НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Объектно-ориентированное программирование на языке Java

Лямбда-выражения

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

История

Язык программирования Java был разработан как объектно-ориентированный в 1990-е годы, когда методика объектно-ориентированного программирования стала господствующей в разработке программного обеспечения. А задолго до объектно-ориентированного программирования появились языки функционального программирования вроде Lisp и Scheme, но их преимущества не были должным образом оценены за пределами академических кругов. И лишь недавно значение функционального программирования возросло потому, что оно оказывается вполне пригодным для параллельного и событийно-ориентированного (так называемого "реактивного") программирования. Это совсем не означает, что объекты сами по себе плохи. Напротив, выигрышная стратегия состоит в том, чтобы сочетать объектно-ориентированное программирование с функциональным.

Введение

Java изначально полностью объектно-ориентированный язык. За исключением примитивных типов, все в Java – это объекты. Даже массивы являются объектами. Экземпляры каждого класса – объекты. Не существует ни единой возможности определить отдельно (вне класса) какую-нибудь функцию. И нет никакой возможности передать метод как аргумент или вернуть тело метода как результат другого метода. Все так. Но так было до Java 8.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Введение

В функциональном языке lambda-выражения – это функции; но в Java, lambda-выражения – представляются объектами, и должны быть связаны с конкретным объектным типом, который называется функциональный интерфейс. Далее мы рассмотри, что он из себя представляет.

Lambda-выражения - это метод без объявления, т.е. без модификаторов доступа, возвращающие значение и имя. Они позволяют написать метод и сразу же использовать его. Особенно полезно в случае однократного вызова метода, т.к. сокращает время на объявление и написание метода без необходимости создавать класс.

Lambda-выражения в Java обычно имеют следующий синтаксис:

(аргументы) -> (тело).

Введение

- Лямбда представляет набор инструкций, которые можно выделить в отдельную переменную и затем многократно вызвать в различных местах программы.
- Основу лямбда-выражения составляет лямбда-оператор, который представляет стрелку ->. Этот оператор разделяет лямбда-выражение на две части: левая часть содержит список параметров выражения, а правая собственно представляет тело лямбда-выражения, где выполняются все действия.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Введение

- Лямбда-выражение не выполняется само по себе, а образует реализацию метода, определенного в функциональном интерфейсе. При этом важно, что **функциональный интерфейс должен содержать только один единственный метод без реализации.**

- Рассмотрим пример:

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Введение

```
private void run() {
   Operationable op = new Operationable(){
       public int calculate(int x, int y){
          return x + y;
       }
    };
    int z = op.calculate(20, 10);    System.out.println(z); // 30
}


public interface Operationable {
    public int calculate(int x, int y);
}
```

Введение

- В роли функционального интерфейса выступает интерфейс Operationable, в котором определен один метод без реализации - метод calculate. Данный метод принимает два параметра - целых числа, и возвращает некоторое целое число.
- По факту лямбда-выражения являются в некотором роде сокращенной формой внутренних **анонимных классов**, которые ранее применялись в Java. В частности, предыдущий пример мы можем переписать следующим образом:

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Введение

```
private void run() {
    Operationable op;
    op= (x, y) -> x + y;
    int z = op.calculate(20, 10);
    System.out.println(z); // 30
}


public interface Operationable {
    public int calculate(int x, int y);
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Введение

Чтобы объявить и использовать лямбда-выражение, основная программа разбивается на ряд этапов:

Определение ссылки на **функциональный интерфейс**:

```
Operationable operation;
```

Создание лямбда-выражения:

```
operation = (x,y)->x+y;
```

Причем параметры лямбда-выражения соответствуют параметрам единственного метода интерфейса Operationable, а результат соответствует возвращаемому результату метода интерфейса. При этом нам не надо использовать ключевое слово return для возврата результата из лямбда-выражения.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Введение

А если надо вычислить другие арифметические действия. Например:

```
Operationable op1= (x, y) -> x + y;
Operationable op2= (x, y) -> x * y;
Operationable op3= (x, y) -> x / y;

