

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

**Курсова робота**

**з дисципліни «Програмування»**

**на тему: «Шаблони проєктування.Магазин дитячих іграшок»**

Виконав:

студент II курсу, групи КП-03

Хоменко Максим Вячеславович

\_\_\_\_\_

Керівник роботи:

Доцент кафедри ПЗКС, к.т.н., доцент,

Заболотня Тетяна Миколаївна

\_\_\_\_\_

Оцінка \_\_\_\_\_

(дата, підпис)

Київ – 2022

## **ЗМІСТ**

<b>СПИСОК ВИЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ.....</b>	<b>3</b>
<b>ВСТУП.....</b>	<b>4</b>
<b>1. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ МАГАЗИНУ ДИТЯЧИХ ІГРАШОК.....</b>	<b>6</b>
1.1.Модульна організація програми.....	6
1.2.Функціональні характеристики.....	8
<b>2. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЄКТУВАННЯ.....</b>	<b>9</b>
2.1.Обґрунтування вибору та опис шаблонів проєктування для програмної реалізації системи магазину дитячих іграшок.....	9
2.2.Діаграма класів.....	21
2.3.Опис результатів роботи програми.....	23
<b>ВИСНОВКИ.....</b>	<b>28</b>
<b>СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....</b>	<b>29</b>

**ПЗ** – програмне забезпечення.

**БД** – база даних.

**Фреймворк** – програмне забезпечення, яке полегшує процес розроблення на об'єднання різних модулів програмного проекту.

**MVVM** – model, view, viewmodel; архітектурний шаблон проектування.

**DDL** - мова визначень даних.

**DML** - мова маніпулювання даними.

**DAO** - data access objects; об'єкти доступу до даних.

**EF CORE** - Entity Framework Core; бібліотека доступу до даних.

**DRY** - Don't Repeat Yourself; принцип "не повторюй себе".

**DIP** - Dependency Inversion Principle; принцип інверсії залежностей.

**SRP** - Single Responsibility Principle; принцип єдиної відповідальності.

**ОСР** - Open-Closed Principle; принцип відкритості-закритості.

## ВСТУП

Велика кількість іграшкових магазинів за своїм принципом роботи передбачають необхідність керувати даними про наявні товари в магазині: додавати іграшки, змінювати характеристики товару, видаляти дані про іграшку, а також надавати сервіс для купівлі товару - а також надання сервісу для придбання товарів.

Таким чином власники магазинів натрапляють на ряд проблем як-от: наявність повної інформації про товар, актуальність цієї інформації (параметри іграшки, наявність в асортименті), помилки під час реалізації товарів за невірною ціною, крадіжки, швидкість продажів.

Системи віртуальних магазинів вирішують ці проблеми, заощаджуючи чи не малі кошти власника магазину.

Рішення, презентоване в цій роботі, є аналогом для персонального комп'ютера з операційною системою MS Windows 10, 11.

*Об'єктом дослідження* даного проєкту є процес обліку даних про іграшки, наявні у віртуальному магазині.

*Метою роботи* є розроблення програмного забезпечення магазину дитячих іграшок з використанням шаблонів проектування.

Для досягнення визначеної мети необхідно виконати такі завдання:

- абстрагувати об'єкти предметної галузі;
- розробити структурну організацію ПЗ за допомогою застосування основних принципів ООП та шаблонів проектування;
- визначити та описати функціональні характеристики програми;
- обґрунтувати вибір шаблонів проектування, використаних для побудови програми;
- розробити дизайн інтерфейсу користувача;
- виконати реалізацію програмного забезпечення відповідно до

вимог технічного завдання;

- виконати тестування розробленої програми;
- оформити документацію з курсової роботи.

Розроблене ПЗ обліку магазину дитячих іграшок складається з 4-х модулів:

- модуль абстрактних змінювачів та отримувачів даних(сервіси) та моделей даних системи;
- модуль, що собою визначає реалізацію абстрактних сервісів даних;
- модуля, що керує роботою сервісів даних, згідно з логікою, визначеною предметною галуззю;
- інтерфейсу користувача.

