PAOiM z 24
Lab. 2 – Kolekcje

Cel zadania: Stworzyć prosty symulator do zarządzania schroniskami/sklepami zoologicznymi.

Zaimplementuj:

1. Typ wyliczeniowy **AnimalCondition** z polami: Zdrowe, Chore, W trakcie adopcji, Kwarantanna
2. Klasę **Animal** z polami: name (String), species (String), condition (AnimalCondition), age (int), price (double). Zaproponuj inne pola.
    a. Konstruktor pozwalający na łatwą inicjalizację obiektu (powyższe pola)
    b. Metodę print() wypisujący na standardowe wyjście pełne informacje o zwierzęciu
    c. Klasa Animal powinna implementować interfejs Comparable<Animal>, który pozwala na porównanie zwierząt według ich imienia, gatunku lub wieku.
3. Klasę **AnimalShelter**, która zawiera takie informacje jak: shelterName (String), animalList (List<Animal>), maxCapacity (int) – maksymalna ilość zwierząt w schronisku. Oraz następujące metody:
    a. addAnimal(Animal animal) – dodająca zwierzę do schroniska. Jeśli dane zwierzę już istnieje (porównując według imienia, gatunku i wieku), wyświetl komunikat. Jeśli pojemność zostanie przekroczona, wypisz komunikat na standardowe wyjście błędów (System.err).
    b. removeAnimal(Animal animal) – usuwająca zwierzę ze schroniska.
    c. getAnimal(Student) – Zmniejszający ilość zwierząt o jeden (zmieniający stan na zaadoptowany i usuwający go z danego schroniska)
    d. changeCondition(Animal animal, AnimalCondition condition) – zmieniająca stan zwierzęcia.
    e. changeAge(Animal, int) – zmieniająca wiek zwierzęcia (można automatycznie lub ręcznie).
    f.
    g. countByCondition(AnimalCondition condition) – zwracająca ilość zwierząt w danym stanie.
    h. sortByName() – zwracająca posortowaną listę zwierząt – po imieniu alfabetycznie.
    i. sortByPrice() – zwracająca posortowaną listę zwierząt po cenie – rosnąco.
    j. search(String name) – przyjmująca imię zwierzęcia i zwracająca je. Zastosuj Comparator.
    k. searchPartial(String) – Przyjmujący fragment imienia/gatunku i zwracający wszystkie zwierzęta, które pasują.
    l. summary() – wypisująca na standardowe wyjście informację o wszystkich zwierzętach.
    m. max() – zastosuj metodę `Collections.max`
4. Klasę ShelterManager która będzie przechowywała w Map<String, AnimalShelter> schroniska. (Kluczem jest nazwa schroniska), zaimplementuj metody:
    a. addShelter(String name, int capacity) – dodająca nowe schronisko o podanej nazwie i zadanej pojemności do spisu schronisk.
    b. removeShelter(String name) – usuwająca schronisko o podanej nazwie.
    c. findEmpty() – zwracająca listę pustych schronisk.
    d. summary() – wypisująca na standardowe wyjście informacje zawierające: nazwę schroniska 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 musisz tworzyć 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)