PAU IO
Lab. 2 – Kolekcje

Cel zadania: Stworzyć prosty symulator dziennika ocen z przedmiotu.

Zaimplementuj:

1. Typ wyliczeniowy StudentCondition z polami: odrabiający, chory, nieobecny…
2. Klasę Student z polami: imię (String), nazwisko (String), stan studenta (StudentCondition), rok urodzenia (integer), ilość punktów (double). Zaproponuj inne pola.
    a. Konstruktor pozwalający na łatwą inicjalizację obiektu (powyższe pola)
    b. Metodę print wypisujący na standardowe wyjście pełne informacje o studencie
    c. Niech klasa Student implementuje interfejs Comparable< Student > pozwalający na porównanie studentów ze względu na nazwisko.
3. Klasę Class, która zawiera takie informacje jak: nazwa grupy, lista studentów, maksymalna ilość studentów. Oraz następujące metody:
    a. addStudent(Student) – Dodająca studenta do grupy. Jeśli dany student będzie już obecny w grupie (student o tym imieniu istnieje) to należy wyświetlić komunikat. Student może zostać dodany, tylko jeśli niezostanie przekroczona pojemność grupy. Jeśli pojemność zostanie przekroczona wypisz komunikat na standardowe wyjście błędów (System.err)
    b. addPoints(Student, double) – dodająca danemu studentowi punkty
    c. getStudent(Student) – Zmniejszający ilość studentów o jeden (usuwający go całkowicie, jeśli ilość jego punktów będzie równa 0.)
    d. changeCondition(Student, StudentCondition) – zmieniający stan studenta
    e. removePoints(Student, double) – usuwający daną ilość punktów studentowi.
    f. search(String) - Przyjmującej nazwisko studenta i zwracający go. Zastosuj Comparator
    g. searchPartial(String) – Przyjmujący fragment nazwiska/imienia studenta i zwracający wszystkie osoby, które pasują.
    h. countByCondition(StudentCondition) – zwracający ilość osób o danym stanie
    i. summary() – wypisującą na standardowe wyjście informację o wszystkich osobach
    j. sortByName() – zwracającą posortowana listę studentów – po nazwie alfabetycznie
    k. sortByPoints() – zwracającą posortowaną listę studentów po ilości punktów – malejąco – zastosuj własny Comparator
    l. max() – zastosuj metodę `Collections.max`
4. Klasę ClassContainer przechowującą w Map<String, Class > grupy. (Kluczem jest nazwa grupy), zaimplementuj metody:
    a. addClass(String, double) – dodającą nową grupę o podanej nazwie i zadanej pojemności do spisu grup
    b. removeClass(String) – usuwający grupę o podanej nazwie
    c. findEmpty() – zwracający listę pustych grup
    d. summary() – wypisujący na standardowe wyjście informacje zawierające: nazwę grupy i procentowe zapełnienie

Dodać inne przydatne metody i zmienne.
Pokazać działanie wszystkich metod w aplikacji w metodzie main poprzez uruchomienie każdej metody wedle potrzeb. **NIE twórz menu – pokaż przykładowe wywołania w metodzie main.**

5. Teoria:
   a) Co zyskujemy pisząc
      `List<?> myList = new ArrayList<?>();`
      zamiast
      `ArrayList<?> myList = new ArrayList<?>();`
   b) ArrayList vs LinkedList – kiedy używać jakich list?
      https://javastart.pl/static/klasy/interfejs-list/
   c) HashMap vs TreeMap vs LinkedHashMap – kiedy używać jakich map
      https://javastart.pl/static/klasy/interfejs-map/
   d) List vs Map vs Set – w jakich przypadkach użyć którą kolekcję?
   e) Interfejs Comparable – jak go używać? jakie problemy rozwiązuje?
   f) Użyteczne metody algorytmiczne z klasy Collections (sort, max)
   g) Różnica między metodą equals a operatorem == (na przykładzie obiektu String)
   h) Po co używamy adnotacji `@Override`
      https://stackoverflow.com/questions/94361/when-do-you-use-javas-override-annotation-and-why
   i) Klasa wewnętrzna i anonimowa klasa wewnętrzna (anonymous inner class). Gdzie i po co je wykorzystujemy (odpowiedzieć na przykładach).
   j) Czym są wyrażenia lambda, jak się je konstruuje, gdzie mogą być przydatne
      https://www.geeksforgeeks.org/lambda-expressions-java-8/
      https://www.geeksforgeeks.org/java-lambda-expression-with-collections/
      https://softwareengineering.stackexchange.com/questions/195081/is-a-lambda-expression-something-more-than-an-anonymous-inner-class-with-a-singl

6. Wskazówki:

   1. Typ wyliczeniowy z automatyczną konwersją na String

```
private enum Answer {
  YES {
    @Override public String toString() {
      return "yes";
    }
  },

  NO,
  MAYBE
}
```

   2. Jak wykorzystać Comparator w algorytmach:

```
List<Student> students = new ArrayList<>();
students.add(new Student("Adam", 5));
students.add(new Student("Grzgorz", 2));

// Implementacja inplace - klasa anonimowa
Student s1 = Collections.max(students, new Comparator<Student>() {
    @Override
    public int compare(Student o1, Student o2) {
        return Integer.compare(o1.score, o2.score);
    }
});
```

```
// Implementacja przez wyrażenie Lambda
Student s2 = Collections.max(students, (o1, o2) -> {
    return Integer.compare(o1.score, o2.score);
});
```

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

Po uzyskaniu zaliczenia na zajęciach, prześlij źródła w archiwum **zgodnie z konwencją nazewniczą** (patrz prezentacja)