Gyakorlat_kidolgozas

November 29, 2025

0.1 JSON beolvasás

0.1.1 Importok

Adatok.class

- import com.fasterxml.jackson.annotation.JsonFormat;
- import com.fasterxml.jackson.annotation.JsonIgnore;
- import com.fasterxml.jackson.annotation.JsonProperty;
- import com.fasterxml.jackson.annotation.JsonPropertyOrder;
- import java.time.LocalDate; // Akkor kell ha van dátum

Runner.class

- import com.fasterxml.jackson.core.type.TypeReference;
- import com.fasterxml.jackson.databind.JsonNode;
- import com.fasterxml.jackson.databind.ObjectMapper;
- import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
- import java.io.File;
- import java.io.IOException;
- import java.util.ArrayList;
- import java.util.Comparator;

0.1.2 Annotációk

@JsonPropertyOrder({“v1”, “v2”}) Íráskor ebben a sorrendben írja ki az elemeket. A konstruktor felé kell írni.

[ ]:
```
@JsonPropertyOrder({"modellszam", "meret","szin","evszak","gyartasDatuma"})
public class Cipo {}
```

@JsonFormat(pattern = ””) Az érték formátumát lehet megadni, dátumoknál fontos. A dátumot tároló változó fölé kell írni.

[ ]:
```
@JsonFormat(pattern = "dd/MM/yyyy") //29/11/2025
@JsonFormat(pattern = "yyyy/MM/dd") //2025/11/29
@JsonFormat(pattern = "yyyy.MM.dd") //2025.11.29
@JsonFormat(pattern = "MM-dd-yyy") //11-29-2025
```

@JsonProperty(“v1”) Összeköti a Java mezőt vagy metódust a JSOn kulcs nevével.

[ ]:
```
@JsonProperty("evszak")
public String getEvszak() {
    return evszak;
}
```

0.1.3 Beolvasás

[ ]:
```
public class Runner {
    static void main() throws IOException {
        ObjectMapper mapper = new ObjectMapper();
        mapper.registerModule(new JavaTimeModule());

        JsonNode root = mapper.readTree(new File("C:/Users/Petra/IdeaProjects/
 ↪ElozoZH/src/cipo.json"));
        JsonNode arr = root.path("adatok");

        ArrayList<Cipo> cipok = mapper.readValue(
                arr.traverse(mapper),
                new TypeReference<ArrayList<Cipo>>() { }
        );
        cipok.stream().forEach(System.out::println);
   }
}
```

0.2 Stream

[6]:
```
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
class Person {

    public enum Sex { MALE, FEMALE }

    String name;
    LocalDate birthday;
    Sex gender;
    String emailAddress;

    public Person(String name, LocalDate birthday, Sex gender, String␣
 ↪emailAddress) {
        this.name = name;
        this.birthday = birthday;
        this.gender = gender;
        this.emailAddress = emailAddress;
    }
```

```
    public int getAge() {
        // Egyszerű életkor-számítás: év különbség (nem napra pontos, de␣
 ↪példához elég)
        return LocalDate.now().getYear() - birthday.getYear();
    }

    public String getName() {
        return name;
    }

    public Sex getGender() {
        return gender;
    }

    public String getEmailAddress() {
        return emailAddress;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", birthday=" + birthday +
                ", gender=" + gender +
                ", emailAddress='" + emailAddress + '\'' +
                '}';
    }
}
```

ArrayList feltöltése tetszőleges adatokkal

[8]:
```
List<Person> roster = new ArrayList<>();
        roster.add(new Person("Anna", LocalDate.of(2000, 5, 10), Person.Sex.
 ↪FEMALE, "anna@example.com"));
        roster.add(new Person("Béla", LocalDate.of(1995, 3, 20), Person.Sex.
 ↪MALE, "bela@example.com"));
        roster.add(new Person("Csaba", LocalDate.of(1988, 11, 2), Person.Sex.
 ↪MALE, "csaba@example.com"));
        roster.add(new Person("Dóra", LocalDate.of(2003, 1, 15), Person.Sex.
 ↪FEMALE, "dora@example.com"));
```

[8]: true

0.2.1 .forEach()

[9]:
```
//Csak egy tulajdonság kiírása:
roster.stream().forEach(p -> System.out.print(p.getName() + " "));
```

```
Anna Béla Csaba Dóra
```

[10]:
```
// Összes tulajdonság kiírása:
roster.stream().forEach(System.out::println);
```

```
Person{name='Anna', birthday=2000-05-10, gender=FEMALE,
emailAddress='anna@example.com'}
Person{name='Béla', birthday=1995-03-20, gender=MALE,
emailAddress='bela@example.com'}
Person{name='Csaba', birthday=1988-11-02, gender=MALE,
emailAddress='csaba@example.com'}
Person{name='Dóra', birthday=2003-01-15, gender=FEMALE,
emailAddress='dora@example.com'}
```

0.2.2 .filter()

[ ]:
```
// Nők kiírása:
roster.stream()
    .filter(p -> p.getGender() == Person.Sex.FEMALE) //Nők kiválasztása
    .forEach(System.out::println); // Az összes adat kiírása
```

```
Person{name='Anna', birthday=2000-05-10, gender=FEMALE,
emailAddress='anna@example.com'}
Person{name='Dóra', birthday=2003-01-15, gender=FEMALE,
emailAddress='dora@example.com'}
```

[ ]:
```
// Nők neveinek kiírása:
roster.stream()
    .filter(p -> p.getGender() == Person.Sex.FEMALE) //Nők kiválasztása
    .forEach(p -> System.out.println(p.getName())); // Csak a név kiírása
```

```
Anna
Dóra
```

