Programowanie w JAVA
Lab. 2 – Kolekcje

Cel zadania: Stworzyć prosty symulator salonu samochodowego

Zaimplementuj:

1. Typ wyliczeniowy ItemCondition z polami: NEW, USED, DAMAGED
2. Klasę Vehicle z polami: marka (String), model (String), stan (ItemCondition), cena (double), rok produkcji (integer), przebieg (double), pojemność silnika (double)
    a. Konstruktor pozwalający na łatwą inicjalizację obiektu (marka, model, stan, cena, rok, przebieg, silnik)
    b. Metodę print wypisujący na standardowe wyjście pełne informacje o towarze
    c. Niech klasa Vehicle implementuje interfejs Comparable< Vehicle > pozwalający na porównanie obiektów ze względu na nazwę.
3. Klasę CarShowroom, która zawiera takie informacje jak: nazwa salonu, lista samochodów, maksymalna pojemność salonu (maksymalna ilość wszystkich pojazdów). Oraz następujące metody:
    a. addProduct(Vehicle) – 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)
    b. getProduct(Vehicle) – Zmniejszający ilość danego produktu o jeden lub usuwający go całkowicie, jeśli po zmianie wartość będzie równa 0.
    c. removeProduct(Vehicle) – usuwający dany produkt całkowicie z magazynu.
    d. search(String) - Przyjmującej nazwę produktu i zwracający go. Zastosuj Comparator
    e. searchPartial(String) – Przyjmujący fragment nazwy produktu i zwracający wszystkie produkty, które pasują.
    f. countByCondition(ItemCondition) – zwracający ilość produktów o danym stanie
    g. summary() – wypisującą na standardowe wyjście informację o wszystkich produktach
    h. sortByName() – zwracającą posortowana listę produktów – po nazwie alfabetycznie
    i. sortByAmount() – zwracającą posortowaną listę produktów po ilości – malejąco – zastosuj własny Comparator
    j. max() – zwracającą produkt którego jest najwięcej - zastosuj metodę `Collections.max`
4. Klasę CarShowroomContainer przechowującą w Map<String, CarShowroom > salony. (Kluczem jest nazwa salonu), zaimplementuj metody:
    a. addCenter(String, double) – dodającą nowy salon o podanej nazwie i zadanej pojemności do spisu salonów
    b. removeCenter(String) – usuwający salon o podanej nazwie
    c. findEmpty() – zwracający listę pustych magazynów
    d. summary() – wypisujący na standardowe wyjście informacje zawierające: nazwę salonu 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) 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)
   i) Po co używamy adnotacji `@Override`
      https://stackoverflow.com/questions/94361/when-do-you-use-javas-override-annotation-and-why
   j) Klasa wewnętrzna i anonimowa klasa wewnętrzna (anonymous inner class). Gdzie i po co je wykorzystujemy (odpowiedzieć na przykładach).
   k) 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) do chmury na adres:

https://cloud.kisim.eu.org/s/zaJ2LgHJ7c7KRYZ.