**Міністерство освіти і науки України**

**Національний авіаційний університет**

**Навчально-науковий інститут комп’ютерних**

**інформаційних технологій**



Курсова робота

з дисципліни «Об’єктно орієнтоване програмування»

Варіант 5

Виконав студент:

групи ПІ-223Б

Карпов І.Б.

Перевірив викладач:

Дишлевий О.П.

Київ 2022

# **Завдання**

1. В якості типу застосування обрати консольне застосування з командним рядком, як виняток, win forms, wpf.
2. Спроектувати та реалізувати систему класів, в основу якої покладено логічну структуру даних, наведену у варіанті, для накопичення та обробки даних домену відповідно варіанту курсової роботи.
3. Структура програмної системи курсового проекту:  
   Код програмної системи має складатись не менш як з трьох частин (окремих проектів) відповідно до багатошарової архітектури системи, де шарами архітектури є: шар (рівень) доступу до даних (DAL), шар бізнес-логіки (BLL), шар представлення (інтерфейс програмної системи) (PL). Тип програмного модулю – DLL.  
   Шар доступу до даних організувати таким чином, щоб він забезпечував збереження даних у файлах за допомогою серіалізації або у БД із застосуванням Entity Framework.  
   Тип програмного модулю – DLL.  
   В шарі бізнес-логіки побудувати класи, що представляють об’єкти та дії над ними відповідно до предметної області, наприклад, читачі, книги та абонементи, читач може взяти книгу на свій абонемент при наявності у сховищі та повернути її у заданий термін. Для створення наборів об’єктів предметної області (студенти, викладачі, книги, замовлення, страви, тощо) використовувати класи-узагальнені колекції. Всі операції бізнес-логіки використовують об’єктну модель даних. Якщо дані потрібно отримати зі сховища даних або зберегти у сховищі, шар бізнес-логіки звертається до шару доступу до даних, передаючи\отримуючи збережені дані.  
   Шар представлення – це додаток типу відповідного до обраного інтерфейсу (консольний чи GUI). Людина-користувач через шар представлення взаємодіє з програмною системою через інтерфейс, реалізований шаром представлення. Для виконання дій на вимогу користувача шар представлення звертається до шару бізнес-логіки, передаючи в нього ведені користувачем дані та команди виконати певні дії. Результати виконаних дій, отримані від шару бізнес-логіки, шар представлення відображує користувачу у консолі\формах.  
   **Принцип багатошаровості при будуванні архітектури є обов’язковим до застосування!**
4. Реалізувати обов’язковий контроль коректності введених даних в шарі представлення.
5. Реалізувати перевірку виняткових ситуацій в шарі бізнес-логіки та доступу до даних; в тому числі – при роботі з даними. При необхідності створити власні класи виключень (наприклад, виключення для ситуації перевищення ліміту кількості книжок на абонементі в бібліотеці). Генерація/ виникнення вийняткових ситуацій, як правило відбувається на іншому рівні (рівнях), ніж їх обробка.
6. Написати модульні тести, використовуючи певний фреймворк (MSTest, Nunit, Xunit, тощо), до бізнес логіки. Мінімальне покриття тестами – 100% функціоналу однієї з сутностей, а також – мінімум 50% функціоналу усіх інших сутностей. Покриття можна продемонструвати відповідними засобами, наприклад AxoCover, CodeCoverage та ін. (допускається також детальне описання функціоналу, покритого модульними тестами). Модульні тести повинні бути окремим проектом в рішенні. Для оформлення коду модульних тестів обов’язково застосовувати принцип Triple A.
7. Сутності за зв’язки між ними повинні бути спроектовані, відповідно до базових правил ООП, composition over inheritance, loose coupling – high cohesion, inversion of control (IoC). А також - з використанням узагальнень.
8. При написанні коду застосувати правила «хорошого» стилю програмування. А саме: створювати мнемонічні ідентифікатори, чітко форматувати код, застосувати при необхідності коментарі. Не використовувати public полів у класах. За необхідності доступу до них використати властивості, індексатори чи окремі методи-аксесори. Опис класів наводити в окремих файлах. За неохайне оформлення коду можливе зниження оцінки за курсову роботу!
9. Пояснювальна записка має містити наступні складові:
10. Титульний лист
11. Зміст
12. Опис завдання
13. Опис шарів проекту та загальна архітектура проекту
14. Опис інших компонентів проекту (при наявності)
15. Діаграма класів з усіма зв’язками та поясненнями
16. Описання деяких особливостей реалізації (допускається приведення прикладів з кількома рядками коду)
17. Описання функціоналу та використання (допускається використання кількох скрішнотів)
18. Використані джерела
19. Лістінг коду проекту (не друкувати, а викласти в репозиторій/ додати архівом)
20. Скріншоти (результати введення\виведення) проекту (не друкувати, а викласти в репозиторій/ додати архівом)
21. До розробленого застосування та оформленої пояснювальної записки розробити презентацію та доповідь (приблизно 5 хв.).

