2018
Programmazione a oggetti
Esempi files e stream

Giorgio Bruno

Dip. Automatica e Informatica
Politecnico di Torino
email: giorgio.bruno@polito.it

Lista di appelli

Si definisca una lista di appelli (listaAppelli).

Un appello contiene il nome del corso, la data (LocalDate) e il n. di studenti prenotati.

Si inizializzi la lista con questi appelli.

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40

Esempio di toString di un appello: matematica,2016-06-22,30

Classe Appello

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40

```
public class Appello {
private String corso;
private LocalDate data;
private int n;
public Appello(String corso, LocalDate data, int n) {
        this.corso = corso; this.data = data; this.n = n;}
public String toString() {return corso + "," + data + "," + n;}
```

+ getters

main con lista degli appelli

```
List<Appello> listaAppelli = new ArrayList<>();
listaAppelli.add(new Appello("matematica", LocalDate.parse("2016-06-22"), 30));
listaAppelli.add(new Appello("storia", LocalDate.parse("2016-06-12"), 20));
listaAppelli.add(new Appello("geografia", LocalDate.parse("2016-06-22"), 40));
```

Main

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40

Si produca una nuova lista ordinata per date crescenti e corsi crescenti.

List<Appello> listaAppelliOrdinata = listaAppelli.stream().
?????

Si produca una stringa come questa dalla lista ordinata
[storia 2016-06-12, geografia 2016-06-22, matematica 2016-06-22]

Main

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40

Si produca una **nuova lista** ordinata per date crescenti e corsi crescenti

```
List<Appello> listaAppelliOrdinata = listaAppelli.stream()
.sorted(comparing(Appello::getData).thenComparing(Appello::getCorso))
.collect(toList());
```

Si produca una **stringa** come questa
[storia 2016-06-12, geografia 2016-06-22, matematica 2016-06-22]

```
String strListaAppelliOrdinata = listaAppelliOrdinata.stream()
.map(a -> {return a.getCorso() + " " + a.getData();})
.collect(joining(", ", "[", "]"));
System.out.println(strListaAppelliOrdinata);”
```

Main

Si produca una mappa dei corsi ordinati per date crescenti

SortedMap<LocalDate, List<String>> mappa1 = listaAppelli.stream()
?????

risultato

{2016-06-12=[storia], 2016-06-22=[geografia, matematica]}

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40

Main

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40

Si produca una mappa dei corsi ordinati per date crescenti

```
SortedMap<LocalDate, List<String>> mappa1 = listaAppelli.stream()
.sorted(comparing(Appello::getCorso))
.collect(groupingBy(Appello::getData, TreeMap::new,
        mapping(Appello::getCorso, toList())));
System.out.println(mappa1);
```

{2016-06-12=[storia], 2016-06-22=[geografia, matematica]}

Lettura da file

Si legga il file seguente (appelli.txt)

```
matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 30
italiano 2016-06-02 20
matematica 2016-07-07 25
italiano 2016-07-12 15
```

e si produca una lista di appelli.

La stampa della lista produce questo risultato.

[matematica,2016-06-22,30, storia,2016-06-12,20, geografia,2016-06-22,40, storia,2016-06-22,30, italiano,2016-06-02,20, matematica,2016-07-07,25, italiano,2016-07-12,15]

Nota: si aggiunga il metodo

Appello genAppello(String linea)

per generare un appello dalla linea contenente corso, data e n.iscritti

Lettura del file

```
try(BufferedReader in = new BufferedReader(new FileReader("appelli.txt"))) {
        List<Appello> listaAppelliDaFile = in.lines()
                    //.peek(linea -> {System.out.println(linea);})
                    .map(linea -> Appello.genAppello(linea))
                    .collect(toList());
        System.out.println(listaAppelliDaFile);
   } catch (Exception e) {e.printStackTrace();
}
```

[matematica,2016-06-22,30, storia,2016-06-12,20, geografia,2016-06-22,40, storia,2016-06-22,30, italiano,2016-06-02,20, matematica,2016-07-07,25, italiano,2016-07-12,15]

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 30
italiano 2016-06-02 20
matematica 2016-07-07 25
italiano 2016-07-12 15

Metodo genAppello

public class Appello {

private String corso; private LocalDate data; private int n;

```
public static Appello genAppello(String linea) {
Scanner s = new Scanner(linea);
Appello appello = null;
try {
        String corso = s.next();
        LocalDate data = LocalDate.parse(s.next());
        int n = s.nextInt();
        appello = new Appello(corso, data, n);
} catch (Exception e) {System.out.println(linea + " linea errata");}

finally {s.close();}
return appello;
}
}
```

Lettura da file con errori

Si legga il file seguente (appelliConErrori.txt)

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 30
italiano 2016-06-2 20
matematica 2016-07-07 25
italiano 2016-07-12 15

e si produca una lista di appelli

saltando le linee errate.

La stampa della lista produce questo risultato.

[matematica,2016-06-22,30, storia,2016-06-12,20, geografia,2016-06-22,40, storia,2016-06-22,30, matematica,2016-07-07,25, italiano,2016-07-12,15]

Lettura da file con errori

```
try(BufferedReader in = new BufferedReader(new
            FileReader("appelliConErrori.txt")))  {

    List<Appello> listaAppelliDaFile = in.lines()
    .map(linea -> Appello.genAppello(linea))
    .filter(linea -> {return linea != null;} )
    .collect(toList());
System.out.println(listaAppelliDaFile);
    } catch (Exception e) {e.printStackTrace();
}
```

```
matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 30
italiano 2016-06-2 20
matematica 2016-07-07 25
italiano 2016-07-12 15
```

Lettura da file

Si legga il file seguente (G:\\appelli.txt)

```
matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 30
italiano 2016-06-02 20
matematica 2016-07-07 25
italiano 2016-07-12 15
```

e si produca una lista di appelli.

