Programmazione ad Oggetti
Mod. 2

1/7/2022

Studente _____ Matricola _____

1. Realizziamo in Java 8+ una sottoclasse di `java.util.ArrayList` di nome `SkippableArrayList` parametrica su un tipo `T` che estende la superclasse con un iteratore in grado di discriminare gli elementi secondo un predicato e di processarli tramite due funzioni distinte a seconda dell'esito dell'applicazione del predicato all'elemento.

   (a) [2 punti] Si definisca una *interfaccia funzionale* di nome `Predicate` specializzando l'interfaccia generica `java.util.Function` del JDK in modo che il dominio sia un generic `T` ed il codominio sia `Boolean`.

   (b) [2 punti] Si definisca una interfaccia `Either` parametrica su un tipo generico `T` e che definisce due metodi. Il primo metodo, di nome `onSuccess`, prende un `T` e ritorna un `T` e viene chiamato dall'iteratore quando il predicato ha successo. Il secondo metodo, di nome `onFailure`, viene invocato invece quando il predicato fallisce, prende un argomento di tipo `T` e non produce alcun risultato, tuttavia può lanciare una eccezione di tipo `Exception`.

   (c) [6 punti] Si definisca la sottoclasse `SkippableArrayList` parametrica su un tipo `E` e si implementi un metodo pubblico avente firma `Iterator<E> iterator(Predicate<E> p, Either<E> f)` che crea un iteratore con le caratteristiche accennate sopra. Più precisamente:

   - l'iteratore parte sempre dall'inizio della collezione ed arriva alla fine, andando avanti di un elemento alla volta normalmente;
   - ad ogni passo l'iteratore applica il predicato `p` all'elemento di tipo `T` corrente, che chiameremo `x`: se `p(x)` computa `true` allora viene invocato il metodo `onSuccess` di `f` e passato l'elemento `x` come argomento; altrimenti viene invocato il metodo `onFailure` e passato `x` come argomento a quest'ultimo;
   - l'invocazione di `onFailure` deve essere racchiusa dentro un blocco che assicura il *trapping* delle eccezioni: in altre parole, una eccezione proveniente dall'invocazione di `onFailure` non deve interrompere l'iteratore;
   - quando viene invocato `onSuccess`, il suo risultato viene restituito come elemento corrente dall'iteratore;
   - quando viene invocato `onFailure`, l'iteratore ritorna l'elemento originale che ha fatto fallire il predicato.

   (d) [4 punti] Si scriva un esempio di codice `main` che:

   - costruisce una `ArrayList` di interi vuota di nome `dst`;
   - costruisce una `SkippableArrayList` di interi di nome `src` e la popola con numeri casuali compresi tra 0 e 10, inclusi gli estremi[1];
   - invocando **solamente una volta** il metodo `iterator(Predicate<E>, Either<E>)` di `src` con gli argomenti opportuni, somma 1 a tutti gli elementi di `src` maggiori di 5 e appende in coda a `dst` quelli minori o uguali a 5.

   (e) [1 punti (bonus)] Il metodo `iterator()` con due parametri richiesto dal punto (c) è un override o un overload?

Total for Question 1: 14

2. (a) [6 punti] Si implementi in Java 8+ una classe `FiboSequence` le cui istanze rappresentano sequenze contigue di numeri di Fibonacci di lunghezza data in costruzione. Tali istanze devono essere *iterabili* tramite il costrutto *for-each* di Java, devono pertanto implementare l'interfaccia parametrica del JDK `java.util.Iterable<T>`. Ad esempio, il seguente codice deve compilare e stampare i primi 100 numeri di Fibonacci:

[1] Si utilizzi la classe `Random` del JDK per generare numeri casuali.

```
for (int n : new FiboSequence(100)) {
    System.out.println(n);
}
```

L'implementazione richiesta deve utilizzare una funzione ricorsiva che calcola l'n-esimo numero di Fibonacci.

(b) [4 punti] Si modifichi la classe `FiboSequence` in modo che i numeri di Fibonacci generati sottostiano ad un meccanismo di *caching* che ne allevia il costo computazionale memorizzando il risultato di ogni passo di ricorsione, in modo che ogni computazione successiva con il medesimo input costi solamente un accesso in lettura alla cache. Ogni istanza della classe di `FiboSequence` deve possedere la propria cache. Si utilizzino liberamente le mappe del JDK.

(c) [2 punti] Si modifichi la classe `FiboSequence` in modo che la cache sia condivisa tra molteplici istanze.

Total for Question 2: 12

3. [7 punti] Si definisca in linguaggio C++ una classe `smart_ptr` templatizzata su un tipo `T` che implementi la logica di uno *smart pointer*. Si implementino tutti i costruttori, i distruttori e gli operatori che si ritengono necessari ed utili affinché uno smart pointer si comporti in maniera compatibile con un pointer C. In altre parole, uno smart pointer deve implementare non solo il *reference counting* ma deve anche comportarsi come un puntatore classico, inclusi gli operatori di incremento/decremento, l'aritmetica dei puntatori ed ovviamente il de-reference.

Total for Question 3: 7

Question:	1	2	3	Total
Points:	14	12	7	33
Bonus Points:	1	0	0	1
Score:				