Μεθοδολογία Προγραμματισμού

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Νικόλαος Πεταλίδης

Τμήμα Μηχανικών Πληροφορικής και Επικοινωνιών
Διεθνές Πανεπιστήμιο της Ελλάδος

Εισαγωγή
Εαρινό Εξάμηνο

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Συναρτησιακός προγραμματισμός

- Είναι ένας τρόπος προγραμματισμού όπου το αποτέλεσμα ενός προγράμματος προκύπτει μέσα από την εφαρμογή μίας ή περισσότερων συναρτήσεων
- Σε αντίθεση με τον προστακτικό (imperative) τρόπο προγραμματισμού όπου το αποτέλεσμα ενός προγράμματος προκύπτει από την αλλαγή της τιμής διάφορων μεταβλητών
- Στη Java υποστηρίζεται από την έκδοση 1.8 και μετά

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Παράδειγμα προστακτικού προγραμματισμού

- Έστω ότι θέλουμε να βρούμε αν η πόλη Chicago είναι σε μια λίστα από πόλεις που μας έχει δοθεί

Προστακτικός προγραμματισμός

```
boolean found = false;
for (String city: cities) {
   if (city.equals("Chicago")) {
      found = true;
      break;
   }
}
System.out.println("Found chicago?:" + found);
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Παράδειγμα συναρτησιακού προγραμματισμού

Συναρτησιακός προγραμματισμός

```
System.out.println("Found chicago?"
  + cities.contains("Chicago"));
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Διαφορές

- Δε χρειαστήκαμε μεταβλητές που αλλάζουν τιμές (found)
- Δε χρειαστήκαμε την επανάληψη
- Λιγότερος κώδικας, αντικατοπτρίζει καλύτερα το τί θέλουμε από το πώς θα το πετύχουμε

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Ένα άλλο παράδειγμα

Έστω ότι θέλουμε να αθροίσουμε τις τιμές ενός πίνακα, που είναι μεγαλύτερες από 20, βάζοντας σε κάθε μία από αυτές έκπτωση 10%. Θα μπορούσαμε να έχουμε τον πιο κάτω πίνακα με τις τιμές:

```
final List<BigDecimal> prices = Arrays.asList(
  new BigDecimal("10"), new BigDecimal("30"),
  new BigDecimal("17"), new BigDecimal("20"),
  new BigDecimal("15"),new BigDecimal("18"),
  new BigDecimal("45"), new BigDecimal("12"));
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Λύση με προστακτικό προγραμματισμό

```
BigDecimal totalOfDiscountedPrices = BigDecimal.ZERO;

for (BigDecimal price: prices) {
  if (price.compareTo(BigDecimal.valueOf(20))>0)
    totalOfDiscountedPrices =
      totalOfDiscountedPrices.add(
          price.multiply(BigDecimal.valueOf(0.9)));
 }

 System.out.println("Total of discounted prices: " +
     totalOfDiscountedPrices);
```

Λύση με συναρτησιακό προγραμματισμό

```
final BigDecimal totalOfDiscountedPrices =
    prices.stream()
        .filter(price ->
                price.compareTo(BigDecimal.valueOf(20)) > 0)
        .map(price ->
                price.multiply(BigDecimal.valueOf(0.9)))
        .reduce(BigDecimalZero, BigDecimal::add);

System.out.println("Total of discounted prices: " +
        totalOfDiscountedPrices);
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Μικρές επεξηγήσεις

- Οι συναρτήσεις stream (), filter (), map (), reduce () προστέθηκαν στη Java 1.8 και επιδρούν στα στοιχεία μιας συλλογής (Collection)
- Ειδικότερα οι filter (), map (), reduce () είναι συναρτήσεις *υψηλότερης τάξης* μια που όπως φαίνεται δέχονται ως παράμετρο άλλες συναρτήσεις

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Συλλογές

Αρκετές από τις ευκολίες που μας δίνει η προσθήκη των νέων μηχανισμών της Java αφορούν στη διαχείριση συλλογών. Πιο συγκεκριμένα έχουμε νέους τρόπους

- Να διατρέξουμε τα στοιχεία μιας συλλογής
- Να μετατρέψουμε τα στοιχεία της
- Να επιλέξουμε στοιχεία της
- Να ενώσουμε στοιχεία

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Διατρέχοντας τα στοιχεία μιας λίστας

Υπάρχει μια νέο μέθοδος forEach () στο Iteratable interface. Η μέθοδος αυτή δέχεται ως παράμετρο ένα αντικείμενο τύπου Consumer το οποίο υλοποιεί μια μέθοδο accept (). Αν θέλαμε να τυπώσουμε τα στοιχεία αυτής της λίστας:

```
final List<String> friends = Arrays.asList("Brian",
                "Nate", "Neal", "Raju", "Sara", "Scott");
```

θα μπορούσαμε να το κάνουμε έτσι:

```
friends.forEach(new Consumer<String>() {
   public void accept(final Sting name) {
      System.out.println(name);
      }
});
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Τι βελτιώσαμε;

Η μοναδική βελτίωση που κάναμε ήταν ότι εξαλείψαμε το βρόγχο **for**, και πλέον ορίζουμε μόνο το τι θέλουμε να κάνουμε με τα στοιχεία της λίστας, αντί να ορίζουμε και πώς θα τη διατρέξουμε.
Ο κώδικας όμως είναι δυσανάγνωστος. Έχουμε χρησιμοποιήσει το συντακτικό των ανώνυμων κλάσεων της Java το οποίο είναι ομολογουμένως δύσκολο.

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Εκφράσεις λάμδα

Αντί της χρήσης του δυσανάγνωστου συντακτικού των ανώνυμων κλάσεων θα μπορούσαμε να χρησιμοποιήσουμε ένα lambda expression και να γράφαμε τον κώδικα απλώς ως εξής:

```
friends.forEach((final String name)
    -> System.out.println(name));
```

Ένα lambda expression είναι ουσιαστικά μια συνάρτηση χωρίς όνομα. Στο συγκεκριμένο παράδειγμα η συνάρτηση αυτή παίρνει ως παράμετρο ένα String και το τυπώνει στην κονσόλα.

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Απλουστεύσεις

Ο compiler της Java είναι αρκετά έξυπνος και μπορεί πολλές φορές να καταλάβει τον τύπο της μεταβλητής. Έτσι ο ακόλουθος κώδικας είναι επίσης σωστός:

```
friends.forEach((name)
    -> System.out.println(name));
```

ή και χωρίς τις παρενθέσεις

```
friends.forEach(name
    -> System.out.println(name));
```

ή

```
friends.forEach(System.out::println);
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Μετασχηματίζοντας λίστες

Έστω ότι θέλουμε να αλλάξουμε τη λίστα των ονομάτων του προηγούμενου παραδείγματος σε κεφαλαίο. Μια παραδοσιακή προσέγγιση θα ήταν η ακόλουθη:

```
final List<String> uppercaseNames = new ArrayList<>();
for (String name: friends) {
   uppercaseNames.add(name.toUpperCase());
}
```

Χρησιμοποιώντας lambda expressions θα μπορούσαμε να το γράψουμε ως εξής:

```
final List<String> uppercaseNames = new ArrayList<>();
friends.forEach(name->
   uppercaseNames.add(name.toUpperCase()));
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Προβλήματα με την προηγούμενη προσέγγιση

Το βασικό πρόβλημα προς μια καθαρά συναρτησιακή προσέγγιση είναι η χρήση της μεταβλητής uppercaseNames η οποία μεταβάλλεται συνεχώς. Αντ' αυτού μπορούμε να χρησιμοποιήσουμε το νέο Stream interface και τις μεθόδους του

```
friends.stream().
    map(name->name.toUpperCase())
    .foreach(//Do whatever you want here)
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Stream

- Η μέθοδος stream () μετατρέπει ένα Collection σε Stream
- Η μέθοδος map() εφαρμόζει τον κώδικα μέσα στις παρενθέσεις σε κάθε στοιχείο της συλλογής και *δημιουργεί μια νέα συλλογή με αυτά τα στοιχεία*.

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Φιλτράροντας στοιχεία

Έστω ότι θέλουμε από τη λίστα με τα ονόματα να βρούμε όσα ξεκινάνε από N. Η συναρτησιακή προσέγγιση θα ήταν η ακόλουθη

```
final List<String> startsWithN = friends.stream().
    filter(name->name.startsWith("N"))
    .collect(Collectors.toList());
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Filter

- Η μέθοδος filter () μοιάζει με τη map γιατί επιστρέφει ένα νέο collection
- Περιμένει ένα lambda expression που επιστρέφει **boolean**
- Σε αντίθεση με τη map() το collection που θα επιστρέψει μπορεί να έχει λιγότερα στοιχεία από το αρχικό
- Ένα στοιχείο προστίθεται στο collection που επιστρέφει, όταν το lamdba expression επιστρέψει true

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Collect

- Η μέθοδος collect () μαζεύει τα στοιχεία από ένα stream και τα συλλέγει σε ένα νέο collection
- Χρησιμοποιείται συχνά για να αποθηκευθούν τα αποτελέσματα μιας έκφρασης σε μια μεταβλητή
- Προσφέρει περισσότερες δυνατότητας παράλληλης εκτέλεσης, από ό,τι αν προσπαθούσαμε μόνοι μας να δημιουργήσουμε το νέο collection

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Επαναχρησιμοποίηση lambda expressions

- Μια έκφραση lambda μπορεί να αποθηκευθεί σε μια μεταβλητή και να επαναχρησιμοποιηθεί.
- Για παράδειγμα:

```
final Predicate<String> startsWithN =
    name -> name.startsWith("N");

final long countAList = aList.stream()
                             .filter(startsWithN)
                             .count();
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Παραμετροποίηση σε μεγαλύτερο βαθμό

- Στο προηγούμενο παράδειγμα ελέγχουμε μόνο για το γράμμα N
- Θα θέλαμε περισσότερη ευελιξία και να ορίζουμε εμείς το γράμμα
- Θα μπορούσαμε να το ορίσουμε με αυτόν τον τρόπο

```
public static Predicate<String>
    checkIfStartsWith(final String letter)
{
    return name->name.startsWith(letter);
}
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Closure και lexical scoping

- Στο προηγούμενο παράδειγμα επιστρέφουμε ένα lambda expression
- name είναι η παράμετρος που παίρνει το lambda expression
- Τι είναι το letter ;

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Closure και lexical scoping

- Η μεταβλητή letter δεν ορίζεται στο lambda expression αλλά στον ορισμό του lambda expression
- Η java θα πάρει την τιμή της από εκεί. Αυτό είναι το lexical scoping ή closure
- Με τον τρόπο αυτό αποθηκεύουμε τιμες σε ένα σημεί για να τα χρησιμοποιήσουμε αργότερα

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Παράδειγμα

```
final long countFriendsStartN =
    friends.stream()
    .filter(checkIfStartsWith("N")).count();
```

Εισαγωγή στο συναρτησιακό προγραμματισμό με Java

Βιβλιογραφία

Οι διαφάνειες είναι από το

Venkat Subramaniam, "Functional Programing in Java", The Pragmatic Programmers, 2014