Објектно орјентисано програмирање

Владимир Филиповић
vladaf@matf.bg.ac.rs
Александар Картељ
kartelj@matf.bg.ac.rs

Ламбда изрази

Владимир Филиповић
vladaf@matf.bg.ac.rs
Александар Картељ
kartelj@matf.bg.ac.rs

Математички факултет
vladaf@matf.bg.ac.rs

Мотивација

- Употреба анонимних класа је обично везана за *ad-hoc* имплементацију одређене функције:
  - Ова функција се користи једнократно;
  - Смисао постојања овакве класе је пренос функције, па се често он назива и функцијски објекат (слично показивачу на функцију);
  - Овакав запис је, међутим, доста гломазан.
- Ламбда изрази омогућавају да се ова иста употреба реализује елегантније.
- Кроз наредних неколико примера ће, корак по корак, бити разјашњена ова мотивација.

Математички факултет
vladaf@matf.bg.ac.rs

Пример - опис

- Претпоставимо да је потребно да дизајнирамо **социјалну мрежу**.
- Једна од могућности у оквиру система је административни панел који би омогућио:
  - Администратору да излиста све кориснике који испуњавају одређени критеријум;
  - На овај начин би администратор даље могао да врши одређене активности над тим корисницима:
    - Шаље поруке и обавештења;
    - Брише или ажурира податке и слично.
  - Критеријума може да буде доста, па је потребно водити рачуна о компактности кода.

Математички факултет

vladaf@matf.bg.ac.rs

Пример (2)

- Главни ентитет је класа особа:

```
public class Person {
        public enum Sex { MALE, FEMALE }
        String name;
        LocalDate birthday;
        Sex gender;
        String emailAddress;

        public int getAge() {
                // ...
        }

        public void printPerson() {
                // ...
        }
}
```

Математички факултет
vladaf@matf.bg.ac.rs

Наивни приступ – појединачне функције

- Рецимо да је први захтев листање свих корисника који су старији од задатог броја година:

```
static void printPersonsOlderThan(List<Person> roster, int age) {
        for (Person p : roster) {
                if (p.getAge() >= age) {
                        p.printPerson();
                }
        }
}
```

Математички факултет
vladaf@matf.bg.ac.rs

Наивни приступ – појединачне функције (2)

- Међутим, пожељно је да излистамо и особе које су у неком задатом опсегу година:

```
static void printPersonsWithinAgeRange( List<Person> roster,
        int low, int high) {
        for (Person p : roster) {
                if (low <= p.getAge() && p.getAge() < high) {
                        p.printPerson();
                }
        }
}
```

- Јасно је да критеријума може да буде јако пуно, па прављење појединачних функција за сваки критеријум није решење!

Математички факултет

vladaf@matf.bg.ac.rs

Напреднији приступ - анонимне класе

- Следећи логичан корак би било прављење методе за испис особа којој се жељени критеријум прослеђује као функцијски објекат, односно анонимна класа.

```
static void printPersons( List<Person> roster, CheckPerson tester) {
        for (Person p : roster) {
                if (tester.test(p)) {
                        p.printPerson();
                }
        }
}
```

- Где прослеђени функцијски објекат за задавање критеријума имплментира интерфејс:

```
interface CheckPerson {
        boolean test(Person p);
}
```

Математички факултет
vladaf@matf.bg.ac.rs

Напреднији приступ - анонимне класе (2)

- Тада је, на доста елегантнији начин, могуће задати доста произвољан критеријум, нпр:

```
printPersons( roster,
              new CheckPerson() {
                     public boolean test(Person p) {
                            return p.getGender() == Person.Sex.MALE
                                   && p.getAge() >= 18
                                   && p.getAge() <= 25;
                     }
              }
);
```

Математички факултет

vladaf@matf.bg.ac.rs

Још напреднији приступ – ламбда изрази

- Међутим, употребом ламбда израза можемо то да изразимо још елегантније:

```
printPersons( roster,
              (Person p) -> p.getGender() == Person.Sex.MALE
                         && p.getAge() >= 18
                         && p.getAge() <= 25
);
```

- Синтакса ламбда израза је следећа:
  1. (arg1, arg2, ...) – листа параметара ламбда израза;
  2. `->` – почетак тела ламбда израза;
  3. Након чега следи сам ламбда израз који по синтакси представља валидан Јава израз, и чија повратна вредност мора да одговара повратном типу наведеном у одговарајућем интерфејсу (нпр. CheckPerson).

