Lambda Expression

Reja:

1. Lambda Expression
2. Method references
3. Functional Interface API

Lambda Expression

1. Syntax
2. Lambda Expression Parameter
3. Lambda Expression Body
4. Lambda Expression Return Type
5. Lambda Expression hamda static va instance o’zgaruvchilar
6. Lambda Expression va local o’zgaruvchilar
7. Generic Functional Interface
8. Lambda Expression method parametri sifatida
9. Lambda Expression method natijasi sifatida

Syntax

Operationable addition = (a,b) -> a+b;

(a,b) -> a+b;

```
(parameter_list) -> {function_body}
```

- ✓Functional interface ni amalga oshirish usullaridan biri;
- ✓Alohida o’zgaruchiga o’zlashtirish mumkin bo’lgan bir nechta instruksiya(ko’rstma yoki amal)lar majmui;

Lambda Expression Parameter

- **Zero Parameters**

```
() -> System.out.println("Zero parameter lambda");
```

- **One Parameter**

```
(param) -> System.out.println("One parameter: " + param);
 param  -> System.out.println("One parameter: " + param);
```

- **Multiple Parameters**

```
(p1, p2) -> System.out.println("Multiple parameters: " + p1 + ", " + p2);
```

Lambda Expression Body

```
(oldState, newState) ->
    System.out.println("Old state: " + oldState+"\nNew state: " + newState)

(oldState, newState) -> {
    System.out.println("Old state: " + oldState);
    System.out.println("New state: " + newState);
}
```

Lambda Expression Return Type

```
(param) -> {
    System.out.println("param: " + param);
    return "return value";
}
```

Terminal (void qaytaradigan) lambda expressions

```
interface Printable {
    void print(String s);
}

public class LambdaApp {
    public static void main(String[] args) {
        Printable printer = s -> System.out.println(s);
        printer.print("Hello Java!");
    }
}
```

Bitta Functional Interface orqali birnechta Lambda Expression yaratish mumkin

```
Operationable operation1 = (int x, int y)-> x + y;
Operationable operation2 = (int x, int y)-> x - y;
Operationable operation3 = (int x, int y)-> x * y;

System.out.println(operation1.calculate(20, 10)); //30
System.out.println(operation2.calculate(20, 10)); //10
System.out.println(operation3.calculate(20, 10)); //200
```

Lambda Expression hamda static va instance o’zgaruvchilar

```
public class LambdaApp {

    static int x = 10;
    int y = 20;
    public static void main(String[] args) {

        Operation op = ()->{
            x=30;
           // this.y=50 //static bo’lmagan methodda this orqali ishlatsa bo’ladi
            return x+y;
        };
        System.out.println(op.calculate()); // 50
        System.out.println(x); //30
        System.out.println(y); //20

    }
}
interface Operation{
    int calculate();
}
```

PERSONAL
DEVELOPMENT PROCESS
stay weird, stay different

Lambda Expression va local o’zgaruvchilar

```
public static void main(String[] args) {

    int n=70;
    int m=30;
    Operation op = ()->{

        //n=100; - mumkin emas
        return m+n;
    };
    // n=100;  - mumkin emas
    System.out.println(op.calculate()); // 100
}
```

Generic Functional Interface

```
public class LambdaApp {

    public static void main(String[] args) {

        Operationable<Integer> operation1 = (x, y)-> x + y;
        Operationable<String> operation2 = (x, y) -> x + y;

        System.out.println(operation1.calculate(20, 10)); //30
        System.out.println(operation2.calculate("20", "10")); //2010
    }
}
interface Operationable<T>{
    T calculate(T x, T y);
}
```

Lambda Expression method parametri sifatida

```
public class LambdaApp {

    public static void main(String[] args) {

        Expression func = (n)-> n%2==0;
        int[] nums = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        System.out.println(sum(nums, func)); // 20
    }
    private static int sum (int[] numbers, Expression func)
    {
        int result = 0;
        for(int i : numbers)
        {
            if (func.isEqual(i))
                result += i;
        }
        return result;
    }
}

interface Expression{
    boolean isEqual(int n);
}
```

Lambda Expression method natijasi sifatida

```
interface Operation{
    int execute(int x, int y);
}

public class LambdaApp {

    public static void main(String[] args) {

        Operation func = action(1);
        int a = func.execute(6, 5);
        System.out.println(a);          // 11

        int b = action(2).execute(8, 2);
        System.out.println(b);          // 6
    }

    private static Operation action(int number){
        switch(number){
            case 1: return (x, y) -> x + y;
            case 2: return (x, y) -> x - y;
            case 3: return (x, y) -> x * y;
            default: return (x,y) -> 0;
        }
    }
```

