PAOiM IT
Lab. 2 – Kolekcje

Cel zadania: Stworzyć prosty symulator do zarządzania stadniną koni.

Zaimplementuj:

1. Typ wyliczeniowy **HorseCondition** z polami reprezentującymi stan zdrowia: zdrowy, chory, trening, karencja, sprzedany
2. Klasę **Horse** z polami name (String), breed (String) – rasa konia, type (HorseType) – typ konia (zimnokrwisty lub gorącokrwisty), status (HorseStatus) – aktualny stan, age (int) – wiek, price (double) – wartość konia w złotówkach. 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 Horse powinna implementować interfejs Comparable, który pozwala na porównanie koni według ich imienia, rasy lub wieku.
3. Klasę **Stable**, która zawiera takie informacje jak: stableName (String), horseList (List<Horse>), maxCapacity (int) – maksymalna ilość koni w stadninie. Oraz następujące metody:
    a. addHorse(Horse horse) –Dodająca konia do stadniny. Jeśli dane zwierzę już istnieje (porównując według imienia, rasy i wieku), wyświetl komunikat. Jeśli pojemność zostanie przekroczona wypisz komunikat na standardowe wyjście błędów (System.err)
    b. removeHorse(Horse horse) – usuwająca zwierzę ze stadniny (np. jeśli waga < X lub koń został sprzedany).
    c. sickHorse(Horse horse) – Zmniejszający ilość zwierząt o jeden (zmieniający stan na chory)
    d. changeCondition(Horse horse, HorseCondition condition) – zmieniający stan konia
    e. change Weight(Horse horse, double kg) – zmieniający wagę konia.
    f. countByStatus(HorseStatus status) – zwraca liczbę koni w danym stanie.
    g. sortByName() – zwracająca posortowaną listę zwierząt – po imieniu alfabetycznie.
    h. sortByPrice() – zwracająca posortowaną listę zwierząt po cenie – rosnąco.
    i. search(String name) – przyjmująca imię konia i zwracająca je. Zastosuj Comparator.
    j. searchPartial(String) – Przyjmujący fragment imienia/rasy i zwracający wszystkie zwierzęta, które pasują.
    k. summary() – wypisująca na standardowe wyjście informację o wszystkich koniach.
    l. max() – zastosuj metodę Collections.max
4. Klasę **StableManager** przechowującą w Map<String, Stable > wszystkie stadniny. (Kluczem jest nazwa stadniny), zaimplementuj metody:
    a. addStable(String name, int capacity) – dodająca nową stadninę o podanej nazwie i zadanej pojemności do spisu stadnin.
    b. removeStable(String name) – usuwająca stadninę o podanej nazwie.
    c. findEmpty() – zwracająca listę pustych stadnin.
    d. summary() – wypisująca na standardowe wyjście informacje zawierające: nazwę stadniny 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)