ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Java Built-in Interfaces

Αθ. Ανδρούτσος

Lambdas

- Τα **Lambda Expressions** είναι –όπως είδαμε- ένα νέο σημαντικό χαρακτηριστικό ήδη από την Java 8
- Το πρόβλημα που λύνουν είναι πως σε μία εγγενώς αντικειμενοστραφή γλώσσα όπως η Java θα έχουμε **Functional Programming**
- Δηλαδή να μπορούμε να χρησιμοποιούμε functions και ιδιαίτερα ως callbacks
- Έτσι ο κώδικας γίνεται πιο σύντομος, πιο απλός, πιο κατανοητός, πιο ευανάγνωστος

Java Collections Library

- Τα callbacks, που υλοποιούνται με **Lambda Expressions χρησιμοποιούνται ιδιαίτερα σε συνδυασμό με τα Java Collections** κάνοντας πιο εύκολο το iteration, δηλαδή τη διάσχιση (traversal) αυτών των δομών δεδομένων καθώς και το φιλτράρισμα και την εξαγωγή δεδομένων από τα Collections, χωρίς να χρειάζεται να γράφουμε boilerplate (επαναλαμβανόμενο) κώδικα με for

Lambdas και interfaces

- Για να χρησιμοποιούμε Lambdas με interfaces, το interface θα πρέπει να περιέχει μία μόνο μέθοδο (**functional interface**)
- Τα Lambdas μπορούν να υλοποιήσουν ένα functional interface χωρίς να δημιουργήσουμε κάποια κλάση ή anonymous class
- Αν όμως το interface έχει περισσότερες από μία μεθόδους, τότε δεν μπορούν να χρησιμοποιηθούν lambdas, παρά μόνο anonymous classes

Multicore Architecture

- Τώρα που πλέον η ισχύς των επεξεργαστών έχει φτάσει σε ένα άνω όριο, η έμφαση δίνεται στις αρχιτεκτονικές πολλών επεξεργαστών (πολλών πυρήνων – multicore, κάθε πυρήνας είναι ένας κανονικότατος επεξεργαστής)
- Τα **Lambdas έχουν σχεδιαστεί ώστε να υποστηρίζουν multicore αρχιτεκτονικές**, να παρέχουν δηλαδή παράλληλη επεξεργασία, που με τη σειρά της βελτιώνει τις επιδόσεις και μειώνει τον χρόνο εκτέλεσης

Πλεονεκτήματα Lambdas

- Συνοπτική σύνταξη
- References σε μεθόδους και constructors
- Μειωμένος χρόνος εκτέλεσης σε σύγκριση με τις anonymous classes

Παράδειγμα

- Θα δούμε στη συνέχεια ένα παράδειγμα με ένα collection από teachers το οποίο θέλουμε να μπορούμε να φιλτράρουμε και να λαμβάνουμε πίσω teachers με συγκεκριμένα id π.χ. id == 10 ή id >= 10 ή id <= 5 κλπ.
- Πως θα μπορούσαμε όμως να υλοποιήσουμε δυναμικά κάτι τέτοιο, να μπορούμε δηλαδή δυναμικά @runtime να δίνουμε συνθήκες αναζήτησης (filtering)

Predicates

- Αν παρατηρήσουμε οι συνθήκες id == 10, id >= 10, id <= 5 επιστρέφουν true/false για κάθε id που ελέγχουν
- Είναι στην πραγματικότητα boolean functions ή αλλιώς predicates

Teacher class (1)


```
1   package gr.aueb.cf.ch20;
2
3   public class Teacher {
4       private int teacherId;
5       private String firstname;
6       private String lastname;
7
8       public Teacher() {}
9       public Teacher(int teacherId, String firstname, String lastname) {...}
14
15      public int getTeacherId() { return teacherId; }
18      public void setTeacherId(int teacherId) { this.teacherId = teacherId; }
21      public String getFirstname() { return firstname; }
24      public void setFirstname(String firstname) { this.firstname = firstname; }
27      public String getLastname() { return lastname; }
30      public void setLastname(String lastname) { this.lastname = lastname; }
```

- Έστω μία κλάση Teacher (βλ. επόμενη διαφάνεια)

Teacher class (2)


```
34      public void printFullName() {
35          System.out.println("ID: " + teacherId + " Firstname: " +
36                  firstname + " Lastname: " + lastname);
37      }
38
39  @   public static void printTeacher(Teacher teacher) {
40          System.out.printf("ID: %d, Firstname: %s, Lastname: %s\n", teacher.getTeacherId(),
41                  teacher.getFirstname(), teacher.getLastname());
42      }
43
44      @Override
45      public String toString() {
46          return "Teacher{" +
47                  "teacherId=" + teacherId +
48                  ", firstname='" + firstname + '\'' +
49                  ", lastname='" + lastname + '\'' +
50                  '}';
51      }
```

- Η **printFullName()** είναι instance method. Η ***printTeacher()*** έχει δηλωθεί ως *static* ώστε να μπορεί να καλείται χωρίς να χρειάζεται η κλάση μας να γίνει instantiate