**Завдання згідно з варіантом**

|  |  |
| --- | --- |
| № варіанту | Домен розробленої системи |
| 5 | Біржа праці: ведення особових справ безробітних в центрі зайнятості |

*Функціональні вимоги до програмного забезпечення*

1. Управління вакансіями і резюме
   1. Можливість додавати вакансію і резюме до категорії
   2. Можливість видаляти вакансію і резюме з категорії
   3. Можливість змінювати дані вакансій і резюме
   4. Можливість переглянути дані конкретної вакансії і резюме
   5. Можливість переглянути відсортований список всіх вакансій
   6. Можливість переглянути відсортований список всіх вакансій
2. Управління безробітними
   1. Можливість додавати безробітних
   2. Можливість видаляти безробітних
   3. Можливість змінювати дані безробітних
   4. Можливість переглянути дані конкретного безробітного
   5. Можливість переглянути список всіх безробітних
      1. Можливість відсортувати список по імені
      2. Можливість відсортувати список по прізвищу
3. Управління фірмами замовниками
   1. Можливість додати замовників
   2. Можливість видаляти замовників
   3. Можливість змінювати дані замовників
   4. Можливість переглянути дані конкретного замовника
   5. Можливість переглянути список всіх замовників
      1. Можливість відсортувати список по імені
      2. Можливість відсортувати список по прізвищу
4. Пошук
   1. Можливість пошуку по ключовому слову серед вакансій
   2. Можливість пошуку по ключовому слову серед безробітних

# **Опис шарів проекту та загальна архітектура проекту**

## Відповідно до варіанту завдання було реалізовано програму «Labor Exchange», яка має наступні рівні:

## **DAL – найнижчий рівень, рівень даних**

* 1. Містить класи сутностей, з якими ведеться робота:
     1. Category – клас було додано із логічних висновків, оскільки в завдані сказано, що вакансії та резюме можуть бети додані до певної категорії. Містить ім’я категорії та список вакансій і резюме;
     2. CustomerCompany – клас, що представляє компанію замовник. Містить наступні дані: ім’я компанії, ім’я та прізвище замовника, опис компанії, номер телефону, місто, адресу, домашню сторінку;
     3. Resume – клас, що представляє резюме. Містить наступні дані: позицію, на яку претендує людина, ім’я та прізвище людини, вік, місто, e-mail та номер телефону;
     4. Unemployed – клас, що представляє безробітного. Містить наступні дані: ім’я та прізвище безробітного, вік, місто та список резюме безробітного;
     5. Vacancy – клас, що представляє вакансії. Містить наступні дані: назву вакансії, опис вакансії, зарплату, компанію замовник та місто (місто є в тому випадку, якщо є компанія замовник).
  2. Даний рівень містить два інтерфейси (IEntity, IContainsOtherEntities):
     1. IEntity – даний інтерфейс реалізовують всі класи сутностей. Описує методи IEntity SetData(EntityOptions options) та void ChangeData(EntityOptions options). Перший метод використовується при створені екземпляру класа сутності в класі EntityCreator, другий – для зміни даних сутності;
     2. IContainsOtherEntities – даний інтерфейс реалізовують класи Category, Unemployed та Vacancy. Описує три методи: void AddEntity(IEntity entity), void DeleteEntity(IEntity entity) та void DeleteEntity(IEntity entity, int index). Перший метод потрібен для додавання однієї сутності до іншої (наприклад, додати резюме до категорії), інші два для видалення (перший видаляє по параметрам, другий – по індексу).
  3. Інші класи:
     1. EntityOptions – містить список з усіх властивостей класів сутностей. Потрібен для створення екземпляру класу сутності та зміни даних сутності;
     2. EntityCreator – узагальнений клас, який створює сутність заданого типу за переданими параметрами. Для коректної та правильної роботи узагальнення має обмеження: where T : class, IEntity, new();
     3. EntityContext – узагальнений клас, який відповідає за читання(десеріалізацію) та запис(серіалізацію) даних у файли.
  4. Виключення:
     1. В рівні DAL є лише одне виключення – TheFileIsEmptyException. Вказує на спробу прочитати порожній файл.

