Zadanie 2

Kolekcje i odwzorowania

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

2. Zaimplementuj:

a) klasę Teacher z polami: imię (String), nazwisko (String), stan nauczyciela (TeacherCondition), 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 nauczycielu,

3. Klasa Teacher powinna implementować interfejs Comparable< Teacher > pozwalający na porównanie nauczycieli ze względu na nazwisko.

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

a) addTeacher(Teacher) – metoda dodaje nauczyciela do grupy. Jeśli dany nauczyciel będzie już obecny w grupie (nauczyciel o tym imieniu i nazwisku już istnieje) to należy wyświetlić komunikat. Nauczyciel może zostać dodany, tylko jeśli nie zostanie przekroczona pojemność grupy. Jeśli pojemność zostanie przekroczona należy wypisać odpowiedni komunikat,

b) addSalary(Teacher, double) – dodająca danemu nauczycielowi pewną kwotę do wynagrodzenia,

c) removeTeacher (Teacher) – metoda usuwająca całkowicie nauczyciela,

d) changeCondition(Teacher, TeacherCondition) – metoda zmieniająca stan nauczyciela,

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

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

g) countByCondition (TeacherCondition) – zwraca ilość nauczycieli o danym stanie,

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

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

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

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

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

a) addClass(String, double) – metoda dodaje nową grupę nauczycielską 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< Teacher > teachers = new ArrayList<>();
teachers.add(new Teacher ("Adam", 5));
teachers.add(new Teacher ("Grzegorz", 2));
// Implementacja inplace - klasa anonimowa
Teacher o1 = Collections.max(teachers, new Comparator< Teacher >() {
```

```
@Override
public int compare(Teacher t1, Teacher t2) {
return Integer.compare(t1.salary, t2.salary);
}
});
// Implementacja przez wyrażenie Lambda
Teacher o2 = Collections.max(teachers, (t1, t2) -> {
return Integer.compare(t1.salary, t2.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.

https://www.tutorialspoint.com/java/lang/string_contains.htm

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.