Teacher Search (1)

- Ας υποθέσουμε ότι έχουμε μία λίστα από Teachers και θέλουμε να φιλτράρουμε με κριτήρια
- Μία πρώτη απλή σκέψη θα ήταν να περνάμε το id ως παράμετρο μία μεθόδου
- Θα δημιουργήσουμε δύο μεθόδους: 1) μία που θα αναζητά (φιλτράρει) για ισότητα, και 2) μία μέθοδο που θα φιλτράρει για id μεγαλύτερα από το id που δίνεται ως παράμετρος

Teacher Search (2)


```
46  @       public static void printEquals(List<Teacher> teachers, int teacherID) {
47              for (Teacher teacher : teachers) {
48                  if (teacher.getTeacherId() == teacherID) {
49                      System.out.println("Full Name: " + teacher.getFirstname()
50                              + ", " + teacher.getLastname());
51                  }
52              }
53          }
54
55  @       public static void printGreaterThanID(List<Teacher> teachers, int teacherID) {
56              for (Teacher teacher : teachers) {
57                  if (teacher.getTeacherId() > teacherID) {
58                      System.out.println("Full Name: " + teacher.getFirstname()
59                              + ", " + teacher.getLastname());
60                  }
61              }
62          }
```

- Και οι δύο μέθοδοι παίρνουν ως παραμέτρους μία λίστα teachers και μία παράμετρο για το id

Teacher Search (3)


```
13  public class Main {
14
15      private static final List<Teacher> teachers = Arrays.asList(
16              new Teacher(1, "A.", "Cooper"),
17              new Teacher(2, "B.", "Dylan"),
18              new Teacher(3, "E.", "Mask"),
19              new Teacher(4, "B.", "Gates")
20      );
21
22      public static void main(String[] args) {
23          printEquals(teachers, 4);
24          printGreaterThanID(teachers, 2);
```

- Το πρόβλημα εδώ είναι ότι στις δύο μεθόδους **περνάμε ως παραμέτρους τιμές, όχι συνθήκες για filtering**. Αν θέλαμε για παράδειγμα να βρίσκουμε id <= value θα έπρεπε να γράψουμε και μία ακόμα μέθοδο

interface

- Πως μπορούμε όμως @runtime να **περνάμε ως παραμέτρους predicates** (boolean functions)
- Απλά μπορούμε να έχουμε ως **παράμετρο ένα Functional Interface**
- Οπότε **@runtime θα μπορούμε να περνάμε ως παραμέτρους lambdas** ως predicates

Implementation of Interface


```java
package gr.aueb.cf.ch20;

@FunctionalInterface
public interface ITeacherIDFilter {
    boolean filterId(Teacher teacher);
}
```

- Έστω το functional interface **TeacherIDFilter** που περιέχει μία μόνο boolean μέθοδο την **filterId(Teacher teacher)**

getFilteredTeachers


```
43 @    public static List<Teacher> getFilteredTeachers(List<Teacher> teachers, ITeacherIDFilter filter) {
44          List<Teacher> teachersToReturn = new ArrayList<>();
45          for (Teacher teacher : teachers) {
46              if (filter.filterId(teacher)) {
47                  teachersToReturn.add(teacher);
48              }
49          }
50          return teachersToReturn;
51      }
```

- Η getFilteredTeachers() είναι μία μέθοδος που παίρνει ως παράμετρο το ITeacherIdFilter
- Στο σώμα της ελέγχει με την .filterId() και επιστρέφει μία λίστα

Ανώνυμη κλάση


```
18    public static void main(String[] args) {
19
20        List<Teacher> filteredTeachers = getFilteredTeachers(teachers, new ITeacherIDFilter() {
21            @Override
22            public boolean filterId(Teacher teacher) {
23                return teacher.getTeacherId() == 3;
24            }
25        });
26
27        for (Teacher teacher : filteredTeachers) {
28            teacher.printFullName();
29        }
```

- Η υλοποίηση του interface με ανώνυμη κλάση είναι verbose. Θα δούμε στην επόμενη διαφάνεια την υλοποίηση με lambda

Lambda


```
18  public static void main(String[] args) {
19      List<Teacher> filteredTeachers = getFilteredTeachers(teachers, teacher -> teacher.getTeacherId() == 3);
20
21      for (Teacher teacher : filteredTeachers) {
22          teacher.printFullName();
23      }
```

- Τα Lambda expressions είναι πιο συνοπτικά
- Παρατηρήστε το 1ο μέρος του lambda, που έχει μία παράμετρο την **teacher** (όταν έχουμε μία μόνο παράμετρο δεν χρειάζονται παρενθέσεις, σε όλες τις άλλες περιπτώσεις χρειάζονται παρενθέσεις)
- Επίσης στο 2ο μέρος του lambda -μετά το arrow- είναι το expression. Αν έχουμε μία απλή εντολή return, δεν χρειάζεται ούτε άγκιστρα, ούτε return (το return υπονοείται)
- Αν έχουμε περισσότερες από μία εντολές ή θέλουμε να γράψουμε τη λέξη return, τότε χρειάζονται άγκιστρα