## **BLL – рівень бізнес логіки**

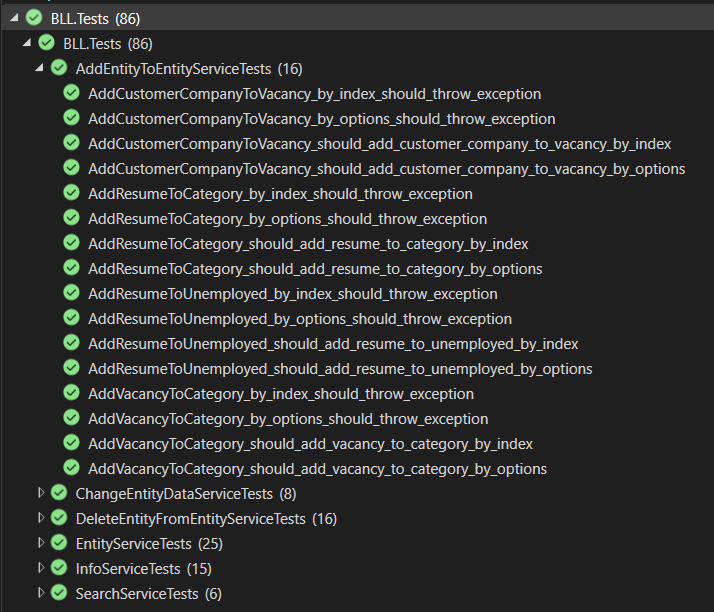
* 1. Головними класами є класи-сервіси:
     1. EntityService – містить методи додавання сутностей за переданими з рівня PL параметрами, методи видалення сутностей за переданими з рівня PL параметрами або за індексом, а також узагальнені приватні методи, які потрібні для того, щоб позбутися дублювання коду;
     2. AddEnityToEntityService – містить методи для додавання однієї сутності до іншої за переданими з PL параметрами, або по індексу. Сутність до якої додавати визначається по індексу. Також містить узагальнені методи, які потрібні, щоб позбутися дублювання коду;
     3. DeleteEntityFromEntityService – містить методи для видалення однієї сутності до іншої за переданими з PL параметрами, або по індексу. Сутність з якої видаляти визначається по індексу. Також містить узагальнені методи, які потрібні, щоб позбутися дублювання коду;
     4. ChangeEntityDataService – містить методи для зміни даних сутності за переданими з PL параметрами та індексом. Також містить узагальнені методи, які потрібні, щоб позбутися дублювання коду;
     5. InfoService – отримує дані потрібної сутності, та повертає відсортований (за потрібності) список DataPresenter’ів з заповненими даними;
     6. SearchService – отримує з InfoService потрібний список DataPresenter’ів та по ключовому слову вибирає з них ті, в яких міститься ключове слово, потім повертає список DataPresenter’ів зі збігами.
  2. Інші сутності:
     1. EntityOptions – такий самий, як і в DAL, тільки добавилось поле для шляху до файлу. Клас потрібен, щоб не порушувати залежності проектів, і не звертатись в PL до EntityOptions з DAL;
     2. OptionsConverter – конвертує BLL.EntityOptions в DAL.EnityOptions;
     3. DataPresenter – містить властивості для індексу конкретного DataPresenter та словник для заповнення його даними в InfoService;
     4. SortingType – енумератор з типами сортування. Використовується в InfoService та в PL.
  3. Виключення:
     1. EntitiesEmptyException – вказує на порожній список сутностей;
     2. NoMatchesFoundException – вказує на відсутність збігів при пошуку по ключовому слову.

