Zadanie 2

cz.1

Kolekcje i odwzorowania

1. Zdefiniuj typ wyliczeniowy EmployeeCondition z polami: obecny, delegacja, chory, nieobecny…

2. Zaimplementuj:

a) klasę Employee z polami: imię (String), nazwisko (String), stan pracownika (EmployeeCondition), rok urodzenia (integer), wynagrodzenie (double). Można wprowadzić też inne pola.

b) konstruktor pozwalający na łatwą inicjalizację obiektu (uwzględniający zdefiniowane wcześniej pola)

c) metodę printing wypisującą na standardowe wyjście pełne informacje o pracowniku.

3. Klasa Employee powinna implementować interfejs Comparable< Employee > pozwalający na porównanie pracowników ze względu na nazwisko.

4. Utwórz klasę ClassEmployee, która będzie zawierać takie informacje jak: nazwa grupy pracowniczej, lista pracowników, maksymalna ilość pracowników. Klasa ta powinna uwzględniać też następujące metody:

a) addEmployee(Employee) – metoda dodaje pracownika do grupy. Jeśli dany pracownik będzie już obecny w grupie (pracownik o tym imieniu i nazwisku już istnieje) to należy wyświetlić komunikat. Pracownik może zostać dodany, tylko jeśli nie zostanie przekroczona pojemność grupy. Jeśli pojemność zostanie przekroczona należy wypisać odpowiedni komunikat,

b) addSalary(Employee, double) – dodająca danemu pracownikowi pewną kwotę do wynagrodzenia,

c) removeEmployee(Employee) – metoda usuwająca całkowicie pracownika,

d) changeCondition(Employee, EmployeeCondition) – metoda zmieniająca stan pracownika,

e) search(String) – metoda powinna sprawdzić istnienie nazwiska pracownika i zwrócić jego dane. Proszę zastosować Comparator,

f) searchPartial(String) – metoda przyjmująca fragment nazwiska/imienia pracownika i zwracająca wszystkie osoby, które pasują do tego kryterium,

g) countByCondition (EmployeeCondition) – zwraca ilość pracowników o danym stanie,

h) summary() – wypisuje na standardowe wyjście informację o wszystkich pracownikach,

i) sortByName() – zwraca posortowaną listę pracowników – po nazwie alfabetycznie,

j) sortBySalary() – zwraca posortowaną listę pracowników po wynagrodzeniu – malejąco – zastosuj własny Comparator,

k) max() – zastosuj metodę Collections.max.

5. Zaimplementuj klasę ClassContainer przechowującą w Map<String, ClassEmployee> grupy pracownicze (kluczem jest nazwa grupy) i następujące metody:

a) addClass(String, double) – metoda dodaje nową grupę pracowniczą o podanej nazwie i zadanej pojemności do spisu grup,

b) removeClass(String) – metoda usuwa grupę o podanej nazwie,

c) findEmpty() – metoda zwraca listę pustych grup,

d) summary() – metoda wypisująca na standardowe wyjście informacje zawierające: nazwę grupy i jej procentowe zapełnienie.

Proszę przemyśleć również kwestię dodania innych przydatnych metod i zmiennych. Działanie poszczególnych metod w aplikacji w metodzie main będzie demonstrowane poprzez ich uruchomienie wedle potrzeb. Proszę nie tworzyć menu.

Wskazówki:

1. Typ wyliczeniowy z automatyczną konwersją na String

```
private enum Answer {

YES {

@Override public String toString() {

return "yes";

} },

NO,

MAYBE
```

}

2. Wykorzystanie Comparator w algorytmach:

```
List< Employee> employees = new ArrayList<>();

employees.add(new Employee ("Adam", 5));

employees.add(new Employee ("Grzegorz", 2));

// Implementacja inplace - klasa anonimowa

Employee s1 = Collections.max(employees, new Comparator< Employee >() {

@Override

public int compare(Employee w1, Employee w2) {

return Integer.compare(w1.salary, w2.salary);

}

});

// Implementacja przez wyrażenie Lambda

Employee s2 = Collections.max(employees, (w1, w2) -> {

return Integer.compare(w1.salary, w2.salary);

});
```

https://javastart.pl/static/algorytmy/sortowanie-kolekcji-interfejsy-comparator-i-comparable/

3. Metoda contains(String) klasy String zwraca true jeśli podany w argumencie napis zawiera się w obiekcie na rzecz którego została uruchomiona metoda.

4. Interfejsy Comparable oraz Comparator są częścią języka Java! Implementując metodę compareTo lub compare pamiętaj, że muszą one zwracać liczbę całkowitą. Jeśli obiekt ma być w pewnej hierarchii przed innym to zwracamy wartość mniejszą od 0, jeśli za innym to większą od 0, natomiast jeśli są równe to zwracane jest 0. Metodę compareTo możesz jawnie uruchomić np. na obiekcie typu String w celu jego porównania.