Introduzione a Java 8

Lambda expressions

L'introduzione delle **lambda expressions** *è*, senza dubbio, il piu' importante cambiamento introdotto in questa nuova versione di Java.

Sintassi :

parametri -> body ;

Esempi :

```
() -> System.out.println("ciao");
(String s) -> System.out.println(s);
(i, j) -> i - j;
s -> System.out.println(s);
```

Confronto con Java 7

Java 7

```
List<Integer> numbers = Arrays.asList(2,4,124,55);

for (int i=0; i < numbers.size(); i++){
     System.out.print(numbers.get(i));
}
```

Java 8

```
List<Integer> numbers = Arrays.asList(2,4,124,55);

numbers.forEach(s -> System.out.print(s+" "));
```

Metodo forEach

```
public interface Iterable<T>{
 public default void forEach(Consumer<? super T> c){
     for (T t: this)
          c.accept();
 }
}
```

Il metodo forEach si occupa di analizzare, uno per uno, tutti gli elementi di una struttura dati che implementi l'interfaccia Iterable. Ad esempio, una Lista o uno Stream definito su una lista. Consumer, invece, *è* un'interfaccia che contiene il metodo astratto **void accept(T t)** ed alcuni default methods;

La principale caratteristica del forEach consiste in una **iterazione interna** e non esterna degli elementi. Finora, infatti, per poter scorrere gli elementi di una lista occorreva specificare sia gli elementi da iterare che il meccanismo di iterazione. Con l'introduzione di questo nuovo metodo, *è* possibile scorrere tutti gli elementi di una lista specificando solamente gli elementi da iterare e non il meccanismo di iterazione.

Stream

Uno **Stream** può essere definito come una sequenza di elementi e non come una struttura dati. Rispetto alle altre strutture dati, infatti, lo Stream non permette una memorizzazione "interna" degli elementi. Ad ogni Stream, infatti, deve essere associato una sorgente dove memorizzare le opportune informazioni.

Le principali operazioni che possono essere eseguite su uno stream sono:

<R> Stream<R> map(Function<? super T, ? extends R> mapper) :
questo metodo applica una determinata funzione a tutti gli elementi dello Stream. R identifica il tipo di Stream, mentre Function è un'interfaccia contenente il metodo astratto R apply(T t) ed alcuni default methods;

Stream<T> filter(Predicate<? super T> predicate) :
questo metodo filtra tutti gli elementi dello Stream in base ad una determinata condizione (predicate) passata come parametro. Predicate è un'interfaccia caratterizzata, similmente a Function, dal metodo astratto boolean test(T t) e da alcuni default methods;

Stream

T reduce(T identity, BinaryOperator<T> accumulator) : questo metodo riduce tutti gli elementi dello Stream ad un'unico valore. In particolare, identity corrisponde al valore iniziale del risultato restituito in output, mentre BinaryOperator rappresenta un'interfaccia che ha l'obiettivo di eseguire un'operazione su due operandi dello stesso tipo;

Optional<T> min(Comparator<? super T> comparator) : questo metodo ritorna il minimo elemento dello Stream utilizzando come funzione di comparazione il metodo compareTo definito, ad esempio, dal body delle lambda expressions. Il valore ritornato come output *è* di tipo Optional, ossia una classe "contenitore" che contiene un valore non nullo;

Optional<T> max(Comparator<? super T> comparator) : questo metodo ritorna il massimo elemento dello Stream utilizzando come funzione di comparazione il metodo compareTo definito, ad esempio, dal body delle lambda expressions. Il valore ritornato come output *è* di tipo Optional, ossia una classe "contenitore" che contiene un valore non nullo;

Esempio: Minimo e Massimo

Esempio tratto dalle slides su Python di Stefano Pergoli

Supponiamo di voler calcolare il massimo ed il minimo di una lista di stringhe. Ecco il codice che andremo ad eseguire:

```
String a = "ciao";
String b = "funzione";
String c = "terzastringa";
List<String> s = new ArrayList<String>();
s.add(a); s.add(b); s.add(c);

System.out.println(s.stream().min((u, j)->u.compareTo(j)).get());

System.out.println(s.stream().max((u, j)->u.compareTo(j)).get());
```

