Урок 11.
Лямбда-выражения

Гарбузов Николай
Donriver Inc.

DonRiver

Agenda

- Введение в Лямбда-выражения
- Функциональный интерфейс и лямбда выражения
- Обобщенный функциональный интерфейс
- Передача лямбда-выражений в качестве аргументов
- Ссылки на методы, конструкторы
- Немного хардкора

Введение в Лямбда-выражения

Лямбда выражения особые конструкции языка, которые

- повышают выразительность языка
- сокращают объем кода
- упрощают реализацию параллельной обработки данных
- могут быть использованы в любом месте кода, где известен их тип
- и тд

Функциональный интерфейс и лямбда выражения

Лямбда выражения состоят из

- **лямбда выражения**
- **функционального интерфейса**

Функциональный интерфейс и лямбда выражения

Лямбда выражения состоят из

- **функционального интерфейса** интерфейс, содержащий ровно 1 абстрактный метод (напр, Runnable, Comparable)
- **лямбда выражения** анонимный метод, реализующий метод, который определен в функциональном интерфейсе

Функциональный интерфейс и лямбда выражения

Лямбда выражения состоят из

- **функционального интерфейса** интерфейс, содержащий ровно 1 абстрактный метод (напр, Runnable, Comparable)
- **лямбда выражения** анонимный метод, реализующий метод, который определен в функциональном интерфейсе

Q:
А можно ли иметь методы по умолчанию?

Функциональный интерфейс и лямбда выражения

Лямбда выражения состоят из

- **функционального интерфейса** интерфейс, содержащий ровно 1 абстрактный метод (напр, Runnable, Comparable)
- **лямбда выражения** анонимный метод, реализующий метод, который определен в функциональном интерфейсе

Q:
А можно ли иметь методы по умолчанию?

A:
Да, можно, главное,
чтобы был ровно один абстрактный метод!

Функциональный интерфейс и лямбда выражения

Лямбда выражения

- определяются с помощью лямбда-оператора “->”
- могут быть однострочными и блоковыми

Функциональный интерфейс и лямбда выражения

Лямбда выражения

- определяются с помощью лямбда-оператора “->”
- могут быть однострочными и блоковыми
- могут работать с пустым списком аргументов

Функциональный интерфейс и лямбда выражения

Лямбда выражения

- определяются с помощью лямбда-оператора “->”
- могут быть однострочными и блоковыми
- могут работать с пустым списком аргументов
- могут использовать переменные контекста, но только завершенные!

Функциональный интерфейс и лямбда выражения

Лямбда выражения

- определяются с помощью лямбда-оператора “->”
- могут быть однострочными и блоковыми
- могут работать с пустым списком аргументов
- могут использовать переменные контекста, но только завершенные!

()->10	лямбда выражение, возвращающее 10
i -> ++i	лямбда выражение - инкремент
(i,j) -> i==j ? 0 : (i>j ? 1 : -1)	лямбда выражение - компаратор

Обобщенный функциональный интерфейс

- **Функциональный интерфейс** может быть обобщенным
- **Лямбда выражение** может быть параметризовано!

Обобщенный функциональный интерфейс

- **Функциональный интерфейс** может быть обобщенным
- **Лямбда выражение** может быть параметризовано!

```
interface MyCalc<T>{ T calc(T t)}

MyCalc<Integer> calcInt = i -> ++i;

MyCalc<Long> calcLong = l -> ++l;
```

Передача лямбда-выражений в качестве аргументов

Лямбда выражения особые конструкции языка, которые

- повышают выразительность языка
- упрощают реализацию параллельной обработки данных
- сокращают объем кода
- могут быть использованы в любом месте кода, где известен их тип

Передача лямбда-выражений в качестве аргументов

Рекурсия с помощью лямбда выражения

```
public interface Recursion {
   int rec(Recursion r, int i);
}

Recursion fact = (r, i) -> i == 1 ? 1 : i * r.rec(r, i - 1);

fact.rec(fact,5); // 120
```

Передача лямбда-выражений в качестве аргументов

Рекурсия с помощью лямбда выражения

```
public interface Recursion {
   int rec(Recursion r, int i);
}

Recursion fact = (r, i) -> i == 1 ? 1 : i * r.rec(r, i - 1);

fact.rec(fact,5); // 120
```

Ссылки на методы, конструкторы

- Ссылка на метод может быть
  - на статический метод Имя_Класса::Имя_Метода
  - на метод экземпляра Имя_Экземпляра::Имя_Метода

- Ссылка на конструктор Имя_Класса::new

Ссылки на методы, конструкторы

- Ссылка на метод может быть
  - на статический метод Имя_Класса::Имя_Метода
  - на метод экземпляра Имя_Экземпляра::Имя_Метода

- Ссылка на конструктор Имя_Класса::new

Q:
Зачем?

A:

- Повышает динамичность языка
- Позволяет создавать более гибкую архитектуру

Ссылки на методы, конструкторы

- Ссылка на метод может быть
  - на статический метод Имя_Класса::Имя_Метода
  - на метод экземпляра Имя_Экземпляра::Имя_Метода

- Ссылка на конструктор Имя_Класса::new

Q:
Зачем?

A:

- Повышает динамичность языка
- Позволяет создавать более гибкую архитектуру
- Сохраняет проверки на уровне компиляции!!!

Немного хардкора

Раньше было только 4 инструкции в байт-коде для вызова метода

- invokestatic
- invokevirtual
- invokeinterface
- invokespecial

Немного хардкора

Раньше было только 4 инструкции в байт-коде для вызова метода

- invokestatic - для статичным методов
- invokevirtual - для обычных методов (через таблицы методов)
- invokeinterface - для метода интерфейса (идет поиск метода)
- invokespecial - для конструктора

- !! invokedynamic !! - для вызова методов динамически!

Немного хардкора

invokedynamic - некоторая общая инструкция, позволяющая выбирать методы гибко и динамически

Состоит из

- invokedynamic байткода
- Динамический места вызова
- Хэндлеры методов

Home Work

- Добавить поддержку лямбд в консольный чат. Использовать их вместо анонимных классов при передаче кода в Thread