

# Stream API

## 1. Lambda Expression

### 1.1. Lambda Expression – Trường hợp không có tham số

```
@FunctionalInterface
interface Drawable{
    public void draw();
}

public class LambdaExpressionExample2 {
    public static void main(String[] args) {
        int width=10;

        Drawable d2= ()->{
            System.out.println("Drawing " + width);
        };
        d2.draw();
    }
}
```

### 1.2. Lambda Expression - Có một tham số

```
interface Sayable{
    public String say(String name);
}

public class LambdaExpressionExample4{
    public static void main(String[] args) {
        // Cách 1
        Sayable s1=(name)->{
            return "Hello, "+name;
        };
        System.out.println(s1.say("Sonoo"));

        // Cách 2
        Sayable s2= name ->{
            return "Hello, "+name;
        };
        System.out.println(s2.say("Sonoo"));
    }
}
```

### 1.3. Lambda Expression – Có nhiều tham số

---

```
interface Addable{
    int add(int a,int b);
}

public class LambdaExpressionExample5{
    public static void main(String[] args) {
        // Cách 1
        Addable ad1=(a,b)->(a+b);
        System.out.println(ad1.add(10,20));
        //Cách 2
        Addable ad2=(int a,int b)->(a+b);
        System.out.println(ad2.add(100,200));
    }
}
```

### 1.4. Lambda Expression – không sử dụng return (sử dụng return)

---

```
interface Addable{
    int add(int a,int b);
}

public class LambdaExpressionExample6 {
    public static void main(String[] args) {

        // Cách 1: không cần return
        Addable ad1=(a,b)->(a+b);
        System.out.println(ad1.add(10,20));

        // Cách 2: sử dụng return
        Addable ad2=(int a,int b)->{
            return (a+b);
        };
        System.out.println(ad2.add(100,200));
    }
}
```

### 1.5. Lambda Expression – Kết hợp với Foreach Loop

```
public static void main(String[] args) {  
  
    List<String> list=new ArrayList<String>();  
    list.add("ankit");  
    list.add("mayank");  
    list.add("irfan");  
    list.add("jai");  
  
    list.forEach(  
        ( n )-> System.out.println(n)  
    );  
}
```

### 1.6. Lambda Expression - Multiple Statements

```
@FunctionalInterface  
interface Sayable{  
    String say(String message);  
}  
  
public class LambdaExpressionExample8{  
    public static void main(String[] args) {  
        Sayable person = (message)-> {  
            String str1 = "I would like to say, ";  
            String str2 = str1 + message;  
            return str2;  
        };  
        System.out.println(person.say("time is precious."));  
    }  
}
```

## 1.7. Lambda Expression – Comparator

```
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        super();
        this.id = id;
        this.name = name;
        this.price = price;
    }
}

public class LambdaExpressionComparator{
    public static void main(String[] args) {

        List<Product> list=new ArrayList<Product>();
        list.add(new Product(1,"HP Laptop",25000f));
        list.add(new Product(3,"Keyboard",300f));
        list.add(new Product(2,"Dell Mouse",150f));

        System.out.println("Sorting on the basis of name...");
        Collections.sort( list, (p1,p2)->{
            return p1.name.compareTo(p2.name);
        });

        for(Product p:list){
            System.out.println(p.id+" "+p.name+" "+p.price);
        }
    }
}
```

## 2. Tham chiếu phương thức

## 2.1. Tham chiếu phương thức tĩnh

```
interface Sayable{
    void say();
}

public class MethodReference {
    public static void saySomething(){
        System.out.println("Hello, this is static method.");
    }
    public static void main(String[] args) {

        Sayable sayable = MethodReference::saySomething;

        // Calling interface method
        sayable.say();
    }
}
```

## 2.2. Tham chiếu phương thức instance

```
interface Sayable{
    void say();
}

public class InstanceMethodReference {
    public void saySomething(){
        System.out.println("Hello, this is non-static method.");
    }
    public static void main(String[] args) {
        //Cách 1
        InstanceMethodReference methodReference = new InstanceMethodReference();

        Sayable sayable = methodReference::saySomething;

        sayable.say();

        //Cách 2
        Sayable sayable2 = new InstanceMethodReference()::saySomething;
        // Calling interface method
        sayable2.say();
    }
}
```