Method references

1. **Method parametri sifatida static methodga havola**
2. **Method parametri sifatida instance methodga havola**
3. **Konstruktorga havola (constructor references)**

Method parametri sifatida static methodga havola (method references)

```
interface Expression{
    boolean isEqual(int n);
}

class ExpressionHelper{
    static boolean isEven(int n){
        return n%2 == 0;
    }

    static boolean isPositive(int n){
        return n > 0;
    }
}
```

```
public class LambdaApp {

    public static void main(String[] args) {

        int[] nums = { -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5};
        System.out.println(sum(nums,
ExpressionHelper::isEven));

        Expression expr = ExpressionHelper::isPositive;
        System.out.println(sum(nums, expr));
    }

    private static int sum (int[] numbers, Expression func)
    {
        int result = 0;
        for(int i : numbers)
        {
            if (func.isEqual(i))
                result += i;
        }
        return result;
    }
}
```

Method parametri sifatida instance methodga havola

```
public class LambdaApp {

    public static void main(String[] args) {

        int[] nums = { -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5};
        ExpressionHelper exprHelper = new ExpressionHelper();
        System.out.println(sum(nums, exprHelper::isEven)); // 0
    }

    private static int sum (int[] numbers, Expression func)
    {
        int result = 0;
        for(int i : numbers)
        {
            if (func.isEqual(i))
                result += i;
        }
        return result;
    }
}
```

Konstruktorga havola (constructor references)

```
public class LambdaApp {

    public static void main(String[] args) {

        UserBuilder userBuilder = User::new;
        User user = userBuilder.create("Tom");
        System.out.println(user.getName());
    }
}
interface UserBuilder{
    User create(String name);
}

class User{

    private String name;
    String getName(){
        return name;
    }

    User(String n){
        this.name=n;
    }
}
```

PERSONAL
DEVELOPMENT PROCESS
stay weird, stay different

Functional Interface API

- Predicate<T>
- Consumer<T>
- Function<T,R>
- Supplier<T>
- UnaryOperator<T>
- BinaryOperator<T>

Predicate<T>

Berilgan qiymatni o’rnatilgan shartga tekshirish uchun qo’llaniladi.

```
public interface Predicate<T> {
    boolean test(T t);
}
```

```java
import java.util.function.Predicate;

public class LambdaApp {

    public static void main(String[] args) {

        Predicate<Integer> isPositive = x -> x > 0;

        System.out.println(isPositive.test(5)); // true
        System.out.println(isPositive.test(-7)); // false
    }
}
```

Function<T, R>

Bir toifadagi ma’lumotni ikkinchi toifaga o’tkazish uchun qo’llaniladi

```
public interface Function<T, R> {
    R apply(T t);
}
```

```
public class LambdaApp {

    public static void main(String[] args) {

        Function<Integer, String> convert =
                                x-> String.valueOf(x) + " dollar";

        System.out.println(convert.apply(5)); // 5 dollar
    }
}
```

Consumer<T>

Object ustida qanaqadir amal bajarib hech nima qaytarmaydi.

```
public interface Consumer<T> {
    void accept(T t);
}
```

```
public class LambdaApp {

    public static void main(String[] args) {

        Consumer<Integer> printer =
                    x-> System.out.printf("%d dollar \n", x);

        printer.accept(600); // 600 dollar
    }
}
```

Supplier<T>

Hech qanday qiymat qabul qilmasdan belgilangan toifadagi qiymat qaytaradi

```
public interface Supplier<T> {
    T get();
}
```

PERSONAL
DEVELOPMENT PROCESS
stay weird, stay different

```
class User{

    private String name;
    String getName(){
        return name;
    }

    User(String n){
        this.name=n;
    }
}
```

```
public class LambdaApp {

    public static void main(String[] args) {

        Supplier<User> userFactory = ()->{

            Scanner in = new Scanner(System.in);
            System.out.println("Введите имя: ");
            String name = in.nextLine();
            return new User(name);
        };

        User user1 = userFactory.get();
        User user2 = userFactory.get();

        System.out.println("Имя user1: " + user1.getName());
        System.out.println("Имя user2: " + user2.getName());
    }
}
```

E’TIBORINGIZ UCHUN RAXMAT

PERSONAL
DEVELOPMENT PROCESS
stay weird, stay different