Simple Tasks

На свій смак реалізувати **одне** з завдань:

Завдання 2.1

Крок 1. Реалізувати метод equals для класу Person у якого є декілька полів (прізвище, ім’я, вік).

Крок 2. Реалізуйте наступний сценарій:

a. Створіть екземпляр Person

b. Конвертуйте його в JSON*

c. Конвертуйте назад в об’єкт*

d. Перевірте equals-ом початковий і одержаний об'єкти

* Для серіалізації та десеріалізації в/з JSON можна використовувати бібліотеку gson (https://sites.google.com/site/gson/gson-user-guide).

Бажано!

Реалізуйте unit tests для методу equals за допомогою бібліотеки equals verifier

Завдання 2.2

Напишіть консольну програму, яка дозволяє створювати сутність "Запис в журналі куратора".

Для цього потрібно:

- організувати введення даних з командної строки і передати результат введення у відповідну сутність;
- перевіряти на правильність введення даних (зберігаючи правильно введені) і в разі повної коректності всіх даних – передати їх до відповідного класу в моделі; якщо дані не відповідають необхідному формату, то запропонувати повторне введення.
- відображати всі записи журналу.

Сутність "Запис в журналі куратора" описана наступним набором:

- прізвище студента;
- ім'я студента;
- дата народження студента;
- телефон студента;

- домашня адреса (вулиця, будинок, квартира).

Hard Task

Виконуються всі завдання + вона модифікується в 4-й л/р.

1) Написати інтерфейс MyList, який містить наступні методи

```
void add(Object e) – додає елемент в кінець списку
void add(int index, Object element) – додає елемент у вказане місце списку
void addAll(Object[] c) - додає масив елементів в кінець списку
void addAll(int index, Object[] c) - додає масив елементів у вказане місце
списку
Object get(int index) – повертає елемент за індексом
Object remove(int index) - видаляє елемент за індексом

void set(int index, Object element) – змінює значення елемента

int indexOf(Object o) - пошук індексу за значенням елемента

int size() - розмір списку
Object[] toArray() - перетворює список в масив об'єктів
```

На основі вказаного інтерфейсу реалізувати класи MyLinkedList та MyArrayList без використання аналогічних Java-класів.
MyArrayList має реалізовувати маркерний інтерфейс RandomAccess.

2) Реалізувати LinkedHashSet

3) Реалізувати кеш. Відмінність кешу від Map

While Caches and Maps share somewhat similar APIs, Caches are not Maps and Maps are not Caches. The following section outlines the main similarities and differences.

Like Map-based data-structures:
● Cache values are stored and accessed through an associated key.
● Each key may only be associated with a single value in a Cache.
● Great care must be exercised if mutable objects are used as keys. The behavior of a Cache is undefined if a key is mutated in a manner that affects equals comparisons when a key is used with a Cache.
● Caches depend on the concept of equality to determine when keys and values are the same. Consequently custom key and value classes should define a suitable implementation of the Object.equals method.
● Custom key classes should additionally provide a suitable implementation of the Object.hashCode method.
Although recommended, implementations are not required to call either the Object.hashCode or Object.equals methods defined by custom key classes.
Implementations are free to use optimizations whereby the invocation of these methods is avoided.

As this specification does not define the concept of object equivalence it should be noted applications that make use of custom key classes and rely on implementation specific optimizations to determine equivalence may not be portable.

Unlike Map-based data-structures:

● Cache keys and values must not be null.

Any attempt to use null for keys or values will result in a NullPointerException being thrown regardless of the use.

● Entries may expire.

The process of ensuring Entries are no longer available to an application because they are no longer considered valid is called "expiry"

● Entries may be evicted. Caches are typically not configured to store an entire data set. Instead they are often used to store a small, frequently used subset of the an entire dataset.

To ensure that the size of a Cache doesn’t consume resources without bounds, a Cache implementation may define a policy to constrain the amount of resources a Cache may use at runtime by removing certain entries when a resource limit is exceeded.

The process of removing entries from a Cache when it has exceeded a resource limit is called "eviction". When an Entry is removed from a Cache due to resource constraints, it is said to be "evicted".