Programmazione ad Oggetti
Mod. 2

14/6/2024

Studente _____ Matricola _____

1. Si implementino le seguenti funzioni di ordine superiore in Java 8+.

   (a) [2 punti] La prima è una variante della classica funzione `map()`[1] che opera su iteratori anziché su collection. L'iteratore restituito in output deve applicare la funzione `f` a ciascun elemento di tipo `A` fornito dall'iteratore `it`, producendo oggetti di tipo `B`.

   ```
   static <A, B> Iterator<B> mapIterator(Iterator<A> it, Function<A, B> f)
   ```

   (b) [2 punti] La seconda funzione di ordine superiore è semanticamente equivalente al costrutto for-each: ad ogni oggetto di tipo `T` nell'iterable in input viene applicato il consumer `f`.

   ```
   static <T> void forEach(Iterable<T> it, Consumer<T> f)
   ```

   (c) [2 punti] Si implementi un tipo `Pair` parametrico su due tipi `A` e `B`. Si scelga liberamente se fornire una implementazione mutabile o immutabile.

   (d) [3 punti] Si prenda ora in considerazione la seguente firma di funzione:

   ```
   static <A, B> Iterator<B> applyFuns(Iterable<Pair<Function<A, B>, A>> l)
   ```

   Per ogni coppia dell'iterable in input, essa deve applicare la funzione che si trova nella prima componente della coppia all'oggetto di tipo `A` che si trova nella seconda componente.

   Si implementi la suddetta funzione tramite **una singola invocazione** della `mapIterator()` di cui al punto (a).

   (e) [3 punti] Si prenda ora in considerazione la seguente firma di funzione:

   ```
   static <T> void acceptFuns(Iterable<Pair<Consumer<T>, T>> l)
   ```

   Per ogni coppia dell'iterable in input, essa deve applicare la funzione consumer che si trova nella prima componente della coppia all'oggetto di tipo `T` che si trova nella seconda componente.

   Si implementi la suddetta funzione tramite **una singola invocazione** della `forEach()` di cui al punto (b).

   (f) [6 punti] Con riferimento al punto (a) si consideri la seguente firma di funzione:

   ```
   static <A, B> Iterator<Supplier<B>> asyncMapIterator(Iterator<A> it, Function<A, B> f)
   ```

   Essa consiste in una variante asincrona della `mapIterator()` in cui l'applicazione della funzione `f` ad ogni elemento di tipo `A` prodotto dall'iteratore in input deve avere luogo ogni volta in un thread nuovo. In altre parole, ogni computazione della funzione `f` deve avvenire concorrentemente. Il risultato di ciascuna computazione, naturalmente, non può essere ritornato subito dalla `next()` dall'iteratore in output, altrimenti sarebbe necessario attendere la fine di ciascun thread, vanificando ogni forma di concorrenza. Per questo motivo l'iteratore in output produce `Supplier<B>` anziché oggetti di tipo `B`. Sono i `Supplier`[2] che devono attendere la fine dell'esecuzione dei thread.

   Suggerimento: l'implementazione è piuttosto contenuta, non servono datatype di appoggio né metodi ausiliari. Si sfrutti al massimo lo scoping, in particolare le chiusure delle lambda (o delle anonymous class, se si preferisce).

[1] Conosciuta anche con il nome di `transform()` in certe librerie.

[2] Si rammenti che un `Supplier<T>` non è altro che una *functional interface* con un solo metodo `get()` che non ha parametri e ritorna un oggetto di tipo `T`.

(g) [4 punti] Con riferimento al punto (b) si consideri la seguente firma di funzione:

```
static <T> void asyncForEach(Iterable<T> it, Consumer<T> f) {
```

Questa è una variante asincrona della `forEach()`: l'applicazione della funzione `f` ad ogni elemento di tipo `T` contenuto nell'iterable in input deve avere luogo ogni volta in un thread nuovo. In altre parole, ogni computazione della funzione `f` deve avvenire concorrentemente.

Suggerimento: poiché i `Consumer` non hanno risultato, non si pone il problema di attendere la fine delle computazioni dei thread.

2. Si svolgano i serguenti esercizi in linguaggio C++[3].

(a) [8 punti] Si implementi una funzione templatizzata su un tipo `C` che, dato un container STL avente tipo `C`, ne somma tutti gli elementi tramite l'operatore di addizione e ritorna il risultato della sommatoria.

(b) [5 punti] Quali vincoli sul template parameter `C` sono implicitamente richiesti? Si scrivano dettagliatamente tutti i vincoli per `C` e per gli eventuali member type utilizzati nell'implementazione.

Esempio: il tipo `C` deve avere il tal metodo, deve supportare il tal costruttore, il tal operatore ecc.; poi deve avere un certo member type e quest'ultimo deve supportare il tal metodo, operatore ecc.

Question:	1	2	Total
Points:	22	13	35
Bonus Points:	0	0	0
Score:			

[3] Non è necessaria nessuna revisione recente del linguaggio, l'esercizio è esprimibile in C++ vanilla, detto C++03.