Java

30-dars

Collections framework

Javada alohida Collections framework JDK 1.2 dan qo'shilgan va u o'z ichiga collection classlari hamda interfacelarini oladi.

2 ta interfacelar: Collection (java.util.Collection) va Map (java.util.Map) lar collection frameworkning asosiy interfacelari hisoblanadi.

Collections framework quyidagi afzalliklarga ega:

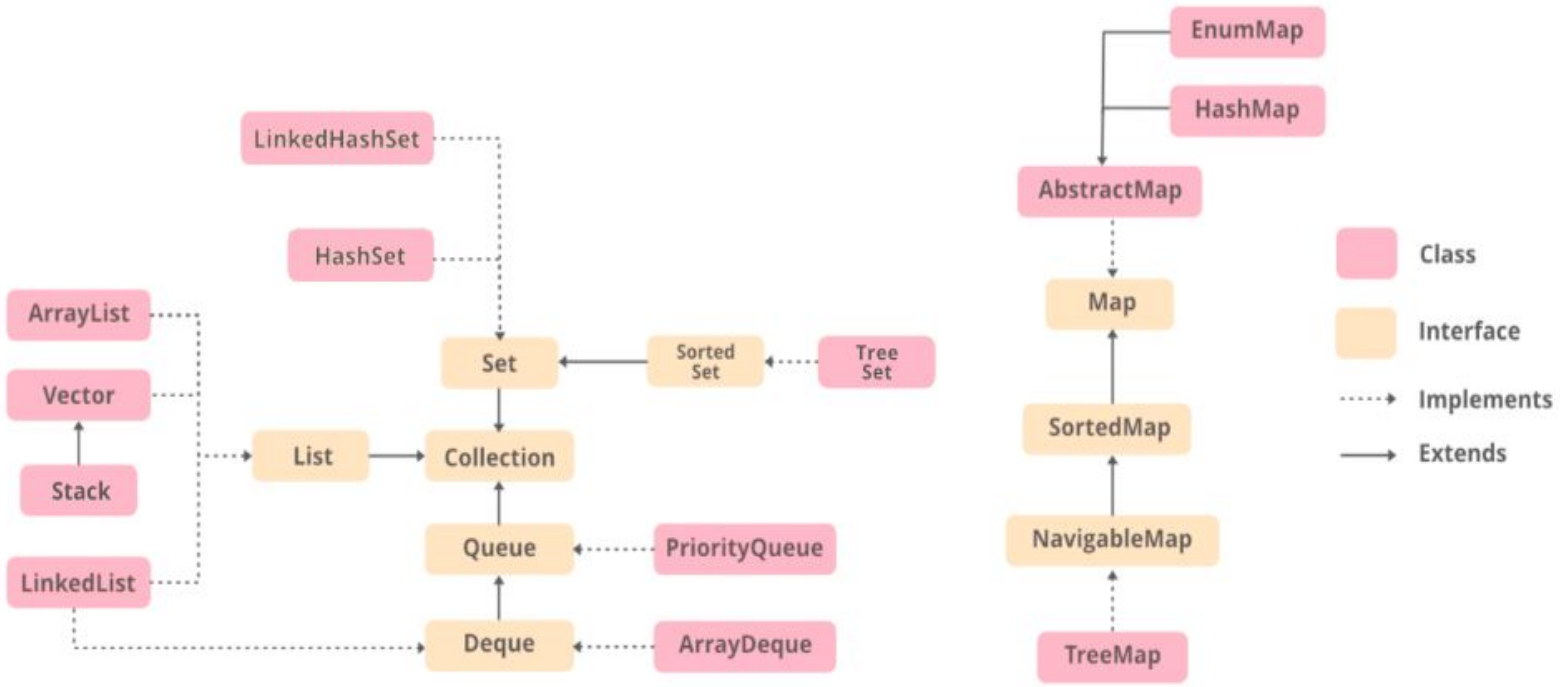
- **Consistent API:** The API has a basic set of interfaces like *Collection*, *Set*, *List*, or *Map*, all the classes (ArrayList, LinkedList, Vector, etc) that implement these interfaces have *some* common set of methods.

- **Reduces programming effort:** A programmer doesn't have to worry about the design of the Collection but rather he can focus on its best use in his program. Therefore, the basic concept of Object-oriented programming (i.e.) abstraction has been successfully implemented.

- **Increases program speed and quality:** Increases performance by providing high-performance implementations of useful data structures and algorithms because in this case, the programmer need not think of the best implementation of a specific data structure. He can simply use the best implementation to drastically boost the performance of his algorithm/program.

Hierarchy of the Collection Framework

Collections framework quyidagi ierarxiyaga ega va bu classlar hamda interfacelar java.util packageda joylashgan.