Реалізовані шаблони проєктування: Фабричний метод, Фасад, Легковаговик, Посередник, MVVM, Команда, Спостерігач.

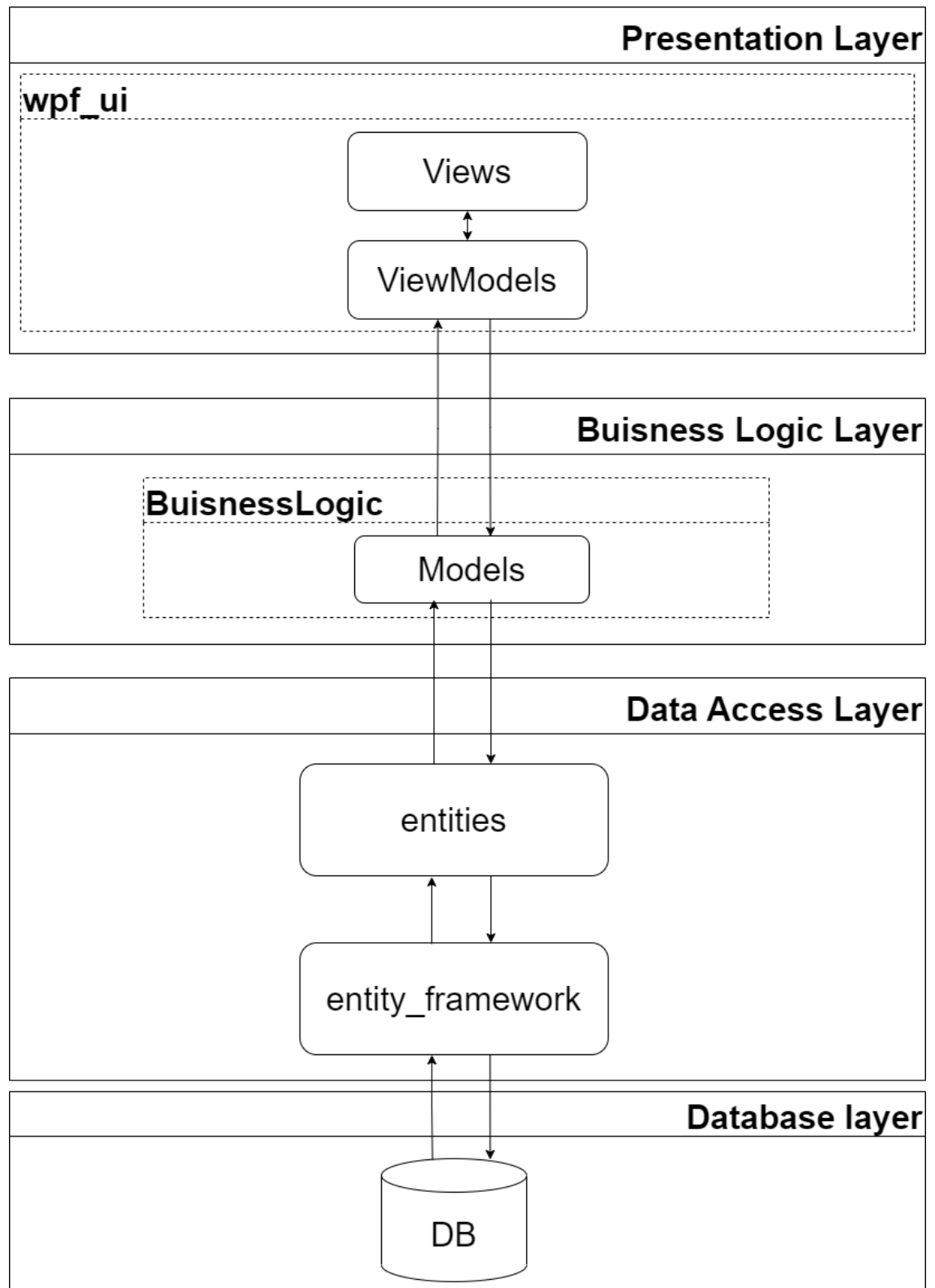
До функціональних можливостей програми належать: додавання, видалення, оновлення даних про товар, перевірки коректності вхідних даних при додаванні нового товару в магазин, маніпулювання товарами в кошику - додавання, видалення. Для функціонування розробленої програми необхідно забезпечити наявність на комп'ютері 2 ГБ оперативної пам'яті, від 800 МБ до 210 ГБ вільного місця на жорсткому диску, встановлений .Net 5.