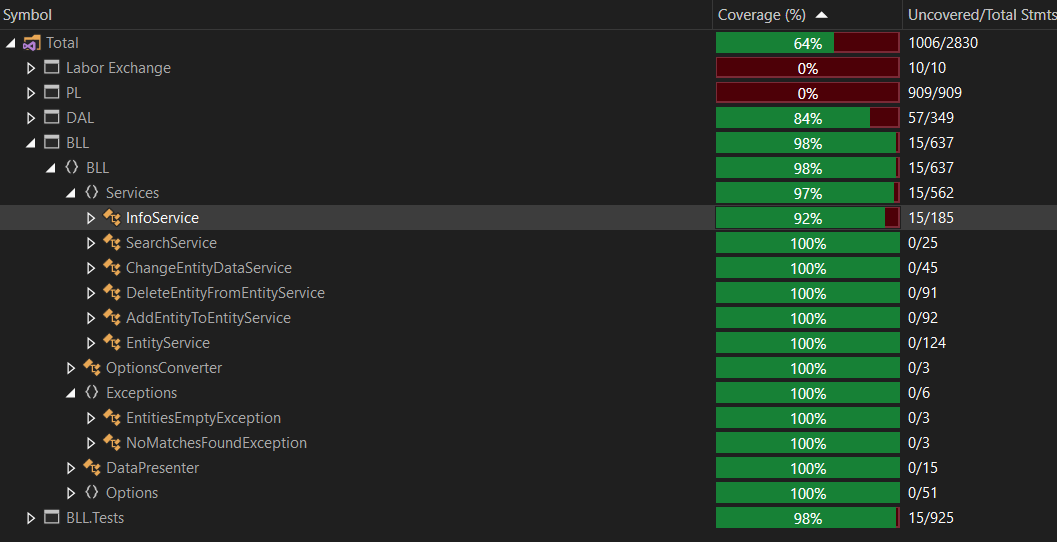
## **PL – рівень представлення**

* 1. Консольне меню. Містить чотири класи (Menu, MenuItem, MenuList, WriteBuffer), за допомогою яких реалізовується консольне меню з керування стрілками на клавіатурі та клавішою Enter.
  2. CommandsHandler – partial клас, який містить всі можливі дії для меню.
  3. ConsoleWorker – додатковий шар над класом Console. Клас дозволяє друкувати кольоровий текст на кольоровому фоні. Замінює виклик методів з Console у всьому PL;
  4. ConsoleInterface – клас, що має лише метод Start, який викликається в Program. В цьому класі відбувається конструювання консольного меню та його запуск;
  5. InputHandler – клас для отримання даних від користувача;
  6. InputChecker – викликається в InputHandler та за переданим regex шаблоном (шаблони зберігаються в файлі конфігурацій App.config) перевіряє введені дані.

# **Опис інших компонентів проекту**

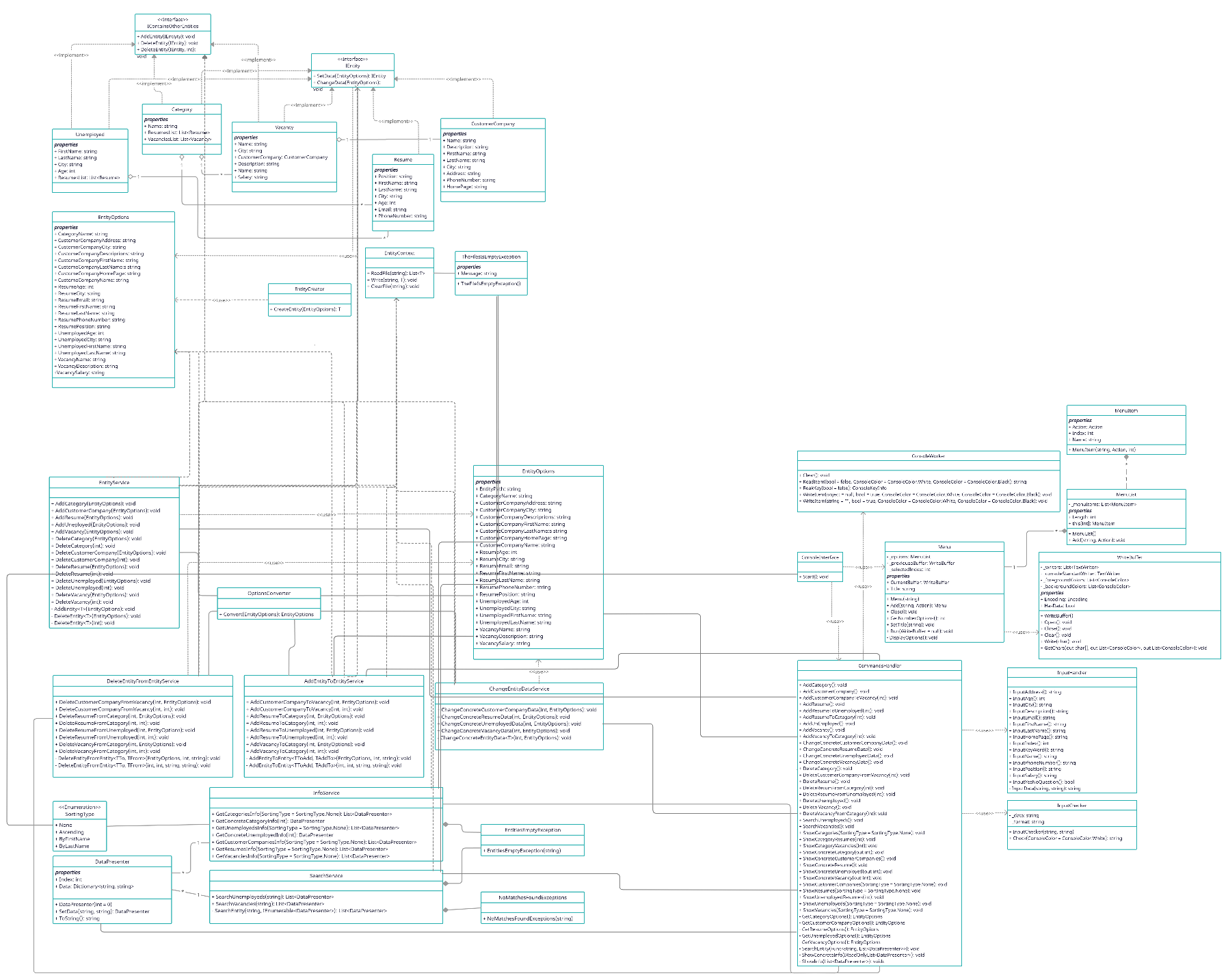
До інших компонентів відноситься проект тестування бізнес логіки. Тестування проводиться для кожного сервісу BLL, складається з 86-ти тестів та має майже 100% покриття усього BLL.