Interface Predicate (1)

- Στο προηγούμενο παράδειγμα γράψαμε το interface ITeacherIDFilter
- Κάτι τέτοιο δεν είναι απαραίτητο, καθώς η Java 8 και πάνω μας παρέχει το **interface Predicate στο package java.util.function**
- Η μέθοδος ***test(T t)*** παίρνει ως παράμετρο μία generic class (στην περίπτωσή μας Teacher) και επιστρέφει ένα boolean. Αυτό είναι ότι χρειαζόμαστε για να κάνουμε επιλογές (filtering)
- Χρησιμοποιώντας το Predicate functional interface μας επιτρέπει να περάσουμε lambda expression ως παράμετρο, οπουδήποτε έχουμε Predicate

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

Interface Predicate (2)


```
65    @     public static List<Teacher> getFilteredTeachers(List<Teacher> teachers, Predicate<Teacher> filter) {
66              List<Teacher> teachersToReturn = new ArrayList<>();
67              for (Teacher teacher : teachers) {
68                  if (filter.test(teacher)) {
69                      teachersToReturn.add(teacher);
70                  }
71              }
72              return teachersToReturn;
73          }
```

- Το interface Predicate δίνει out-of-the-box την μέθοδο test, που μπορούμε να χρησιμοποιήσουμε όπως παραπάνω

Interface Consumer (1)

- Αντί επίσης να έχουμε την for για την εκτύπωση των στοιχείων των καθηγητών μπορούμε να χρησιμοποιήσουμε το functional interface Consumer, που είναι πιο γενικό και μπορούμε να περάσουμε ως παράμετρο οποιαδήποτε τελική action

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

Interface Consumer (2)


```
77  @   public static void printTeachers(List<Teacher> teachers, Consumer<Teacher> consumer) {
78          for (Teacher teacher : teachers) {
79              consumer.accept(teacher);
80          }
81      }
```

```
18      public static void main(String[] args) {
19          List<Teacher> filteredTeachers = getFilteredTeachers(teachers, teacher -> teacher.getTeacherId() == 3);
20          printTeachers(filteredTeachers, teacher -> Teacher.printTeacher(teacher));
21          printTeachers(filteredTeachers, teacher -> teacher.printFullName());
22          printTeachers(filteredTeachers, teacher -> System.out.println(teacher));
23
```

- Στη θέση του Consumer έχουμε lambdas τα οποία όμως καλούνε ήδη έτοιμες μεθόδους (**printTeacher** που είναι static, **printFullName** που είναι instance method, **println** που είναι instance method)
- Και στις τρεις περιπτώσεις, το αντικείμενο teacher εμφανίζεται δύο φορές και αριστερά και δεξιά από το arrow

Method Reference (1)


```
18    public static void main(String[] args) {
19        List<Teacher> filteredTeachers = getFilteredTeachers(teachers, teacher -> teacher.getTeacherId() == 3);
20        printTeachers(filteredTeachers, Teacher::printTeacher);
21        printTeachers(filteredTeachers, Teacher::printFullName);
22        printTeachers(filteredTeachers, System.out::println);
23
```

- Τα method references είναι πιο προηγμένη εναλλακτική των lambda expressions όταν έχουμε έτοιμες μεθόδους και όταν εμφανίζεται δεξιά και αριστερά το ίδιο object

Method Reference (2)


```
18    public static void main(String[] args) {
19        List<Teacher> filteredTeachers = getFilteredTeachers(teachers, teacher -> teacher.getTeacherId() == 3);
20        teachers.forEach(Teacher::printTeacher);
21        teachers.forEach(Teacher::printFullName);
22        teachers.forEach(System.out::println);
```

- Αντί ειδικής μεθόδου για εκτύπωση η Java για Collections μας παρέχει την forEach η οποία παίρνει ως παράμετρο interface Consumer

Method Reference (3)

- Method references μπορούμε να έχουμε ως:

1. Reference σε static method.
2. Reference σε instance method.
3. Reference σε constructor

ForEach


```
26 @    public static void printEqualID(List<Teacher> teachers, Predicate<Teacher> checker, Consumer<Teacher> teacherPrint) {
27          teachers.forEach((teacher) -> {
28                  if (checker.test(teacher)) teacherPrint.accept(teacher);
29          });
30      }
```

- Η *ForEach* δέχεται ως παράμετρο ένα lambda που χρησιμοποιείται για την επεξεργασία κάθε στοιχείου του Collection
- Ο τύπος της *forEach* είναι Consumer interface

Άσκηση

- Κάντε το ίδιο για μία κλάση Product που περιέχει id, title, price, quantity και αφού αναπτύξετε μία μέθοδο έστω getFilteredProducts() για να φιλτράρετε μία List<Product> στη συνέχεια σε μία main() εφαρμόστε την μέθοδο αυτή για να φιλτράρετε τα προϊόντα με διάφορους τρόπους (price, quantity, κλπ)