Розроблене програмне забезпечення може бути використане магазинами іграшок для вирішення ряду проблем з метою заощадження та оптимізації бізнесу.

Пояснювальна записка складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел (3 найменування, з них 1 - іноземною мовою). Робота містить 16 рисунків. Загальний обсяг роботи – 30 друкованих сторінок, з них 25 сторінок основного тексту та 1 сторінка списку використаних джерел.

# 1. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ МАГАЗИНУ ДІТЯЧИХ ІГРАШОК

## 1.1. Модульна організація програми



### *Рис.1.1.1 Модульна організація програми*

Програмну систему умовно можна розділити на чотири рівні: рівень бази даних, рівень доступу до даних, рівень бізнес-логіки, презентаційний рівень (database layer, data access layer, business logic layer, presentation layer відповідно).

Одним із ключових є database layer. Саме на ньому знаходиться БД, з якою взаємодіє додаток. Цьому рівню притаманні DDL- та DML-операції, таблиці, зв'язки, кортежі даних. Головною ціллю цього рівня є збереження та модифікація даних, видача результатів пошуку шляхом SQL-запитів.

Наступним є рівень доступу до даних. Саме на ньому знаходяться модулі: “entities”, який описує об'єкти доступу до даних(DAO) та надає інтерфейси сервісів даних: сервіс отримання даних, сервіс модифікації даних. Реалізація цих інтерфейсів виконується модулем “entity\_framework”, що використовує відому бібліотеку “Entity Framework Core”.

На черзі - рівень бізнес логіки, на якому знаходяться класи, що виконують основну логіку програми завдяки сервісам даних з рівня доступу до даних.

Далі - презентаційний рівень. Містить модуль “wpf\_ui”, що використовує фреймворк WPF для побудови користувацького інтерфейсу. У свою чергу “wpf\_ui” використовує архітектурний шаблон проєктування MVVM. Відповідно, розбитий на частини: користувацький інтерфейс(views), класи, що підтримують стан вікна(viewmodels), класи, які визначають відображувані дані та які виконують бізнес логіку. Таким чином “wpf\_ui” відповідає за відображення та отримання вхідних даних.

## **1.2. Функціональні характеристики**

Система надає такі можливості як:

- управління асортиментом магазину: додавання, видалення, редагування товарів у магазині;
- управління вмістом кошика: додавання товару в кошик, видалення товару з кошика;
- реєстрація та вхід у систему.



## **2. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЄКТУВАННЯ**

### **2.1. Обґрунтування вибору та опис шаблонів проєктування для програмної реалізації іграшкового магазину**

#### **1) Фабричний метод**

**Визначення:** Породжувальний шаблон проєктування, що надає абстрактний інтерфейс (набір методів) для створення об'єкта-продукту, але залишає можливість класам-наслідникам вирішувати, екземпляр якого саме класу створити. Створення об'єкта-продукту видається безпечним, оскільки все, що потрібно для створення, знаходиться в одній точці програми.

**Проблема:** сервіси даних (`DataMutator<T>`, `DataProvider<T>`) для потокобезпечного доступу до БД мають створювати локальну область видимості з параметром-посиланням на контекст даних кожен раз перед виконанням CRUD-операції. Таким чином існує доволі велика ймовірність, що при розширенні системи, користувач, не посвячений у деталі того, як потрібно створювати контекст провайдера (наприклад MS SQL Server), може не вказати певні налаштування. Крім того, оскільки у кожному методі

сервіса даних зустрічатимуться одні й ті ж дії, напрямлені на створення контексту даних (на отримання параметра-посилання), принцип DRY буде порушено. Це призведе до ускладнення процесу внесення змін. До того ж, система сильно залежатиме від провайдера даних, що буде порушенням принципу DIP.

