Programowanie w JAVA
Lab. 2 – Kolekcje

1. **Cel zadania:** Stworzyć prosty symulator usługi magazynowej
2. **Zaimplementuj:**
    a. Typ wyliczeniowy ItemCondition z polami: NEW, USED, REFURBISHED
    b. Klasę Item z polami: nazwa (String), stan (ItemCondition), masa (double), ilość (integer)
        i. Konstruktor pozwalający na łatwą inicjalizację obiektu (nazwa, masa, stan, ilość)
        ii. Metodę print wypisujący na standardowe wyjście pełne informacje o towarze
        iii. Niech klasa Item implementuje interfejs Comparable<Item> pozwalający na porównanie obiektów ze względu na nazwę.
    c. Klasę FulfillmentCenter, która zawiera takie informacje jak: nazwa magazynu, lista produktów, maksymalna pojemność magazynu (maksymalna masa wszystkich produktów). Oraz następujące metody:
        i. addProduct(Item) – Dodająca produkt. Jeśli dany produkt będzie już obecny w magazynie (produkt o tej samej nazwie istnieje) to należy zsumować ich ilość. Produkt może zostać dodany, tylko jeśli niezostanie przekroczona pojemność magazynu. Jeśli pojemność zostanie przekroczona wypisz komunikat na standardowe wyjście błędów (System.err)
        ii. getProduct(Item) – Zmniejszający ilość danego produktu o jeden lub usuwający go całkowicie, jeśli po zmianie wartość będzie równa 0.
        iii. removeProduct(Item) – usuwający dany produkt całkowicie z magazynu.
        iv. search(String) - Przyjmującej nazwę produktu i zwracający go. Zastosuj Comparator
        v. searchPartial(String) – Przyjmujący fragment nazwy produktu i zwracający wszystkie produkty, które pasują.
        vi. countByCondition(ItemCondition) – zwracający ilość produktów o danym stanie
        vii. summary() – wypisującą na standardowe wyjście informację o wszystkich produktach
        viii. sortByName() – zwracającą posortowana listę produktów – po nazwie alfabetycznie
        ix. sortByAmount() – zwracającą posortowaną listę produktów po ilości – malejąco – zastosuj własny Comparator
        x. max() – zwracającą produkt którego jest najwięcej - zastosuj metodę `Collections.max`
    d. Klasę FulfillmentCenterContainer przechowującą w Map<String, FulfillmentCenter> magazyny. (Kluczem jest nazwa magazynu), zaimplementuj metody:
        i. addCenter(String, double) – dodającą nowy magazyn o podanej nazwie i zadanej pojemności do spisu magazynów
        ii. removeCenter(String) – usuwający magazyn o podanej nazwie
        iii. findEmpty() – zwracający listę pustych magazynów
        iv. summary() – wypisujący na standardowe wyjście informacje zawierające: nazwę magazynu i procentowe zapełnienie
3. Pokazać działanie wszystkich metod w aplikacji w metodzie main poprzez uruchomienie każdej metody wedle potrzeb. **NIE twórz menu.**
4. 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) Interfejs Comparator – jak go używać? jakie problemy rozwiązuje?
g) Użyteczne metody algorytmiczne z klasy Collections (sort, max)
h) Różnica między metodą equals a operatorem == (na przykładzie obiektu String)

5. 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, prześlij źródła w archiwum **zgodnie z konwencją nazewniczą** do chmury na adres:
https://cloud.kisim.eu.org/index.php/s/Nfbpg3HGZfdPPeK