### 2.3. Tham chiếu **Constructor**

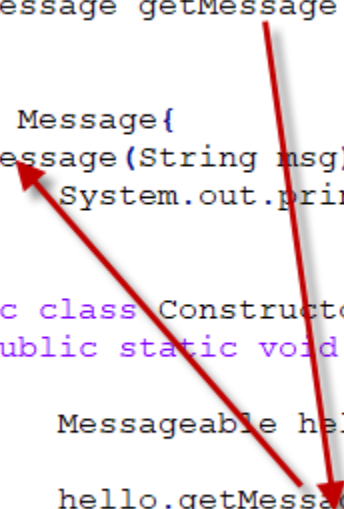
```
interface Messageable{
    Message getMessage(String msg);
}

class Message{
    Message(String msg){
        System.out.print(msg);
    }
}

public class ConstructorReference {
    public static void main(String[] args) {

        Messageable hello = Message::new;

        hello.getMessage("Hello");
    }
}
```



### 3. StreamAPI

### 3.1. Lọc dữ liệu - filter

#### 3.1.1. Lọc dữ liệu trong Collection không sử dụng Stream API

---

```
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}

public class Filter1 {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));
        List<Float> productPriceList = new ArrayList<Float>();
        for(Product product: productsList){
            if(product.price<30000){
                productPriceList.add(product.price);
            }
        }
        System.out.println(productPriceList);    // displaying data
    }
}
```

### 3.1.2. Lọc dữ liệu sử dụng Stream API

```
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}

public class JavaStreamExample {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));
        List<Float> productPriceList2 = productsList.stream()
            .filter(p -> p.price > 30000)// lọc dữ liệu
            .map(p->p.price)           // chỉ lấy price
            .collect(Collectors.toList()); //chuyển thành kiểu list

        System.out.println(productPriceList2);
    }
}
```

### 3.2. Chuyển dữ liệu - map



### 3.2.1. Sử dụng tham chiếu phương thức

```
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;    this.name = name;    this.price = price;
    }
    public int getId() {    return id;    }
    public String getName() {    return name;    }
    public float getPrice() {    return price;    }
}

public class StreamAPI_ThamchieuPhuongthuc {
    public static void main(String[] args) {

        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));

        List<Float> productPriceList =
            productsList.stream()
                .filter(p -> p.price > 30000) // filtering data
                .map(Product::getPrice) // fetching price by referring getPrice method
                .collect(Collectors.toList()); // collecting as list

        System.out.println(productPriceList);
    }
}
```

### 3.2.2. Sử dụng Lambda Expression

```
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}

public class JavaStreamLambdaExpression{
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));
        List<Float> productPriceList2 =productsList.stream()
            .filter(p -> p.price > 30000)// filtering data
            .map(p->p.price) // chỉ chọn price
            .collect(Collectors.toList()); // collecting as list

        System.out.println(productPriceList2);
    }
}
```

## 4. Lớp Collectors

## 4.1. Convert List into Set

```
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}

public class Collectors_toSet {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();

        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));

        // Converting product List into Set
        Set<Float> productPriceList =
            productsList.stream()
                .filter(product->product.price < 30000)    // filter product on the base of price
                .map(product->product.price)               // chỉ chọn price
                .collect(Collectors.toSet());               // collect it as Set(remove duplicate elements)
        System.out.println(productPriceList);
    }
}
```

## 4.2. Convert List into Map

```

class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}

public class JavaStreamExample {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();

        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));

        // Converting Product List into a Map
        Map<Integer,String> productPriceMap =
            productsList.stream()
                .collect( Collectors.toMap( p->p.id, p->p.name ) );

        System.out.println(productPriceMap);
    }
}