Collection interface methodlari

Method	Description
add(Object)	This method is used to add an object to the collection.
addAll(Collection c)	This method adds all the elements in the given collection to this collection.
clear()	This method removes all of the elements from this collection.
contains(Object o)	This method returns true if the collection contains the specified element.
containsAll(Collection c)	This method returns true if the collection contains all of the elements in the given collection.
equals(Object o)	This method compares the specified object with this collection for equality.
hashCode()	This method is used to return the hash code value for this collection.
isEmpty()	This method returns true if this collection contains no elements.
iterator()	This method returns an iterator over the elements in this collection.
max()	This method is used to return the maximum value present in the collection.
parallelStream()	This method returns a parallel Stream with this collection as its source.
remove(Object o)	This method is used to remove the given object from the collection. If there are duplicate values, then this method removes the first occurrence of the object.
removeAll(Collection c)	This method is used to remove all the objects mentioned in the given collection from the collection.
removeIf(Predicate filter)	This method is used to remove all the elements of this collection that satisfy the given predicate.
retainAll(Collection c)	This method is used to retain only the elements in this collection that are contained in the specified collection.
size()	This method is used to return the number of elements in the collection.
spliterator()	This method is used to create a Spliterator over the elements in this collection.
stream()	This method is used to return a sequential Stream with this collection as its source.
toArray()	This method is used to return an array containing all of the elements in this collection.

```java
public class Theatre {
    private final String theatreName;
    private List<Seat> seats = new ArrayList<>();

    public Theatre(String theatreName, int numRows, int seatsPerRow) {
        this.theatreName = theatreName;

        int lastRow = 'A' + (numRows -1);
        for (char row = 'A'; row <= lastRow; row++) {
            for(int seatNum = 1; seatNum <= seatsPerRow; seatNum++) {
                Seat seat = new Seat(row + String.format("%02d", seatNum));
                seats.add(seat);
            }
        }
    }

    public String getTheatreName() {
        return theatreName;
    }

    public boolean reserveSeat(String seatNumber) {
        Seat requestedSeat = null;
        for(Seat seat : seats) {
            if(seat.getSeatNumber().equals(seatNumber)) {
                requestedSeat = seat;
                break;
            }
        }

        if(requestedSeat == null) {
            System.out.println("There is no seat " + seatNumber);
            return false;
        }

        return requestedSeat.reserve();
    }
```

```java
public void getSeats() {
    for(Seat seat : seats) {
        System.out.println(seat.getSeatNumber()) ;
    }
}

private class Seat {
    private final String seatNumber;
    private boolean reserved = false;

    public Seat(String seatNumber) {
        this.seatNumber = seatNumber;
    }

    public boolean reserve() {
        if(!this.reserved) {
            this.reserved = true;
            System.out.println("Seat " + seatNumber + " reserved");
            return true;
        } else {
            return false;
        }
    }

    public boolean cancel() {
        if(this.reserved) {
            this.reserved = false;
            System.out.println("Reservation of seat " + seatNumber + " cancelled");
            return true;
        } else {
            return false;
        }
    }

    public String getSeatNumber() {
        return seatNumber;
    }
}
```

```java
public class Main {

    public static void main(String[] args) {
        Theatre theatre = new Theatre("Olympian", 8, 12);
//          theatre.getSeats();
        if(theatre.reserveSeat("H11")) {
            System.out.println("Please pay");
        } else {
            System.out.println("Sorry, seat is taken");
        }
        if(theatre.reserveSeat("H11")) {
            System.out.println("Please pay");
        } else {
            System.out.println("Sorry, seat is taken");
        }
    }
}
```

```java
public class Theatre {

    private final String theatreName;
    private List<Seat> seats = new ArrayList<>();

    public Theatre(String theatreName, int nuRows, int seatsPerRow) {
        this.theatreName = theatreName;
        int lastRow = 'A' + (nuRows - 1);
        for (char row = 'A'; row <= lastRow; row++) {
            for (int seatNum = 1; seatNum <= seatsPerRow; seatNum++) {
                Seat seat =new Seat(row + String.format("%02d", seatNum));
                seats.add(seat);
            }
        }
    }

    public String getTheatreName() {
        return theatreName;
    }

    public void getSeats() {
        for (Seat seat : seats) {
            System.out.println(seat.getSeatNumber());
        }
    }

    public boolean reserveSeat(String seatNumber) {

        Seat requestedSeat =new Seat(seatNumber);
        var found = Collections.binarySearch(this.seats, requestedSeat, null);

        if (found < 0) {
            System.out.println("There is no seat " + seatNumber);
            return false;
        }

        return this.seats.get(found).reserve();
    }
```

```java
private class Seat implements Comparable<Seat> {

        private final String seatNumber;
        private boolean reserved = false;

        public Seat(String seatNumber) {
            this.seatNumber = seatNumber;
        }

        public String getSeatNumber() {
            return seatNumber;
        }

        public boolean reserve() {
            if (!this.reserved) {
                this.reserved = true;
                System.out.println("Seat " + seatNumber + " reserved");
                return true;
            } else {
                return false;
            }
        }

        public boolean cancel() {
            if (this.reserved) {
                this.reserved = false;
                System.out.println("Reservation of seat " + seatNumber + " cancelled");
                return true;
            } else {
                return false;
            }
        }

        @Override
        public int compareTo(Seat seat) {
            return this.seatNumber.compareToIgnoreCase(seat.seatNumber);
        }
    }

}
```