Si usi la classe Files del package nio.file.

Lettura da file

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 30
italiano 2016-06-02 20
matematica 2016-07-07 25
italiano 2016-07-12 15

Si legga il file seguente (appelli.txt)
e si produca una lista di appelli.
Si usi la classe Files del package nio.

```
import java.nio.file.*;
…
Path fileP = Paths.get("G:\\appelli.txt");
List<Appello> listaAppelliDaFile1 = Files.lines(fileP)
    .map(linea -> Appello.genAppello(linea))
    .filter(linea -> {return linea != null;} )
    .collect(toList());
System.out.println(listaAppelliDaFile1);
```

[matematica,2016-06-22,30, storia,2016-06-12,20, geografia,2016-06-22,40, storia,2016-06-22,30, italiano,2016-06-02,20, matematica,2016-07-07,25, italiano,2016-07-12,15]

Numero medio degli iscritti agli appelli

```
//stampare la media con due decimali
double mediaIscritti = listaLettaDaFile.stream()




System.out.println(String.format("%.2f", mediaIscritti));
```

25,71

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 30
italiano 2016-06-02 20
matematica 2016-07-07 25
italiano 2016-07-12 15

Numero medio degli iscritti agli appelli

```
double mediaIscritti = listaLettaDaFile.stream()
        .mapToInt(Appello::getN)
        .average().orElse(0.0); // se non ci sono iscritti
System.out.println(String.format("%.2f", mediaIscritti));
```

25,71

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 30
italiano 2016-06-02 20
matematica 2016-07-07 25
italiano 2016-07-12 15

Numero di iscritti per corso con i corsi ordinati

Nota: occorre sommare gli iscritti agli appelli dello stesso corso.

```
SortedMap<String, Integer> iscrittiPerCorso = listaLettaDaFile.stream()
```

```
matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 30
italiano 2016-06-02 20
matematica 2016-07-07 25
italiano 2016-07-12 15
```

```
System.out.println(iscrittiPerCorso);
{geografia=40, italiano=35, matematica=55, storia=50}
```

Numero di iscritti per corso con i corsi ordinati

```
SortedMap<String, Integer> iscrittiPerCorso = listaLettaDaFile.stream()
 .collect(groupingBy(Appello::getCorso, TreeMap::new,
                     summingInt(Appello::getN)));
System.out.println(iscrittiPerCorso);
```

```
{geografia=40, italiano=35, matematica=55, storia=50}
```

Corsi ordinati per numero di iscritti decrescente

```
SortedMap<Integer, List<String>> iscrittiPerCorso1 =
listaLettaDaFile1.stream()
        .collect(groupingBy(Appello::getCorso,
                            TreeMap::new,
                            summingInt(Appello::getN)))
        .entrySet().stream()
```

Prima si contano gli iscritti per corsi ordinati.

Poi si raggruppano i corsi per n. di iscritti decrescente.

```
{55=[matematica], 40=[geografia, storia], 35=[italiano]}
```

matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 20
italiano 2016-06-02 20
matematica 2016-07-07 25
italiano 2016-07-12 15

Corsi ordinati per numero di iscritti decrescente

```
SortedMap<Integer, List<String>> iscrittiPerCorso1 =
listaLettaDaFile1.stream()
        .collect(groupingBy(Appello::getCorso,
                            TreeMap::new,
                            summingInt(Appello::getN)))
        .entrySet().stream()
        .collect(groupingBy(e -> e.getValue(),
                ()-> new TreeMap<>(reverseOrder()),
                mapping(e -> e.getKey(), toList())));
```

Prima si contano gli iscritti per corsi ordinati.

Poi si raggruppano i corsi per n. di iscritti decrescente.

Nota: entry: String (corso), Integer (n. iscritti)

```
{55=[matematica], 40=[geografia, storia], 35=[italiano]}
```

```
matematica 2016-06-22 30
storia 2016-06-12 20
geografia 2016-06-22 40
storia 2016-06-22 20
italiano 2016-06-02 20
matematica 2016-07-07 25
italiano 2016-07-12 15
```

Lista di stringhe relative agli appelli

Ogni stringa contiene il n. di iscritti di un appello e la data; il n. di iscritti è decrescente e la data è crescente; il n. di iscritti è formattato in 3 cifre.

```
List<String> appelliPerIscritti = listaLettaDaFile.stream()
.sorted(

            )
.map(a -> {                                                    ;})
.collect(toList());
System.out.println(appelliPerIscritti);
```

[40 2016-06-22, 30 2016-06-22, 30 2016-06-22, 25 2016-07-07, 20 2016-06-02, 20 2016-06-12, 15 2016-07-12]

Lista di stringhe relative agli appelli

Ogni stringa contiene il n. di iscritti di un appello e la data; il n. di iscritti è decrescente e la data è crescente; il n. di iscritti è formattato in 3 cifre.

```
List<String> appelliPerIscritti = listaLettaDaFile.stream()
.sorted(comparing(Appello::getN, reverseOrder())
                        .thenComparing(Appello::getData))
.map(a -> {return String.format("%3d", a.getN()) + " " + a.getData();})
.collect(toList());
System.out.println(appelliPerIscritti);
```

```
[ 40 2016-06-22,  30 2016-06-22,  30 2016-06-22,  25 2016-07-07,  20 2016-06-02,
20 2016-06-12,  15 2016-07-12]
```