L'output sara' il seguente:

```
ciao
terzastringa
```

compareTo

String

```
public int compareTo(String anotherString)
```

compareTo della classe String provvede a comparare la stringa passata come parametro con la stringa oggetto di invocazione del compareTo.
Se anotherString *è* maggiore dell'altra stringa, allora si ritorna 1.
Se anotherString *è* minore dell'altra stringa, allora si ritorna -1.
Se sono uguali, allora si ritorna 0.

Integer

```
public int compareTo(Integer anotherInteger)
```

compareTo della classe Integer provvede a comparare l'intero passato come parametro con l'intero oggetto di invocazione del compareTo.
Se anotherInteger *è* maggiore dell'altro intero, allora si ritorna 1.
Se anotherInteger *è* minore dell'altro intero, allora si ritorna -1.
Se sono uguali, allora si ritorna 0.

Esempio: Minimo, Massimo e Somma

Esempio tratto dalle slides su Python di Stefano Pergoli

Supponiamo adesso di voler calcolare il massimo ed il minimo di una lista di interi. Inoltre, si vuole calcolare anche la somma di tutti gli elementi contenuti nella lista. Ecco il codice che andremo ad eseguire:

```
Integer d = 10;
Integer e = 40;
Integer f = -5;
List<Integer> i = new ArrayList<Integer>();
i.add(d); i.add(e); i.add(f);

System.out.println(i.stream().min((u, j) ->u.compareTo(j)).get());

System.out.println(i.stream().max((u, j)->u.compareTo(j)).get());

System.out.println(i.stream().reduce(0, (z,u)->z+u));
```

L'output sara' il seguente:

```
-5
40
45
```

Esempio: Varie operazioni sulle liste di interi

Esempio tratto dalle slides su Python di Stefano Pergoli

Supponiamo adesso di voler calcolare tutti gli elementi maggiori di 0, minori di 0, tutti i minori di 0 e sommargli 2 e, infine, calcolare la somma di tutti i numeri minori di 0. Ecco il codice:

```
List<Integer> lista = new ArrayList();

lista.add(-40); lista.add(20);

lista.add(60); lista.add(5);

lista.add(-100); lista.add(10);


lista.stream().filter(s -> s > 0).forEach(s->System.out.print(s + " "));

lista.stream().filter(s -> s < 0).forEach(s->System.out.print(s + " "));

lista.stream().filter(s -> s < 0).map(s -> s+2)
                                 .forEach(s->System.out.print(s + " "));

System.out.println(lista.stream().filter(s->s < 0)
                                 .reduce(0, (z,u) -> z+u));
```

Esempio : Ordinamento di una lista

Supponiamo di voler gestire, tramite una **lista**, un database contenente i pi*ù* rilevanti dati anagrafici (nome, cognome, et*à*) dei dipendenti di una particolare ditta.

L'utente deve avere la possibilit*à* di ordinare i vari dipendenti in base al **nome** (in ordine crescente e/o decrescente), in base al **cognome** (in ordine crescente e/o decrescente) oppure in base all' **et***à* (in ordine crescente o decrescente).

Esempio tratto dalle slides su Python di Stefano Pergoli

Esempio : Ordinamento di una lista

```
public class Person {
private String nome; private String cognome; private Integer eta;

Person(String n, String c, Integer e){
this.nome = n;
this.cognome = c;
this.eta = e;}

String get_nome(){return nome;}

String get_cognome(){return cognome;}

Integer get_eta(){return eta;}

void set_nome(String n){this.nome = n;}

void set_cognome(String c){this.cognome = c;}

void set_eta(Integer e){this.eta = e;}
}
```