[ ]:
```
// Férfiak átlagos életkora:
double averageAgeOfMales = roster.stream()
    .filter(p -> p.getGender() == Person.Sex.MALE) // Férfiak meghatározása
    .mapToInt(Person::getAge) // Az objektumból egész számot csinál és átvált␣
 ↪IntStream-re
    .average() // Átlagszámítás
    .orElse(0.0); // Ha nincs érvényes kor akkor 0
System.out.println("Átlagéletkor (férfiak): " + averageAgeOfMales);
```

```
Átlagéletkor (férfiak): 33.5
```

0.2.3 map VS mapToInt

map: Bekéri az objektumot, és utána kell hozzá egy reduce függvény, ahol a .reduce() - 1. arg: kezdőérték pl. 0 - 2. arg: kifejezés, mi történjen pl. (a, b) -> a + b összeadja az értékeket vagy (a, b) -> a - b kivonja az értékeket

mapToInt: // Az objektumból egész számot csinál és átvált IntStream-re, ezután lehet használni matematikai függvényeket pl. sum, avarage stb.

[23]:
```
// Mindenki összéletkora
int totalAge = roster.stream()
    .mapToInt(Person::getAge)
    .sum();
System.out.println("Összes életkor (sum): " + totalAge);

Integer totalAgeReduce = roster.stream()
    .map(Person::getAge)
    .reduce(0, (a, b) -> a + b);
System.out.println("Összes életkor (reduce): " + totalAgeReduce);
```

```
Összes életkor (sum): 114
Összes életkor (reduce): 114
```

0.2.4 .collect

[26]:
```
//Egyféle adat kiírása
List<String> namesOfMaleMembers = roster.stream()
    .filter(p -> p.getGender() == Person.Sex.MALE)
    .map(Person::getName)
    .collect(Collectors.toList());
namesOfMaleMembers.forEach(System.out::println);
```

```
Béla
Csaba
```

[27]:
```
//Többféle adat kiírása
List<String> infos = roster.stream()
    .filter(p -> p.getGender() == Person.Sex.MALE)
    .map(p -> p.getName() + ", " + p.getAge() + ", " + p.getEmailAddress())
    .collect(Collectors.toList());

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

```
Béla, 30, bela@example.com
Csaba, 37, csaba@example.com
```

[28]:
```
Map<Person.Sex, List<Person>> byGender = roster.stream()
        .collect(Collectors.groupingBy(Person::getGender));
byGender.forEach((gender, people) -> {
    System.out.println(gender + " -> " + people);
```

```
});
```

```
MALE -> [Person{name='Béla', birthday=1995-03-20, gender=MALE,
emailAddress='bela@example.com'}, Person{name='Csaba', birthday=1988-11-02,
gender=MALE, emailAddress='csaba@example.com'}]
FEMALE -> [Person{name='Anna', birthday=2000-05-10, gender=FEMALE,
emailAddress='anna@example.com'}, Person{name='Dóra', birthday=2003-01-15,
gender=FEMALE, emailAddress='dora@example.com'}]
```

[31]:
```
//Ha csak egyféle adatra akarjuk a listát
System.out.println(
    roster.stream()
          .filter(p -> p.getGender() == Person.Sex.MALE)
          .collect(Collectors.toList())
);
```

```
[Person{name='Béla', birthday=1995-03-20, gender=MALE,
emailAddress='bela@example.com'}, Person{name='Csaba', birthday=1988-11-02,
gender=MALE, emailAddress='csaba@example.com'}]
```

[32]:
```
//Ha csak többféle adatra akarjuk a listát
System.out.println(
    roster.stream()
          .filter(p -> (p.getGender() == Person.Sex.MALE || p.getGender() ==␣
 ↪Person.Sex.FEMALE))
          .collect(Collectors.toList())
);
```

```
[Person{name='Anna', birthday=2000-05-10, gender=FEMALE,
emailAddress='anna@example.com'}, Person{name='Béla', birthday=1995-03-20,
gender=MALE, emailAddress='bela@example.com'}, Person{name='Csaba',
birthday=1988-11-02, gender=MALE, emailAddress='csaba@example.com'},
Person{name='Dóra', birthday=2003-01-15, gender=FEMALE,
emailAddress='dora@example.com'}]
```

[36]:
```
Map<Person.Sex, List<String>> namesByGender = roster.stream()
    .collect(Collectors.groupingBy(
            Person::getGender,
            Collectors.mapping(Person::getName, Collectors.toList())
    ));
namesByGender.forEach((gender, names) -> {
    System.out.println(gender + " -> " + names);
});
```

```
MALE -> [Béla, Csaba]
FEMALE -> [Anna, Dóra]
```

[39]:
```
Map<Person.Sex, Integer> totalAgeByGender = roster.stream()
        .collect(Collectors.groupingBy( //Csoportosítunk
```

```
                Person::getGender, //Nem szerint
                Collectors.reducing(
                        0, // Kezdőérték
                        Person::getAge, //Mit szeretnénk
                        Integer::sum //összeadni
                )
        ));
totalAgeByGender.forEach((gender, sumAge) -> {
    System.out.println(gender + " -> " + sumAge);
});
```

```
MALE -> 67
FEMALE -> 47
```

[41]:
```
Map<Person.Sex, Double> averageAgeByGender = roster.stream()
        .collect(Collectors.groupingBy( //Csoportosítunk
                Person::getGender, //Mi szerint
                Collectors.averagingInt(Person::getAge) //Átlagszámítás
        ));
averageAgeByGender.forEach((gender, avgAge) -> {
    System.out.println(gender + " -> " + avgAge);
});
```

```
MALE -> 33.5
FEMALE -> 23.5
```