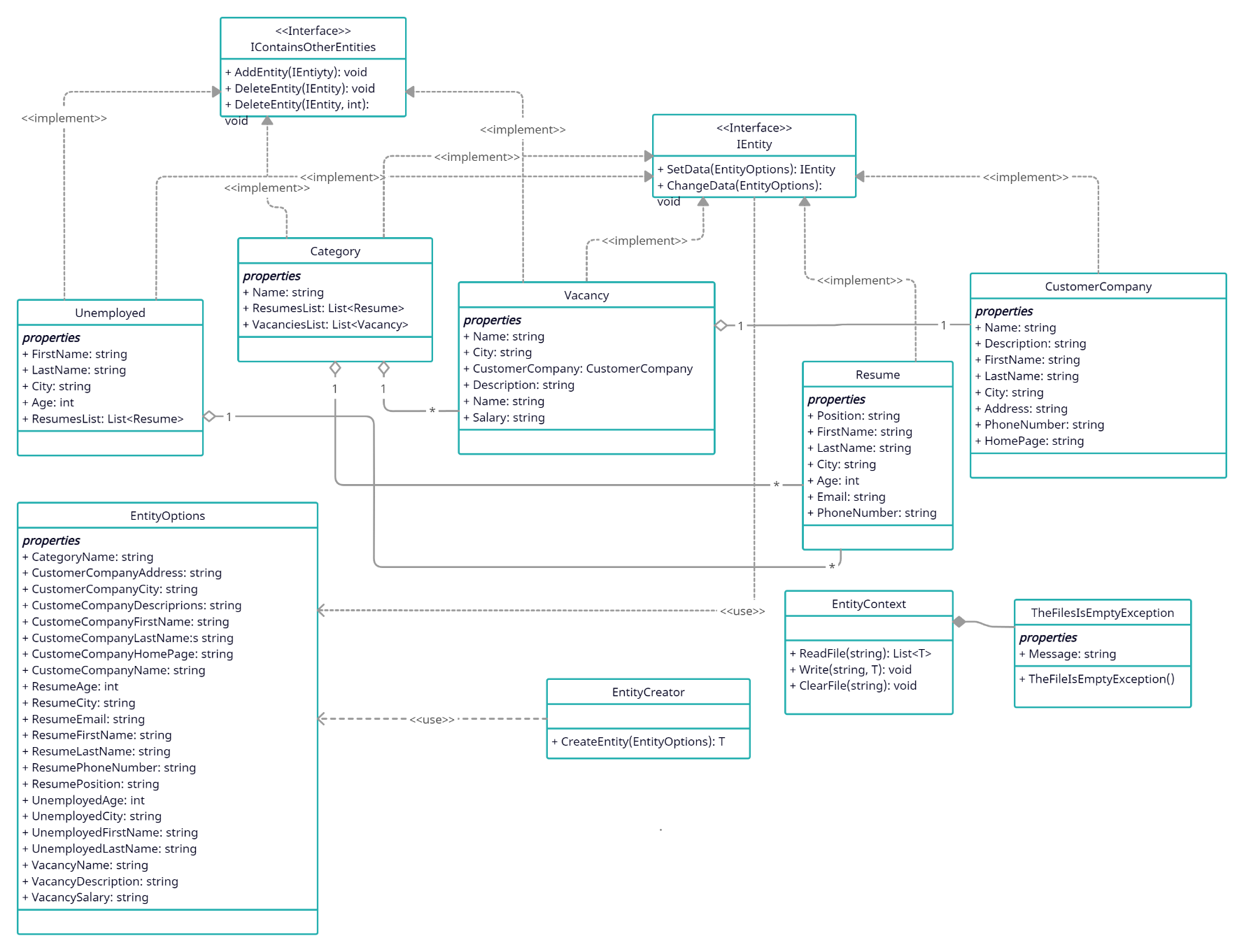
# **Діаграма класів з усіма зв’язками та поясненнями**