Esempio : Ordinamento di una lista

```
public class Elenco implements List<Person>{
 List<Person> lista = new ArrayList<Person>();

 public void do_sort(LessBy less){

  int n = lista.size();

  for (int i=0; i < n; i++){ //BUBBLE SORT
   for(int j=0; j < n-1; j++){
    if (less.lessby(lista.get(j), lista.get(j+1))>0)
              swap(lista, j,j+1);
   }
  }
 }

 private void swap(List<Person> l, int j, int i) {
  Person temp = l.get(j);
  l.set(j, l.get(i));
  l.set(i, temp);
 }
```

Esempio : Ordinamento di una lista

```
public void print(){
 lista.forEach(s -> System.out.print(s.get_cognome() + "␣"));
 System.out.println();
 lista.forEach(s -> System.out.print(s.get_nome() + "␣"));
 System.out.println();
 lista.forEach(s -> System.out.print(s.get_eta() + "␣"));
 System.out.println();
 System.out.println();
}

@Override
public boolean add(Person p) {
 // TODO Auto-generated method stub
 lista.add(p);
 return true;
}

...
}
```

Esempio : Ordinamento di una lista

```
public interface LessBy {

      Integer lessby(Person p1, Person p2);

}


public static void main(String[] args) {
// TODO Auto-generated method stub

 Person b1 = new Person("Giovanni","Verdi",40);
 Person b2 = new Person("Luca","Bianchi",20);;
 Person b3 = new Person("Gianpaolo","Gianpaoli",75);
 Person b4 = new Person("Paolo","Rossi",60);
 Person b5 = new Person("Zeus","Focardi",5);

 Elenco l = new Elenco();
 l.add(b1); l.add(b2); l.add(b3); l.add(b4); l.add(b5);
 l.print();
```

Esempio : Ordinamento di una lista

```
LessByConcrete lbc = new LessByConcrete();
l.do_sort((Person p1, Person p2) -> lbc.lessby(p1, p2));
l.print();
l.do_sort((Person p1, Person p2) -> {
            if (p1.get_eta() < p2.get_eta())       return 1;
            if (p1.get_eta() == p2.get_eta())      return 0;
              else                                 return -1;});
l.print();
l.do_sort((Person p1, Person p2) ->
                p1.get_cognome().compareTo(p2.get_cognome()));
l.print();
l.do_sort((Person p1, Person p2) ->
                p2.get_cognome().compareTo(p1.get_cognome()));
l.print();
l.do_sort((Person p1, Person p2) ->
                p1.get_nome().compareTo(p2.get_nome()));
l.print();
l.do_sort((Person p1, Person p2) ->
                p2.get_nome().compareTo(p1.get_nome()));
l.print();}
}
```

Esempio : Ordinamento di una lista

```
public class LessByConcrete implements LessBy {

  @Override
  public Integer lessby(Person p1, Person p2) {
  // TODO Auto-generated method stub

    if (p1.get_eta() > p2.get_eta())
              return 1;
    if (p1.get_eta() == p2.get_eta())
              return 0;
    return -1;
  }

}
```

Esempio : Ordinamento di una List<T>

L'output, generato dalla classe Main, *è* il seguente:

```
Verdi Bianchi Gianpaoli Rossi Focardi
Giovanni Luca Gianpaolo Paolo Zeus
40 20 75 60 5

Focardi Bianchi Verdi Rossi Gianpaoli
Zeus Luca Giovanni Paolo Gianpaolo
5 20 40 60 75

Gianpaoli Rossi Verdi Bianchi Focardi
Gianpaolo Paolo Giovanni Luca Zeus
75 60 40 20 5

Bianchi Focardi Gianpaoli Rossi Verdi
Luca Zeus Gianpaolo Paolo Giovanni
20 5 75 60 40

Verdi Rossi Gianpaoli Focardi Bianchi
Giovanni Paolo Gianpaolo Zeus Luca
40 60 75 5 20

Gianpaoli Verdi Bianchi Rossi Focardi
Gianpaolo Giovanni Luca Paolo Zeus
75 40 20 60 5

Focardi Rossi Bianchi Verdi Gianpaoli
Zeus Paolo Luca Giovanni Gianpaolo
5 60 20 40 75
```