```

## 5. Lóp Optional

### 5.1. Sử dụng các phương thức get(), isPresent() và ifPresent()

```
import java.util.Optional;

class Student {
    String name;
}

public class OptionalExample1 {
    private Student getStudent() {
        Student student = new Student();
        return null;
    }

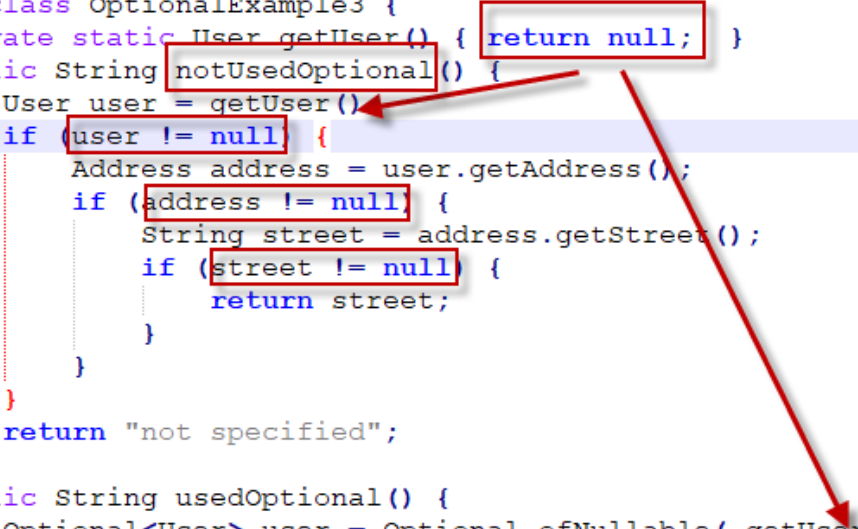
    public void notUsedOptional() {
        Student student = getStudent();
        if (student != null) {
            System.out.println(student.name);
        }
    }

    public void usedOptional() {
        Student student = getStudent();
        Optional<Student> opt = Optional.of(student);
        //Cách 1
        if (opt.isPresent()) {
            System.out.println(opt.get().name);
        }

        // Cách 2
        opt.ifPresent(s -> System.out.println(s.name));
    }
}
```

### 5.3. Sử dụng map

```
import java.util.Optional;
class User {
    private Address address;
    public Address getAddress() { return address; }
}
class Address {
    private String street;
    public String getStreet() { return street; }
}
public class OptionalExample3 {
    private static User getUser() { return null; }
    public String notUsedOptional() {
        User user = getUser();
        if (user != null) {
            Address address = user.getAddress();
            if (address != null) {
                String street = address.getStreet();
                if (street != null) {
                    return street;
                }
            }
        }
        return "not specified";
    }
    public String usedOptional() {
        Optional<User> user = Optional.ofNullable( getUser() );
        return user.map(User::getAddress)
                   .map(Address::getStreet)
                   .orElse("not specified");
    }
}
```



## 5.2. Sử dụng orElse() và orElseGet()

```
import java.util.Optional;

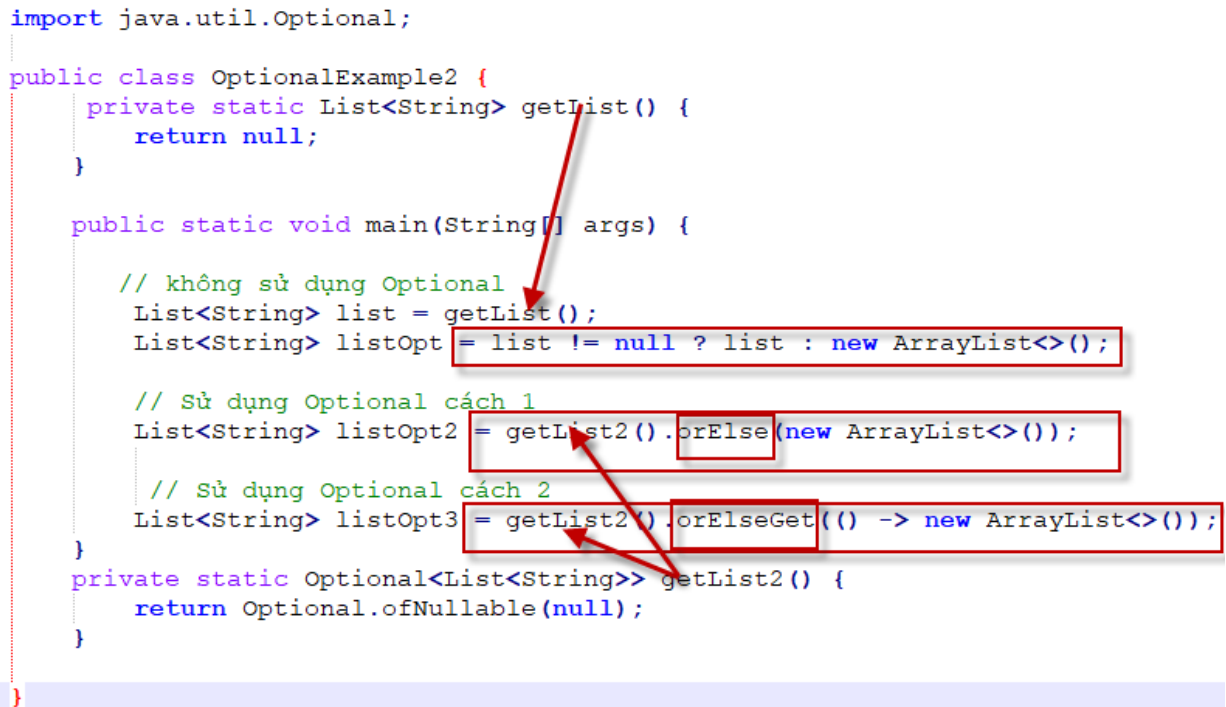
public class OptionalExample2 {
    private static List<String> getList() {
        return null;
    }

    public static void main(String[] args) {

        // không sử dụng Optional
        List<String> list = getList();
        List<String> listOpt = list != null ? list : new ArrayList<>();

        // Sử dụng Optional cách 1
        List<String> listOpt2 = getList().orElse(new ArrayList<>());

        // Sử dụng Optional cách 2
        List<String> listOpt3 = getList().orElseGet(() -> new ArrayList<>());
    }
    private static Optional<List<String>> getList2() {
        return Optional.ofNullable(null);
    }
}
```



