Tema laborator 6
Colectii. Interfete II.

Acest laborator va invata bazele colectiilor. Trebuie sa va aduceti aminte de notiunea de complexitate a algoritmului (O(n)). Veti scrie cod ce testeaza timpii de acces la principalele colectii din Java.

Veti implementa o interfata standard din Java ce va va ajuta la sortari.

Este mai putin de codat si mai mult de citit.

1 Materiale aditionale ce trebuie studiate

Cititi si intelegeti cursul nr 6, partea de colectii si generice

Pentru mai multe detalii http://docs.oracle.com/javase/tutorial/collections/index.html

Cu cat cititi de acolo mai mult, cu atat mai bine. E multa informatie, nu intrati in foarte multe detalii DAR incercati sa acoperiti subiectele principale (ex. cititi despre fiecare tip de colectie ce apare in aceasta tema)

2 Teorie

Un tip generic este o clasa pentru care tipul variabilelor membru nu este cunoscut decat in momentul rularii/instantierii.

De ex clasa declarata in Java ArrayList<T> este un generic. Tipul T va fi inlocuit la instantiere. Exemplu:

```
ArrayList<Integer> listOfIntegers;
ArrayList<Funct> listOfPolynomials; //vezi laborator 3
```

Ganditi-va ca litera T din declaratia clasei este inlocuit de compilator de tipul specificat de dvs in momentul utilizarii genericului.

Colectiile sunt de mai multe tipuri „matematice”

1. Lists: lista obiecte care pastreaza ordinea de inserare, obiectele pot fi duplicate
2. Sets: multimi matematice (obiecte unice, ordinea de inserare nu e relevanta)
3. Maps: seturi asociative, unde pastrez perechi (cheie:valoare), setul de chei fiind unic

Exista mai multe tipuri de implementari pentru fiecare. Exemplu, LinkedList<> pentru o lista in care insertiile sunt O(1) dar cautarile aleatorii O(n), TreeSet unde se pastreaza o colectie ordonata, cu timp de cautare O(log n), etc. Detalii in Java Tutorials

3 Practica

Descarcati si importati proiectul. Vezi laborator 2 pentru detalii.

3.1 Cronometrarea timpilor de acces la colectii

Vom monitoriza timpii diverselor operatii asupra colectiilor. Primul pas e crearea de metode care realizeaza accesul.

Adaugati urmatoarele operatii pe colectii in clasa Main06:

1. public void populeaza(Collection<Integer> collection, int count) – metoda ce insera count elemente aleatoare de tip Integer intr-o colectie. Folositi metoda add(E element) al interfetei Collection.
2. public void adaugaInFata(List<Integer> list, int count) – metoda ce insera count elemente IN FATA tuturor elementelor din lista. Folositi metoda `add(int index, E element)` al interfetei List. Index sa fie setat pe 0.
3. public void cauta(Collection<Integer> collection, int count) - apeleaza metoda boolean contains(Object o) a interfetei Collection de count ori, folosind numere aleatoare
4. public void acceseazaAleatoriu(List<Integer> list, int count) – acceseaza al k-lea element, de count ori. Se foloseste metoda get(int i). k e un numar aleator intre 0 si dimensiunea listei.

Pentru a genera un numar aleator folositi obiecte din clasa Random:

```
Random rndGen = new Random();
k = rndGen.nextInt(100); //Genereaza un nr aleator intre 0 si 100
```

In principiu fiecare metoda trebuie sa urmeze sablonul:

1. Repeta de count ori:
    a. Genereaza un numar aleator folosind metoda nextInt()
    b. Executa operatia pe colectia de intrare.

Adaugati clasa de test Main06Test. Timpii de executie sunt monitorizati automat de metodele de test. Adaugati metode de test dupa modelul:

1. Creaza colectia (ex: ArrayList<Integer> lista = new ArrayList<>)
2. Apeleaza metoda din Main06 cu o valoare mare pentru count (ex 10000)

Pentru operatia de cautare, apelati mai intai populeaza() cu valoarea data, pentru ca metoda cauta sa nu mearga in gol.

Urmatorul tabel va spune ce tipuri de colectii sa testati: (Acolo unde e scris DA, inseamna ca trebuie sa scriete un test ce creaza colectia din capul randului si apeleaza operatia din capul coloanei)

	adaugaInFata	**Cauta**	**acceseazaAleatoriu**
ArrayList<>	DA	DA	DA
LinkedList<>	DA	DA	DA
HashSet<>		DA	
TreeSet<>		DA	

Atentie! Pentru ca timpul raporat sa fie corect, fiecare metoda de test testeaza un singur lucru!

3.2 Folosirea interfetelor existente din Java

In proiect aveti clasa Person si metoda generatePersonList() ce returneaza un ArrayList de persoane.

Veti scrie cateva clase ce implementeaza interfata Interface Comparator<T>. Aceasta interfata are 2 metode, compare si equals. Citit documentatia pentru detalii despre ce rol are fiecare.

Vom folosi comparatorii creati pentru a sorta liste de persoane. Metoda standard din java este public static <T> void sort(List<T> list, Comparator<? super T> c).

Aceasta metoda accepta o lista si un obiect Comparator care stie cum sa compare obiectele.

Vom scrie comparatori ce compara 2 persoane dupa:

1. Nume
2. Varsta
3. Numele strazii

Sortati lista si afisati primele 10-20 persoane, in fiecare caz de sortare.

Hints:

Folositi-va de metoda compareToIgnoreCase a clasei String

Metoda equals implementati-o folosind metoda compare. (returneaza true daca compare returneaza 0)

Clasa Person are deja implementata metoda toString()

Scrieti metode pentru a va scuti de operatiile repetitive (ex: fiecare lista trebuie sortata si afisata. Singurul lucru care difera, e comparatorul)

4 Evaluare

Executati toti pasii din laborator.

5 Referinte:

http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html
http://docs.oracle.com/javase/8/docs/api/java/util/Collection.html
http://docs.oracle.com/javase/8/docs/api/java/util/List.html
http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
http://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html
http://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html
http://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html

http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#sort(java.util.List,%20java.util.Comparator)