Programmazione ad Oggetti mod. 2

20/6/2018

Studente _____ Matricola _____

1. Si implementi in Java una sottoclasse generica di `ArrayList` di nome `FancyArrayList` che estende le funzionalità della superclasse con un iteratore più versatile. Il nuovo iteratore deve essere in grado di andare sia avanti che indietro secondo un valore di incremento intero non nullo; e di processare gli elementi che incontra durante l'attraversamento applicando una funzione di trasformazione[1].
   (a) [1 punti] Si definisca una *interfaccia funzionale* di nome `Function` parametrica sia sul tipo del dominio che sul tipo del codominio, equivalente a quella definita dal JDK 8+ nel package `java.util.function`.
   (b) [4 punti] Si definisca la sottoclasse `FancyArrayList` parametrica su un tipo `E` e si implementi un metodo pubblico avente firma `Iterator<E> iterator(int step, Function<E, E> f)` che crea un iteratore con le caratteristiche accennate sopra tramite una *classe anonima*. Più precisamente:
      - quando il valore del parametro `step` è positivo, l'iteratore parte dall'inizio della collezione e va *avanti* incrementando il cursore di `step` posizioni ad ogni passo; quando invece `step` è negativo, l'iteratore parte dalla fine della collezione e va *indietro* decrementando il cursore;
      - ad ogni passo l'iteratore applica la funzione di trasformazione `f` all'elemento da restituire.
   (c) Si aggiungano a `FancyArrayList` i seguenti metodi pubblici, badando ad implementarli in funzione del metodo `iterator(int, Function<E, E>)` realizzato per l'esercizio precedente, *senza replicazioni* di codice. Per ciascuno si specifichi inoltre se è un override oppure no, utilizzando opportunamente l'annotazione `@Override`.
      i. [2 punti] Si implementi il metodo avente firma `Iterator<E> iterator()` che produce un iteratore convenzionale che procede in avanti di una posizione alla volta.
      ii. [2 punti] Si implementi il metodo avente firma `Iterator<E> backwardIterator()` che produce un iteratore rovescio che procede indietro di una posizione alla volta.
   (d) [4 punti] Si rifattorizzi il metodo `iterator(int, Function<E, E>)` realizzato per il punto (b) in modo che non usi una classe anonima, ma una nuova classe innestata *statica* e parametrica di nome `FancyIterator`. Si presti particolare attenzione all'uso dei generics ed al passaggio esplicito della *enclosing instance* al costruttore.

Total for Question 1: 13

2. Si realizzi in Java un algoritmo che trova il minimo ed il massimo in una lista generica e restituisca i risultati, rispettivamente, come primo e secondo elemento di una coppia omogenea.
   (a) [2 punti] Si definisca un tipo per la coppia eterogenea, ovvero una classe `Pair` parametrica su due tipi distinti `A` e `B` che rappresentano rispettivamente il tipo del primo e del secondo elemento della coppia.
   (b) [4 punti] Si implementi un metodo statico e generico avente la seguente firma[2]:
   `static <E> Pair<E, E> findMinAndMax(List<E> l, Comparator<E> c)`
   L'algoritmo di ricerca del minimo e del massimo deve eseguire *una sola traversata* della lista; si assuma che gli argomenti siano non-nulli e che la lista abbia sempre almeno un elemento.
   (c) [4 punti] Si definisca un metodo in overload con il precedente che non usi un `Comparator` ma aggiunga il constraint `Comparable` al generic `E`, secondo la seguente firma:
   `static <E extends Comparable<E>> Pair<E, E> findMinAndMax(List<E> l)`
   Si implementi questo metodo in funzione del precedente, *senza replicare* l'algoritmo di ricerca.

Total for Question 2: 10

[1] Una funzione di trasformazione è una funzione in cui dominio è uguale al codominio, per esempio $f : \tau \to \tau$ per un qualche insieme τ.
[2] L'interfaccia parametrica `Comparator<T>` include un metodo binario avente firma `int compare(T a, T b)`, il quale confronta i due argomenti e ritorna un intero secondo la semantica standard del confronto *a tre vie* in Java.