Esempio : Ordinamento di una List<T>

Domanda

Se la classe Elenco implementasse l'interfaccia List<T> invece di List<Person>, le lambda expressions sarebbero ugualmente ben definite?

```
public class Elenco<T> implements List<T>{

  List<T> lista = new ArrayList<T>();

  public void do_sort(LessBy<T> less){

    int n = lista.size();

    for (int i=0; i < n; i++){ //BUBBLE SORT
      for(int j=0; j < n-1; j++){
        if (less.lessby(lista.get(j), lista.get(j+1))>0)
              swap(lista, j,j+1);
      }
    }
}
```

Esempio : Ordinamento di una List<T>

Domanda

Se la classe Elenco implementasse l'interfaccia List<T> invece di List<Person>, le lambda expressions sarebbero ugualmente ben definite?

```
public class Elenco<T> implements List<T>{

  List<T> lista = new ArrayList<T>();

  public void do_sort(LessBy<T> less){

    int n = lista.size();

    for (int i=0; i < n; i++){ //BUBBLE SORT
      for(int j=0; j < n-1; j++){
        if (less.lessby(lista.get(j), lista.get(j+1))>0)
              swap(lista, j,j+1);
      }
    }
}
```

Esempio : Ordinamento di una List<T>

```
  private void swap(List<T> l, int j, int i) {
  // TODO Auto-generated method stub
   T temp = l.get(j);
   l.set(j, l.get(i));
   l.set(i, temp);
  }

  @Override
  public boolean add(T t) {
  // TODO Auto-generated method stub
   lista.add(t);
   return true;
  }

  public List<T> get_lista(){
   return lista;
  }
  ...
}
```

Esempio : Ordinamento di una List<T>

```
public interface LessBy<T> {
     Integer lessby(T p1, T p2);
}

public class Main {
 public static void print(List<Person> lista){
  lista.forEach(s -> System.out.print(s.get_cognome() + "␣"));
  lista.forEach(s -> System.out.print(s.get_nome() + "␣"));
  lista.forEach(s -> System.out.print(s.get_eta() + "␣"));
 }

 public static void main(String[] args) {
  Person b1 = new Person("Giovanni","Verdi",40);
  Person b2 = new Person("Luca","Bianchi",20);;
  Person b3 = new Person("Gianpaolo","Gianpaoli",75);
  Person b4 = new Person("Paolo","Rossi",60);
  Person b5 = new Person("Zeus","Focardi",5);
```

Esempio : Ordinamento di una List<T>

```
Elenco<Person> l = new Elenco<Person>();
l.add(b1); l.add(b2); l.add(b3); l.add(b4); l.add(b5);
print(l.get_lista());

l.do_sort((Person p1, Person p2)->
                        p1.get_eta().compareTo(p2.get_eta()));
print(l.get_lista());
l.do_sort((Person p1, Person p2) -> {
                if (p1.get_eta() < p2.get_eta())    return 1;
                if (p1. get_eta() == p2.get_eta()) return 0;
                else                                return -1;});

print(l.get_lista());
l.do_sort((Person p1, Person p2)->
                  p1.get_cognome().compareTo(p2.get_cognome()));

print(l.get_lista());
 }
}
```

Esempio : Ordinamento di una List<T>

L'output, generato dalla classe Main, *è* il seguente:

```
Verdi Bianchi Gianpaoli Rossi Focardi
Giovanni Luca Gianpaolo Paolo Zeus
40 20 75 60 5

Focardi Bianchi Verdi Rossi Gianpaoli
Zeus Luca Giovanni Paolo Gianpaolo
5 20 40 60 75

Gianpaoli Rossi Verdi Bianchi Focardi
Gianpaolo Paolo Giovanni Luca Zeus
75 60 40 20 5

Bianchi Focardi Gianpaoli Rossi Verdi
Luca Zeus Gianpaolo Paolo Giovanni
20 5 75 60 40
```

Risposta

Si, le lambda expressions sono ancora ben definite