**Рішення:** створити інтерфейс, що специфікує метод створення контексту даних, створити єдину точку змін шляхом винесення процесу створення контексту даних у окремий клас, що реалізує вищезазначений

інтерфейс.

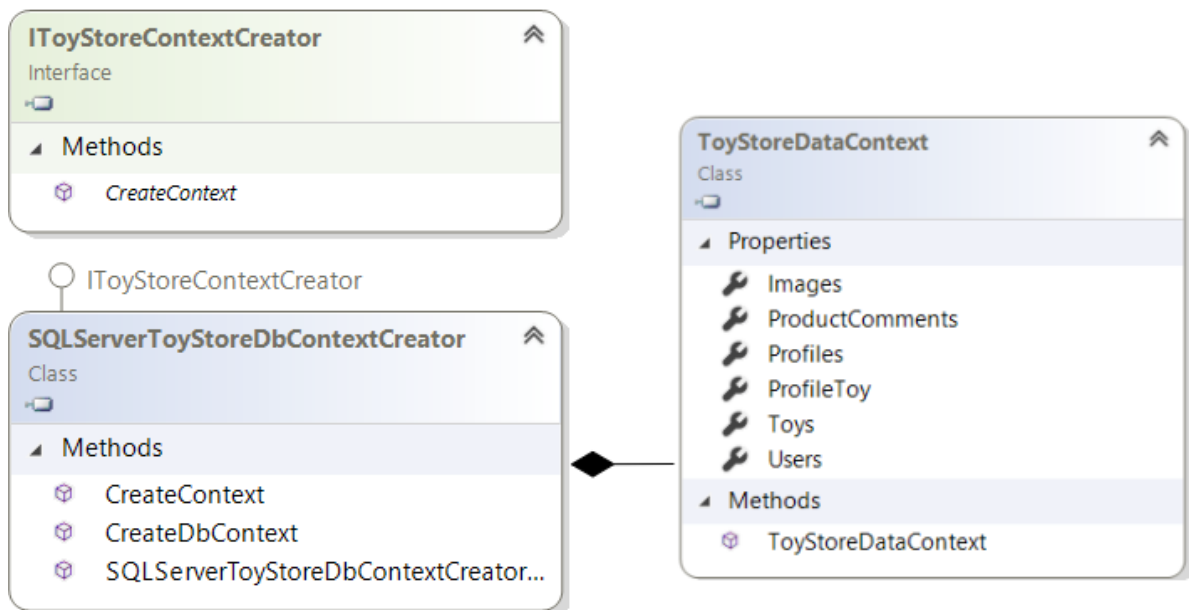


Рис.2.1.1 Діаграма класів, що входять до реалізації шаблону  
“Фабричний метод”

Інтерфейс креатора - IToyStoreContextCreator;

Клас-креатор, в якому визначено дії на створення контексту, що працює з провайдером MSSQLServer - SQLServerToyStoreDbContextCreator;

Клас-продукт, що створюється креатором - ToyStoreDataContext;

Клієнтський код, що споживає інтерфейс креатора - DataProviderService<T>, DataMutatorService<T>.

Результат: виконання принципів DRY, DIP, спрощення підтримки коду та додавання контекстів, що взаємодіють з іншими провайдерами даних.

## 2) Фасад

Визначення: структурний шаблон проектування, що надає єдиний інтерфейс до множини інших інтерфейсів у системі. Фасад визначає верхній інтерфейс, що робить систему легшою до використання.

Проблема: реалізація бізнес-логіки системи вимагає активної взаємодії дата-сервісів: модифікаторів та провайдерів - при чому різних сутностей. Щоб організувати цю взаємодію клас користувача має містити

необхідні залежності та бути в курсі деталей бізнес-логіки, що може результувати в порушення принципу SRP.

Рішення: для виконання певної бізнес-логіки об'єднати необхідні сервіси даних у окремий клас. Створити методи з вичерпними назвами дій, які потрібно виконати. Організувати взаємодію між сервісами даних у цих методах.

