Proiect – Partea a 2-a

Pentru partea a 2-a a proiectului, va trebui să implementați în Java rezolvarea problemei primite, folosind o implementare existentă pentru container (containerul pe care va trebui să-l folosiți, o să-l primiți după ce îmi trimiteți prima parte a proiectului). Puteți folosi containerul de mai multe ori/în mai multe locuri, dar încercați să nu folosiți alt container.

Va trebui să faceți un meniu mic în consolă și să aveți câte o opțiune pentru fiecare cerință a proiectului (cerințele sunt enumerate la partea cu funcționalități din descrierea problemei).

În tabelul de mai jos aveți clasele exacte din Java care trebuie folosite pentru rezolvare:

Containerul folosit	**Interfața**	**Clasa Java care poate fi folosită**
TAD Colecție*	Colectie	ColectieImpl
TAD Colecție Ordonată*	ColectieOrdonata	ColectieOrdonataImpl
TAD Mulțime	Set	HashSet
TAD Mulțime Ordonată	SortedSet	TreeSet
TAD Listă	List	ArrayList sau LinkedList
TAD Listă Ordonată*	ListaOrdonata	ListaOrdonataImpl
TAD Stiva	Stack	Stack
TAD Coadă	Queue	LinkedList
TAD Coadă cu Priorități	PriorityQueue	PriorityQueue
TAD Dicționar	Map	HashMap
TAD MultiDicționar*	MultiDictionar	MultiDictionarImpl

Vă rog să creați containerul folosit în implementarea proiectului sub forma:

- Interfata<TIP> numevariabila = new ClasaJava<TIP>();
- Set<String> s = new HashSet<String>();
- List<Integer> ll = new ArrayList<Integer>();
- Queue<String> coada = new LinkedList<String>();
- Map<String, Integer> dictionar = new HashMap<String, Integer>();

Containerele marcate cu * în tabelul de mai sus nu există în Java. Pentru aceste containere v-am făcut eu o implementare foarte simplă, fișierele sunt în arhiva atașată. Pentru fiecare container există o interfață (clasă în care sunt enumerate doar operațiile, fără implementare și fără atribute) și un fișier cu implementare. Ambele fișiere trebuie adăugate în proiectul vostru, și este suficient să vă uitați în interfață să vedeți ce operații există, nu trebuie să înțelegeți implementarea. Doar să folosiți operațiile.

Containerele ordonate implementate de mine necesită transmiterea unui comparator la constructor Acest comparator va fi folosit pentru a ordona elementele.

Atât containerele din Java, cât și implementările mele, pot primi ca parametru tipul elementelor care vor fi stocate în container (de exemplu, dacă vreți o Colecție de Stringuri, puteți scrie Colecție<String>).

Toate containerele (mai puțin Coada, Stiva și Coada cu Priorități) au iterator. Iteratorul în Java are doar 2 operații (nu trei, așa cum discutăm noi la curs): hasNext (returnează true dacă mai sunt elemente de iterat) și next (returnează elementul curent și trece la elementul următor). Implementările cu * tot aceste 2 operații le au. Pentru a crea un iterator pentru un container (exceptând MultiDicționarul și Dicționarul) putem scrie:

Iterator<String> it = s.iterator()

Evident, în loc de String puteți folosi tipul de date pe care îl aveți voi în container.

Pentru un MultiDicționar, puteți crea un iterator cu:

Iterator<Map.Entry<String, Integer>> it = s.iterator()

Map.Entry este o clasă din Java, care reprezintă o pereche cheie, valoare. Iteratorul merge peste perechi. Elementul returnat de funcția next() este o pereche (de tip Map.Entry), care are operația getKey() și getValue() pentru a returna cheia și valoarea din pereche.

Dicționarul în Java nu are iterator. Dar are operații care returnează o mulțime de chei sau o mulțime de perechi (o pereche este de tip Map.Entry) și puteți defini un iterator pe această mulțime.