## 6. Bài tập áp dụng

### 6.1. Cho các lớp sau:

#### 6. 1.1. Lớp Customer

```
public class Customer {
    private String name;
    private String address;

    public Customer (String n, String c){
        name=n;address=c;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
```

```

        this.name = name;
    }
    public String getCity() {
        return address;
    }
    public void setCity(String address) {
        this.address = address;
    }
    @Override
    public String toString(){
        return "Customer Name: " + name+" Address: "+ address;
    }
}

```

#### 6.1.2 Lớp Transaction

```

public class Transaction {
    private Customer trader;
    private Integer year;// Năm thực hiện giao dịch
    private String prodName;// Tên sản phẩm
    private Integer amount; // Số lượng
    public Transaction(Customer t, int y, String name, int v){
        trader=t; year=y; prodName=name; amount=v;
    }
    public Customer getCustomer() {
        return trader;
    }
    public void setCustomer(Customer trader) {

```

```
        this.trader = trader;
    }
    public Integer getYear() {
        return year;
    }
    public void setYear(Integer year) {
        this.year = year;
    }
}
```

```
@Override
public String toString(){
    return trader.toString()+" Transaction (Year: " + year
        +", ProdName: "+ getProdName()+", Amount: "+ getAmount() +")";
}
}
```

```
public String getProdName() {
    return prodName;
}
public void setProdName(String prodName) {
    this.prodName = prodName;
}
public Integer getAmount() {
    return amount;
}
public void setAmount(Integer amount) {
```



```
        this.amount = amount;
    }
}
```

## 6.2. Hay sử dụng Stream API thực hiện

6.2.1. Tìm tất cả các giao dịch thực hiện trong năm 2023 và sắp xếp chúng theo số lượng tăng dần.

6.2.2. In ra danh sách các địa chỉ của Customer (trùng thì loại)

6.2.3. Xác định tất cả các khách hàng chung **một địa chỉ nào đó**, sort by name của khách hàng và in ra.

6.2.4. Sắp xếp tất cả khách hàng theo tên và in ra.

6.2. 5. Kiểm tra xem có khách hàng ở địa chỉ nào đó hay không (y/n)?

6.2.6. Xác định số lượng (amount) lớn nhất xuất hiện trong các giao dịch và in ra.

6.2.7. Tính và in ra tổng số lượng trong các giao dịch của các khách hàng ở một địa chỉ nào đó

6.2.8. Xác định và in ra giao dịch có số lượng nhỏ nhất

## 7. Stream API nâng cao

Order, OrderDetail, Purchase, PurchaseDetail như trong bài Labs05-Collections-Advanced.

7.1. Cho cấu trúc sau

```
TreeMap< Order, ArrayList<OrderDetail>> tm = new TreeMap<>();
```

Hãy sử dụng các Stream API để tìm tất cả các hóa đơn có tổng số lượng lớn nhất.

7.2. Cho cấu trúc sau

```
HashMap<Customer, TreeMap<Order, ArrayList<OrderDetail>>> hm  
= new HashMap<>();
```

Hãy sử dụng các Stream API để tìm tất cả các khách hàng có tổng số tiền mua hàng lớn nhất.