Загальна діаграма класів:



Верхня видима група – DAL, середня видима група – BLL, права видима група – PL.

Діаграма DAL:



Класи Unemployed, Category, Vacancy, Resume та CustomerCompany, як і говорилось раніше, реалізовують інтерфейс IEntity. Класи Unemployed, Category та Vacancy реалізовують інтерфейс IContainsOtherEntity.

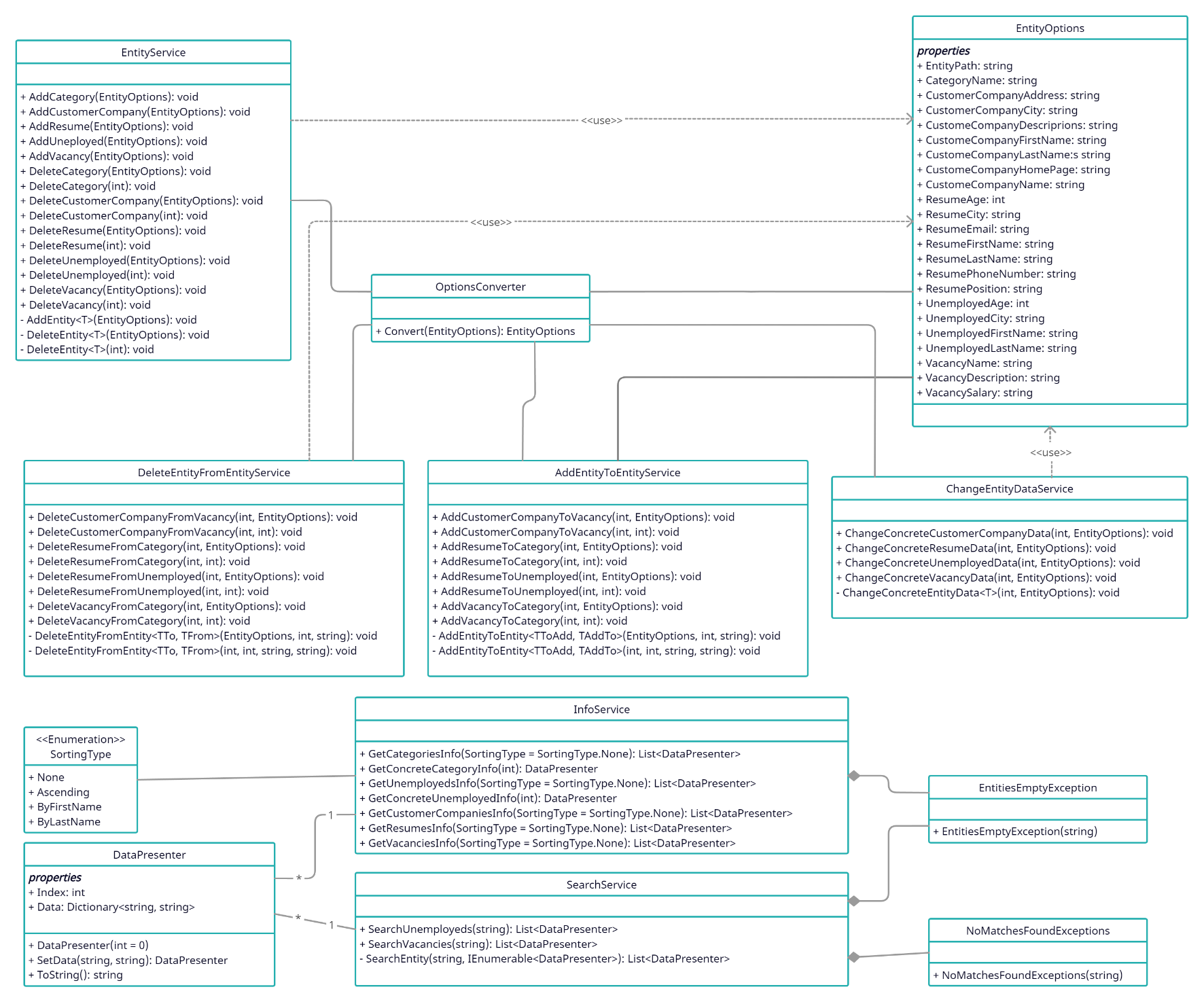
Інтерфейс IEntity та клас EntityCreator залежать від класу EnityOptions.

