Як ми знаємо, абстрактний тип даних (АТД) — це тип даних, який надає для роботи з елементами певного набору цього типу [функцій](https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), а також можливість створювати елементи цього типу за допомогою спеціальних функцій. Вся внутрішня структура такого типу захована від розробника [програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) — в цьому і полягає суть [абстракції](https://uk.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D1%96%D1%8F_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85). Абстрактний тип даних визначає набір [функцій](https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), незалежних від конкретної реалізації типу, для оперування його значеннями. Конкретні реалізації АТД називаються [структурами даних](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85). В програмуванні АТД подаються зазвичай в вигляді інтерфейсів, які приховують відповідні реалізації типів. Ми працюємо з абстрактними типами даних лише через ці інтерфейси, оскільки реалізація може стати іншою в майбутньому. Такий підхід – це застосовний в об’єктно-орієнованому програмуванні принцип інкапсуляції. Основне в цьому підході – приховування реалізації. Ми будемо його використовувати, бо абстрактні типи даних допомагають досягти модульності програмних продуктів і мати кілька альтернативних взаємозамінних реалізацій окремого модуля. Наш шлях розробки має багато переваг, крім інкапсуляції, ще , наприклад, в АТД є локалізація змінних і гнучкість. Локалізація змінних є перевагою, бо код, який використовує об’єкт АТД не потрібно буде редагувати, якщо реалізація АТД змінилася. Оскільки будь-які зміни в реалізації як і раніше повинні відповідати інтерфейсу, а так як код з використанням об’єкту АТД може відноситися тільки до властивостей і здібностей, зазначених в інтерфейсі, можуть бути зроблені зміни в реалізації, при цьому не вимагаючи будь-яких змін в коді, де використовується АТД. Щодо гнучкості, різні варіанти реалізації АТД мають всі ті ж властивості і здібності, котрі еквівалентні і можуть бути використані як взаємозамінні в коді, який використовує АТД. Це дає велику гнучкість при використанні об’єктів абстрактного типу даних в різних ситуаціях. Наприклад, різні реалізації АТД можуть бути більш ефективними в різних ситуаціях. Можна використовувати кожен з них в ситуації, коли він є кращим, збільшуючи таким чином загальну ефективність.

**Для оброблення даних моєї курсової я використовуватиму клас StationArray – який буде мати список як структуру даних і Station як АТД. Він виконуватиме основні функції, які необхідні для коректної роботи програми. Station – абстрактний тип даних, який має 2 потрібні елементи – назва станції і номер станції.**

Для повернення інформації моєї курсової я буду використовувати клас Return. Return – це контейнер зі структурою даних list, який зберігає всю оброблювану інформацію з мого API. В ньому буде міститися інформація про назву поїзда, час, місто відправлення, місто прибуття та інше. Кожне зі значень зберігається у підрозділах. Ці підрозділи називаються RouteInfo. Їхня структура даних – словник всієї необхідної інформації по даному маршруту. Літери отримуються від назви маршруту поїзда, який рухається між містом А(перші три літери) і містом Б (інші три літери). Наприклад, LVI-UZG, що означає: Маршрут проходить зі Львова (LVIV) до Ужгорода (UZGOROD). Такий поділ я вирішив здійснити для зручності, бо програмі буде потрібно використовувати правильну інформацію під час подорожі клієнта і забезпечувати його комфортну мандрівку. По суті, Return – cgbcjr, де його елементами є екземпляри АТД RouteInfo. Так як RouteInfo буде мати словник інформації, у Portfolio буде записуватися інформація як один елемент списку.. Я використовую список як структуру даних для реалізації мого абстрактного типу даних, бо він не зважає на повторюваність елементів.

Через те, що наші залізничні шляхи не пристосовані до подорожі через всю Україну, можуть траплятися випадки, коли нам необхідно буде повернутися до місця, з якого ми починали певний етап нашої подорожі. Проблема полягає в тому, що назви маршрутів незмінні, незважаючи на те, чи їхати туди чи назад. Наприклад, маршрут Львів-Пшемисль визначає рух поїзда між цими двома містами. Якщо пасажир рухається з Пшемисля до Львова, АРІ «Укрзалізниці» дає назву(нехай вже оброблену нами) LVI-PRZ. І якщо цей же пасажир рухається зі Львова до Пшемишля, назва залишається такою як є - LVI-PRZ. Тому до нашого абстрактного типу даних Particle і до основного «складу» інформації Portfolio ми повинні додати операцію перевертання назви АТД. Тоді словник буде мати різні дані для кожного з однакових за назвою маршрутів.

Деякі з цих маршрутів можна буде опускати, тобто за бажання користувача ми можемо видаляти повторюваний маршрут і рухатися далі.

Мій абстрактний тип даних RouteInfo буде містити наступні операції:

Add() – додавання аргументів до словника.

Delete() – видалення аргументів зі словника.

Information() – повернення ініціалізованого словника

Пустий маршрут (якщо такий можливий) буде представлений як пустий список в Particle і як пустий словник відповідно в Portfolio.(Тоді на екран буде вибиватися повідомлення про помилку). Довжина словника зазвичай буде сталою, бо для кожного маршруту нам потрібні ті самі дані. Довжина списку класу Return відповідатиме кількості пересадок і відповідно кількості маршрутів. Визначення чи ми мусимо повернутися назад, щоб продовжити нашу «навколодержавну» подорож допоможе зорієнтувати користувача, чи хоче він робити даний «маневр», чи бажає просто опустити цей пункт призначення. Додавання/Видалення елементів буде здійснюватися за стандартною процедурою словника.

Мій загальний контейнер даних Return буде містити такі операції:

Add() – додавання аргументів до списку

**Для успішної праці з даними я використовуватиму клас StationArray, він міститиме наступні методи:**

add() – додавання елементів до списку

array() – виведення списку на екран

delete() – видалення елементу зі списку

getitem() – отримання елементу списку за індексом

get\_station\_num(params, headers) – отримання номеру станції за її назвою

get\_stations(station\_list) – отримання всіх станцій і їх назв, запис в ініціалізований список

get to another city(st1, st2, time, result) – простий перехід з одного міста в інше

region across – створення повного маршруту навколо певного регіону

**Клас Station має такі методи:**

station\_name() – повертає ініціалізовану назву станції

station\_number() - повертає ініціалізований номер чи особистий код станції

Ці методи допоможуть мені працювати з моїми даними швидше і ефективніше.

Я використовую 2 абстрактних типи даних, бо мені потрібно також представляти результати в окремій формі.