Aula 12 - Desafio (impossível)

Docupedia Export

Author:Silio Leonardo (CtP/ETS)
Date:29-Apr-2024 13:17

Table of Contents

Você já viu na disciplina que Listas Encadeadas/Ligadas tem um problema ao queremos ler seus dados visto que temos $O(n)$ operações até encontrar o elemento desejado. Nossa ideia é implementar um padrão de projeto útil para resolver este problema e estendê-lo para padronizar a forma de se acessar qualquer coleção. A nossa ideia é implementa uma biblioteca de coleções e processamento de listas. Siga as instruções de implementação e boa sorte:

- No package NomeDaSuaBiblioteca.collections adicione as estruturas de dados implementadas nessa disciplina:
  - List<T>, que é uma classe abstrata que tanto ArrayList e LinkedList herdam, a classe possui:
    - get(int) → T $O(1)/O(n)$
    - set(int, T) $O(1)/O(n)$
    - add(T) $O(1)$
  - ArrayList<T>
  - LinkedList<T>
  - Queue<T>
    - enqueue(T) $O(1)$
    - dequeue() → T $O(1)$
  - Stack<T>
    - push(T) $O(1)$
    - pop() → T $O(1)$
  - Hash<T>
    - add(int, T) $O(1)$
    - get(int) → T $O(1)$
- No package NomeDaSuaBiblioteca.collections você deve ter uma classe abstrata que serve de base para todas as coleções.
  - Ela deve possuir apenas uma função de size() que retorna o tamanho e uma variável para guardar esse tamanho.
  - Também é interessante uma função protected para o setSize(int).
- No package NomeDaSuaBilbioteca.collections adicione a interface genérica Iterator<E>:
  - Ela representa um interador sobre uma coleção com dados de um tipo E, ela é a forma de padronizar como acessaremos coleções.
  - O iterador começa sempre na posição '-1' (antes do inicio). Ao chamar a função next() ela avança para próxima posição e retorna o objeto naquela posição.
  - Quando ela for implementada tem-de se considerar a possibilidade de ser chamado next() sem houver elementos a se ler, assim deve existir um exceção Checked em NomeDaSuaBilbioteca.Exceptions que será lançada nessa função. Claramente, ela precisa ser definida como 'public E next() throws MinhaExceção;',
  - Ela possui o método hasNext() que retorna verdadeiro se a coleção tem mais elementos a serem lidos.
- No package NomeDaSuaBiblioteca.collections adicione a interface genérica Iterable<E>:
  - Ela possui um método chamado iterator() que retorna um Iterator<E> que possibilita que a coleção seja lida.
  - Ela possui um método chamado stream() que retorna um objeto de Stream<E>que falaremos no futuro.
  - Todas as suas coleções devem implementar Iterable<E>.

    - Sendo assim você precisa implementar um ArrayListIterator<E>, um LinkedListIterator<E> e assim por diante. Esses objetos serão retornados no iterator() de cada um.
    - Para implementar um iterator implemente a interface Iterator<E>, coloque essas classes em NomeDaSuaBilbioteca.collections.iterators.
- No package NomeDaSuaBiblioteca defina a classe Stream<E> que recebe um Iterable<E> no construtor.
    - Stream<E> é uma classe para processamento de dados que possui métodos que vão utilizar seu Iterable<E> interno para buscar dados.
    - As funções receberão objetos de intefaces que funcionam como funções. Por exemplo, se eu quero fazer uma função que recebe um inteiro e retorna uma string posso fazer da seguinte forma:
        - Function<int, String> f = num -> num.toString();
        - Isso seria a mesma coisa que criar uma classe que implementa Function<int, String> e tem essa implementação.
        - Function, como você pode ver, tem dois parâmetros genéricos, isso é perfeitamente válido.
    - Stream<E> possui a função map(Function<E, R>) que retorna uma Stream<R>. Ou seja, a ideia é mapear os objetos.
        - Como o parâmetro R não está declarado, declare colocando <R> antes do retorno da função. Provavelmente você terá: 'public <R> Stream<R> map(Function<E, R> func)' na declaração.
    - Stream<E> possui a função filter(Function<E, boolean>) que retorna uma Stream<E>. Ou seja, a ideia é filtras os obejtos.
    - Stream<E> possui uma função chamada collect() que retorna um List<E> (podendo ser um ArrayList ou LinkedList).
    - Ao usar map ou filter salve os objetos e retorne uma nova Stream que tem todas as informações das operações que foram feitas, apenas execute as operações de fato no momento do collect.
- Se seu código estiver bem implementado o código abaixo torna-se possível:

```java
import NomeDaSuaBiblioteca;
import NomeDaSuaBiblioteca.collections;

public class Main
{
    public static void main(String[] args)
    {
        Hash<int, Pessoa> hash = new Hash<>();
        hash.add(1, new Pessoa("Don", 14, true));
        hash.add(2, new Pessoa("Trevis", 25, false));
        hash.add(3, new Pessoa("Queila", 23, true));
        hash.add(4, new Pessoa("Alisson", 26, false));
        hash.add(5, new Pessoa("Raissa", 21, true));

        List<String> dados = hash.stream()
            .filter(p -> p.getIdade() > 17)
            .filter(p -> p.isExApprentice())
            .map(p -> p.getNome())
```

```
19              .collect();
20
21          // Ao invés de
22          // List<String> dados = new ArrayList<String>();
23          // for (int i = 1; i < 6; i++)
24          // {
25          //      Pessoa p = hash.get(i)
26          //      if (p.getIdade() > 17)
27          //      {
28          //          if (p.isExApprentice())
29          //          {
30          //              dados.add(p.getName());
31          //          }
32          //      }
33          // }
34      }
35  }
```