Зв’язок між класами Unemployed та Resume, Category та Vacancy, Resume, Vacancy та CustomerCompany визначається як агрегація.

Зв’язок між EntityContext та TheFileIsEmptyException – композиція.

У вищи рівнях взаємодія відбувається, в основному, за рахунок EntityContext та EntityCreator.

Діаграма BLL:

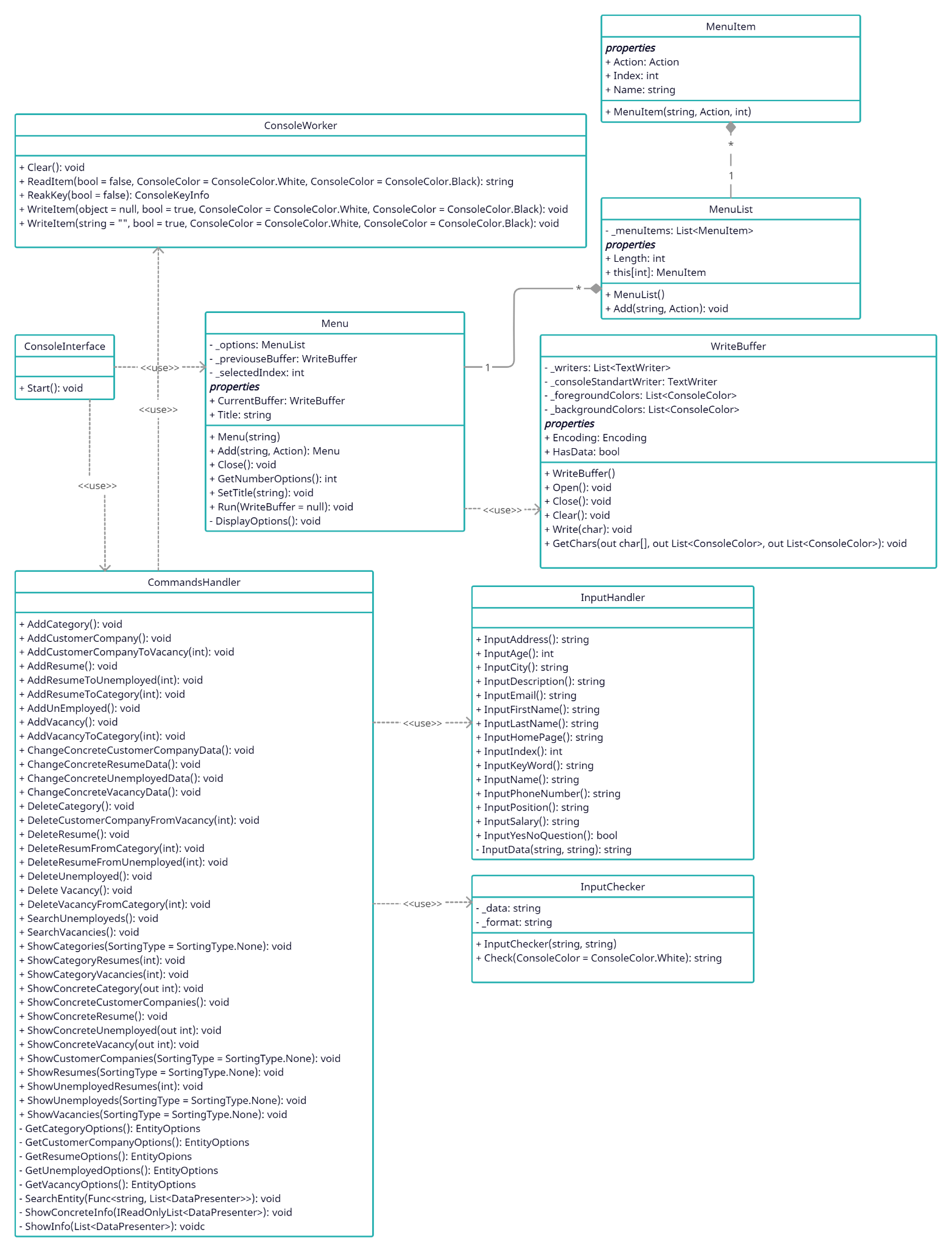


У даному рівні взаємодія між класами відбувається на основі залежностей та асоціацій.

Класи цього рівня в основному використовують або сутності з нижчого рівня, або класи цього ж рівня, для виконання своїх завдань.

Також, сутності даного рівня використовуються у рівні представлення для взаємодії із рівнем даних або виконанням певних дій.

Діаграма PL:



Даний рівень використовує рівень BLL для взаємодії із даними. Як видно з загальної діаграми класів, класи найвищого рівня взаємодіють із рівнем даних через рівень бізнес логіки. Таким чином, у рівні представлення не має використання типів, визначених на рівні даних.

У даному рівні також найпоширенішим типом зв’язку є залежність (використання інших класів).

# **Описання деяких особливостей реалізації**

Однією з ключових особливостей реалізації є клас рівня даних EntityCteator. Це узагальнений клас, який потрібен для того, щоб за переданими параметрами створювати потрібні сутності.

namespace DAL

{

public static class EntityCreator<T> where T : class, IEntity, new()

{

public static T CreateEntity(EntityOptions options) => new T().SetData(options) as T;

}

}

Клас побудований таким чином, що він може працювати правильно навіть зі збільшенням кількості сутностей. Дана роботи досягається за рахунок обмеження узагальненого типу як class, IEntity, new(). Сутності які буде створювати даний клас повинні реалізовувати інтерфейс IEntiy та мати конструктор без параметрів.

