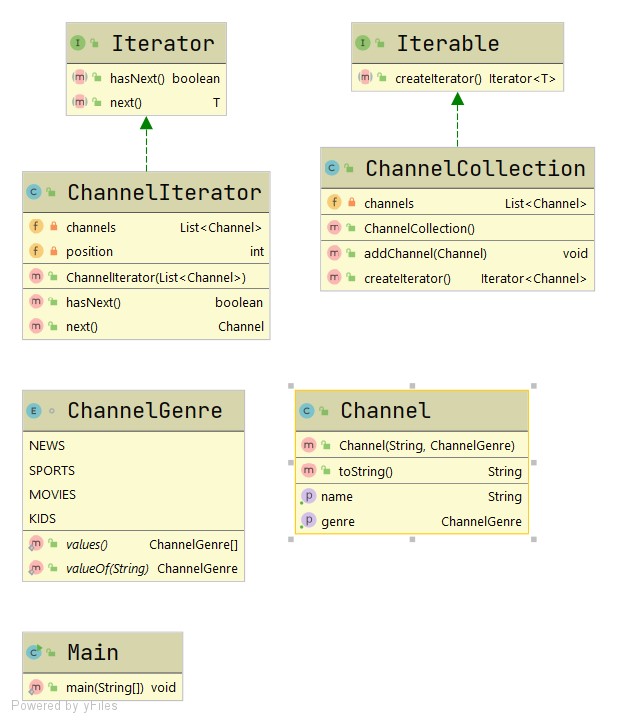
**Mẫu Iterator:**

**Giới thiệu:**

Cung cấp một cách thống nhất để duyệt qua các phần tử của một tập hợp mà không tiết lộ chi tiết về cấu trúc bên trong của tập hợp đó.

**Sau đây là ví dụ về mẫu Iterator:**



Đây là ví dụ về cách triển khai mẫu Iterator trong Java để duyệt qua một danh sách các kênh truyền hình.

**Cấu trúc:**

1. Interface Iterator:

Interface Iterator định nghĩa các phương thức cốt lõi của một iterator:

hasNext(): Kiểm tra xem còn phần tử nào trong tập hợp hay không. Trả về true nếu còn ít nhất một phần tử, false nếu không còn.

next(): Trả về phần tử tiếp theo trong tập hợp. Nếu không còn phần tử nào, nó sẽ ném ra một ngoại lệ NoSuchElementException.

public interface Iterator<T> {

boolean hasNext();

T next();

}

1. Enum ChannelGenre:

Enum ChannelGenre đại diện cho các thể loại khác nhau mà một kênh có thể thuộc vào (Tin tức, Thể thao, Phim, Trẻ em).

public enum *ChannelGenre* {

***NEWS***, ***SPORTS***, ***MOVIES***, ***KIDS***

}

1. Class Channel:

Lớp Channel đại diện cho một kênh với tên và thể loại. Nó bao gồm getters cho cả hai thuộc tính.

public class Channel {

private String name;

private *ChannelGenre* genre;

public Channel(String name, *ChannelGenre* genre) {

this.name = name;

this.genre = genre;

}

public String getName() {

return name;

}

public *ChannelGenre* getGenre() {

return genre;

}

*@Override*

public String toString() {

return "Channel{" +

"name='" + name + '\'' +

", genre=" + genre +

'}';

}

}

1. Class ChannelCollection:

Lớp ChannelCollection đại diện cho một tập hợp các kênh. Nó lưu trữ danh sách các kênh và cung cấp các phương thức để thêm kênh và tạo đối tượng iterator để duyệt qua danh sách kênh.

public class ChannelCollection {

private List<Channel> channels;

public ChannelCollection() {

this.channels = new ArrayList<>();

}

public void addChannel(Channel channel) {

channels.add(channel);

}

public Iterator<Channel> createIterator() {

return new ChannelIterator(channels);

}

}

1. Class ChannelIterator:

Lớp ChannelIterator triển khai interface Iterator cụ thể để lặp qua đối tượng ChannelCollection. Nó duy trì một biến position để theo dõi kênh hiện tại và cung cấp các phương thức để kiểm tra phần tử tiếp theo và lấy phần tử tiếp theo trong danh sách kênh.

public class ChannelIterator implements Iterator<Channel> {

private List<Channel> channels;

private int position = 0;

public ChannelIterator(List<Channel> channels) {

this.channels = channels;

}

*@Override*

public boolean hasNext() {

return position < channels.size();

}

*@Override*

public Channel next() {

if (!hasNext()) {

throw new NoSuchElementException();

}

Channel channel = channels.get(position);

position++;

return channel;

}

}

1. Class Main:

Lớp Main thể hiện cách sử dụng các lớp ChannelCollection và ChannelIterator. Nó tạo một tập hợp kênh, thêm kênh vào đó và sau đó lấy iterator để lặp qua các kênh. Vòng lặp lặp lại khi còn nhiều kênh (hasNext()) và in từng kênh bằng phương thức next().

public class Main {

public static void main(String[] args) {

ChannelCollection channelCollection = new ChannelCollection();

channelCollection.addChannel(new Channel("BBC News", *ChannelGenre*.***NEWS***));

channelCollection.addChannel(new Channel("ESPN", *ChannelGenre*.***SPORTS***));

channelCollection.addChannel(new Channel("Netflix", *ChannelGenre*.***MOVIES***));

channelCollection.addChannel(new Channel("Disney Channel", *ChannelGenre*.***KIDS***));

Iterator<Channel> iterator = channelCollection.createIterator();

while (iterator.hasNext()) {

Channel channel = iterator.next();

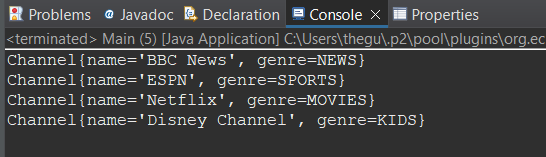
System.***out***.println(channel);

}

}

}

**Kết quả và thử nghiệm:**



Đây là ví dụ cho mẫu Iterator. Lớp ChannelIterator cung cấp một cách để duyệt qua các đối tượng ChannelCollection mà không tiết lộ cách triển khai bên trong của tập hợp kênh. Cách tiếp cận này mang lại sự kết hợp lỏng lẻo giữa tập hợp và mã lặp qua nó.

**Lợi ích:**

Giảm thiểu coupling: Mẫu Iterator giúp giảm thiểu coupling giữa lớp aggregate và lớp client. Lớp client không cần phải biết cách lớp aggregate lưu trữ các phần tử để duyệt qua chúng.

Tăng tính linh hoạt: Mẫu Iterator cho phép bạn dễ dàng thay đổi cách duyệt qua các phần tử của một collection. Bạn có thể tạo ra các iterator khác nhau để duyệt qua các phần tử theo các tiêu chí khác nhau.

Giúp sử dụng lại mã: Mẫu Iterator giúp bạn dễ dàng sử dụng lại mã duyệt qua các phần tử của một collection. Bạn có thể sử dụng cùng một mã duyệt qua các loại collection khác nhau miễn là chúng có cùng interface Iterator.