Математички факултет

vladaf@matf.bg.ac.rs

Уграђени функцијски интерфејси

- Додатно, уместо да сами дефинишемо интерфејсе, могуће је користити неке стандардне уграђене генеричке:

```
static void printPersonsWithPredicate( List<Person> roster,
                                       Predicate<Person> tester) {
        for (Person p : roster) {
                if (tester.test(p)) {
                        p.printPerson();
                }
        }
}
```

Где је Predicate<T> уграђени генерички интерфејс из пакета java.util.function:

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

Математички факултет
vladaf@matf.bg.ac.rs

Погоднија нотација – *filter, forEach*

- Постоји и нешто погоднија нотација за примену ламбда израза над колекцијама података:

```
roster
        .stream()
        .filter( p -> p.getGender() == Person.Sex.MALE
                      && p.getAge() >= 18 && p.getAge() <= 25)
        .forEach(p -> System.out.println(p));
```

Где су функције stream, filter и forEach дефинисане на следећи начин:

Stream<E> **stream**()	Претвара произвољну колекцију у ток за рад са ламбда изразима
Stream<T> **filter**(Predicate<? super T> predicate)	Примењује операцију Predicate<? Super T> над свим елементима тока
void **forEach**(Consumer<? super T> action)	Извршава void функцију над свим елементима тока, који су преостали након филтрирања (filter)

Математички факултет

vladaf@matf.bg.ac.rs

Погоднија нотација - *map*

- Поред могућности филтрирања колекције и примене void функције, могуће су и трансформације сваког елемента листе у други тип података:

```
roster
        .stream()
        .filter( p -> p.getGender() == Person.Sex.MALE &&
                      p.getAge() >= 18 && p.getAge() <= 25)
        .map(p -> p.getName().length())
        .forEach(len -> System.out.println(len));
```

- Функцијом **map** се од сваке појединачне особе узима дужина имена, а потом се у **forEach** функцији даље подразумевано ради са тим податком, дакле, целим бројем;
- Овај број се локално именује са **len**, а могло је и било којим другим именом.

Математички факултет
vladaf@matf.bg.ac.rs

Погоднија нотација – агрегатне функције

- Досад је размотрена само примена void функције над елементима колекције, међутим, могуће је да као резултат неких операција над колекцијом буде враћен резултат:

```
roster
        .stream()
        .filter( p -> p.getGender() == Person.Sex.MALE &&
                p.getAge() >= 18 && p.getAge() <= 25)
        .map(p -> p.getName().length())
        .sum();
```

- У овом примеру се над дужинама имена особа примењује сумирање, такорећи, рачуна се сума дужина имена особа које су мушкарци између 18 и 25 година;
- Могуће је извршавати и разне друге произвољне агрегације над колекцијом употребом функције **reduce.**
  - Више о функцији **reduce** може се пронаћи на овој адреси.

Математички факултет
vladaf@matf.bg.ac.rs

Закључак

- Ламбда изрази нису изворно елементи објектне-оријентисане парадигме.
  - Они се везују за тзв. функционалну парадигму у којој је све функцијски објекат и где се сви проблеми решавају прављењем одговарајућих композиција функција и применом рекурзије.
- Неки од познатијих функционалних програмских језика су: Lisp, Haskell, F#, Elrang и други.
- Већина модерних објектно-оријентисаних језика је прихватила и интегрисала функционалну парадигму.
  - Ово је омогућило пре свега лакши рад са колекцијама и токовима података;
  - Због своје декларативне природе, ламбда изрази омогућавају програмеру интуитивније изражавање, те се очекује да у будућности буду незаобилазни део Јава, али и других ОО програмских језика.

Математички факултет

vladaf@matf.bg.ac.rs

Захвалница

Велики део материјала који је укључен у ову презентацију је преузет из презентације коју је раније (у време када је он држао курс Објектно орјентисано програмирање) направио проф. др Душан Тошић.

Хвала проф. Тошићу што се сагласио са укључивањем тог материјала у садашњу презентацију, као и на помоћи коју ми је пружио током конципцирања и реализације курса.

Надаље, један део материјала је преузет од колегинице Марије Милановић.

Хвала Марији Милановић на помоћи у реализацији ове презентације.