Недоліком даної реалізаціє є те, що кожен раз, коли буде створюватись нова сутність, пртрібно буде доповнювати клас EntityOptions її полями та властивостями.

Власне кажучи, значущою особливістю реалізації є класи EntityOptions (рівні DAL та BLL). Як говорилось вище, ці класи зберігають в собі ті ж властивості, що і класи сутностей. Під час створення екземпляру цього класу, потрібні дані передаються в конструктор за потрібними іменами: string resumeFirstName = "", string resumeLastName = "", int resumeAge = int.MinValue, string resumeCity = "". Параметрам заздалегіть присвоєї значення за замовчуванням, що унеможливлює випадки, коли змінним нічого не присоєно і дає можливість задавати лише потрібні параметри.

Далі, під час створення екзмемпляру класу в класі EntityCreator, чи інших операцій, потрібні дані дістаються та використовуються:

public IEntity SetData(EntityOptions options)

{

FirstName = options.UnemployedFirstName;

LastName = options.UnemployedLastName;

Age = options.UnemployedAge;

City = options.UnemployedCity;

ResumesList = new List<Resume>();

return this;

}

# **Описання функціоналу та використання**

У програмі «Labor Exchange» користувач може виконувати всі дії, відповідно до завдання варіанту.

Взаємодія з програмою відбувається шляхом натискання стрілок вгору чи вниз та Enter.

Обробка натискання клавіш відбувається в класі Menu методі Run. Виклик потрібного функціоналу при натисканні клавіші Enter відбувається за допомогою подій.

# **Лістинг коду**

Код доступний за посиланням:

<https://github.com/IvanKarpov-1/course2-term1-OOP-term-paper/>

# **Скріншоти**

Приклад роботи програми в форматі .gif та скріншоти доступні за цим самим посиланням.