System.out.println(op1.calculate(20, 10)); // 30
System.out.println(op2.calculate(20, 10)); // 200
System.out.println(op3.calculate(20, 10)); // 2
```

Отложенное выполнение

- Одним из ключевых моментов в использовании лямбд является отложенное выполнение (deferred execution). То есть мы определяем в одном месте программы лямбда-выражение и затем можем его вызывать при необходимости неопределенное количество раз в различных частях программы. Отложенное выполнение может потребоваться, к примеру, в следующих случаях:

Отложенное выполнение

- Выполнение кода отдельном потоке
- Выполнение одного и того же кода несколько раз
- Выполнение кода в результате какого-то события
- Выполнение кода только в том случае, когда он действительно необходим и если он необходим
- Передача параметров в лямбда-выражение

Отложенное выполнение

- Параметры лямбда-выражения должны соответствовать по типу параметрам метода из функционального интерфейса.
- При написании самого лямбда-выражения тип параметров писать необязательно, хотя в принципе это можно сделать, например:

- operation = (int x, int y)->x+y;

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Отложенное выполнение

Если метод не принимает никаких параметров, то пишутся пустые скобки, например:

```
()-> 30 + 20;
```

Если метод принимает только один параметр, то скобки можно опустить:

```
n-> n * n;
```

Терминальные лямбда-выражения

Выше мы рассмотрели лямбда-выражения, которые возвращают определенное значение. Но также могут быть и терминальные лямбды, которые не возвращают никакого значения. Например:

```
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!");
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Лямбды и локальные переменные

Лямбда-выражение может использовать переменные, которые объявлены на уровне класса или метода, в котором лямбда-выражение определено. Однако в зависимости от того, как и где определены переменные, могут различаться способы их использования в лямбдах. Рассмотрим первый пример - использования переменных уровня класса:

```
   static int x = 10;
   static int y = 20;
   public static void main(String[] args) {
            Operation op = ()->{  x=30;    return x+y;  };
      System.out.println(op.calculate()); // 50
      System.out.println(x); // 30 - значение x изменилось
   }
}
interface Operation{
   int calculate();
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Лямбды и локальные переменные

- Переменные x и y объявлены на уровне класса, и в лямбда-выражении мы их может получить и даже изменить. Так, в данном случае после выполнения выражения изменяется значение переменной x.
- Теперь рассмотрим другой пример - локальные переменные на уровне метода:

```
public static void main(String[] args) {
    int n=70;    int m=30;
    Operation op = ()->{
        //n=100; - так нельзя сделать
        return m+n;
    };
    // n=100;  - так тоже нельзя
    System.out.println(op.calculate()); // 100
}
```

Лямбды и локальные переменные

- Локальные переменные уровня метода мы также может использовать в лямбдах, но изменять их значение мы уже не сможем. Если мы попробуем это сделать, то среда разработки может нам высветить ошибку и то, что такую переменную надо пометить с помощью ключевого слова final, то есть сделать константой: final int n=70;. Однако это необязательно.
- Более того, мы не сможем изменить значение переменной, которая используется в лямбда-выражении, вне этого выражения. То есть даже если такая переменная не объявлена как константа, по сути она является константой.

Блоки кода в лямбда-выражениях

- Существуют два типа лямбда-выражений: однострочное выражение и блок кода. Примеры однострочных выражений демонстрировались выше.
- Блочные выражения обрамляются фигурными скобками. В блочных лямбда-выражениях можно использовать внутренние вложенные блоки, циклы, конструкции if, switch, создавать переменные и т.д.
- Если блочное лямбда-выражение должно возвращать значение, то явным образом применяется оператор return:

Блоки кода в лямбда-выражениях

```
Operationable operation = (int x, int y)-> {

    if(y==0)
        return 0;
    else
        return x/y;
};

System.out.println(operation.calculate(20, 10)); //2
System.out.println(operation.calculate(20, 0)); //0
```

Обобщенный функциональный интерфейс

- Функциональный интерфейс может быть обобщенным, однако в лямбда-выражении использование обобщений не допускается. В этом случае нам надо типизировать объект интерфейса определенным типом, который потом будет применяться в лямбда-выражении. Например:

```
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);
}
```