method can be used to perform a difference operation between two Sets.

LinkedHashSet in Java Set

- Stores elements in a hash table and a linked list.
- Maintains the order in which elements were inserted.
- Slower than HashSet for basic operations, but faster for iterating through elements.

```
//Create a LinkedHashSet
Set<String> set = new
LinkedHashSet<>();
// Add strings to the set
set.add("Paris");
set.add("New York");
set.add("Kuala Lumpur");
set.add("Paris");
// iterate over the elements in the
HashSet
for (String element : set) {
      System.out.println(element);
}
```

Output:

Paris
New York
Kuala Lumpur

TreeSet in Java Set

- Stores elements in a tree.
- Automatically maintains elements in sorted order.
- Slower than HashSet for basic operations, but has useful operations related to elements order.
- The comparing of elements must be specified by:
  - implementing the Comparable interface for every elements

    OR
  - passing an object that implements the Comparator object to the TreeSet contructor

Implementing the Comparable interface

- Override the abstract compareTo() method
- Example, if Person to be sorted according to name:

```
class Person implements Comparable<Person>{
    private String name;
    private int age;
       :
       :
    @Override
    public int compareTo(Person o) {
        return this.name.compareTo(o.name);
    }
}
```

Implementing the Comparator interface

- Create a Comparator object that overrides the abstract compare() method
- Example, if Person to be sorted according to age:

```
class PersonComparator implements Comparator<Person>{
    @Override
    public int compare(Person o1, Person o2) {
        if (o1.getAge() > o2.getAge()) {
            return 1;
        } else if (o1.getAge() < o2.getAge()) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

Implementing the Comparator interface

- Pass the Comparator object to the TreeSet constructor:

```
PersonComparator comp = new PersonComparator();
TreeSet<Person> sortSet = new TreeSet<>(comp);
```

TreeSet Methods

- Implemented methods from SortedSet interface:
  - headSet(): Returns a view of the set that contains all elements less than the specified element
  - tailSet(): Returns a view of the set that contains all elements greater than or equal to the specified element
- Implemented methods from NavigableSet interface:
  - ceiling(): Finds the least element in the set that is greater than or equal to the specified element.
  - floor(): Finds the greatest element in the set that is less than or equal to the specified element.
  - higher(): Finds the least element in the set that is greater than the specified element.
  - lower(): Finds the greatest element in the set that is less than the specified element.

Conclusion

- Choosing the right Set implementation depends on the specific use case.
- HashSet is the most commonly used Set implementation in Java.
- LinkedHashSet is a good choice when the order of elements is important.
- TreeSet is a good choice when elements need to be sorted.