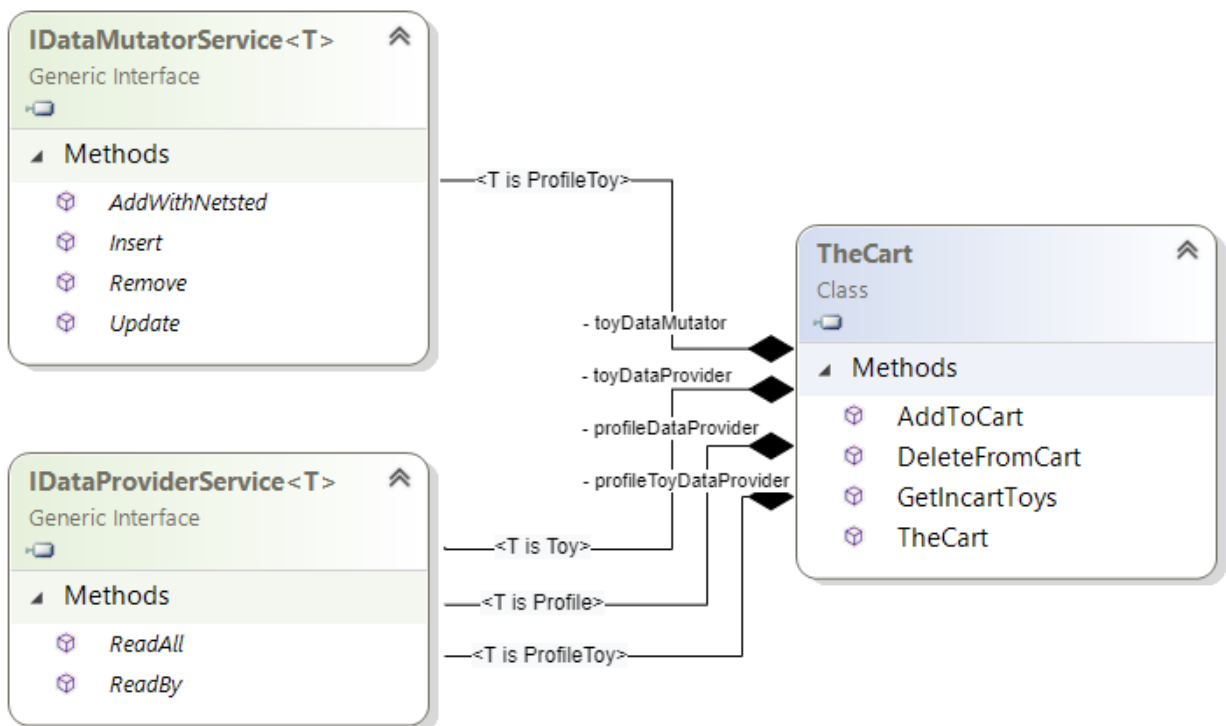


Рис.2.1.2 Діаграма класів, що входять до реалізації шаблону  
“Фасад”

*Підсистеми* - змінювач даних з типом ProfileToy та надавачі даних з типами Toy, Profile та ProfileToy;

*Фасад* - клас TheCart, виконує бізнес логіку кошика.

Результат: виконання принципу SRP, клієнт працює тільки через фасад, відповідно, зміни стосуються тільки коду фасаду, об'єднання складної логіки роботи дата-сервісів системи в зручні методи.

### 3) MVVM

Визначення: архітектурний шаблон проєктування, що розділяє UI на три частини сторінки користувацького інтерфейсу(views), класи, що підтримують стан UI-елементів у вікні(viewmodels), класи, які визначають відображувальні дані та виконують бізнес логіку.

Див *Рис.1.1.1 Presentation Layer & Business Logic Layer*

Проблема: нерідко в звичайних десктопних додатках бізнес логіка переміщується з логікою керування користувацьким інтерфейсом. Створення сильної залежності погіршує тестованість програми та є сприятливою умовою для виникнення запахів коду й порушення принципів SOLID.

Рішення: розділити UI-частину додатка на три директорії: Views, ViewModels, Models. Для кожної сторінки створити відповідний клас, що своїми властивостями характеризує поля сторінки. Створити класи, дані яких мають відображатися на сторінках.

Результат: слабка зв'язність між користувацьким інтерфейсом та бізнес-логікою. Можливість Unit-тестування, висока підтримуваність коду.

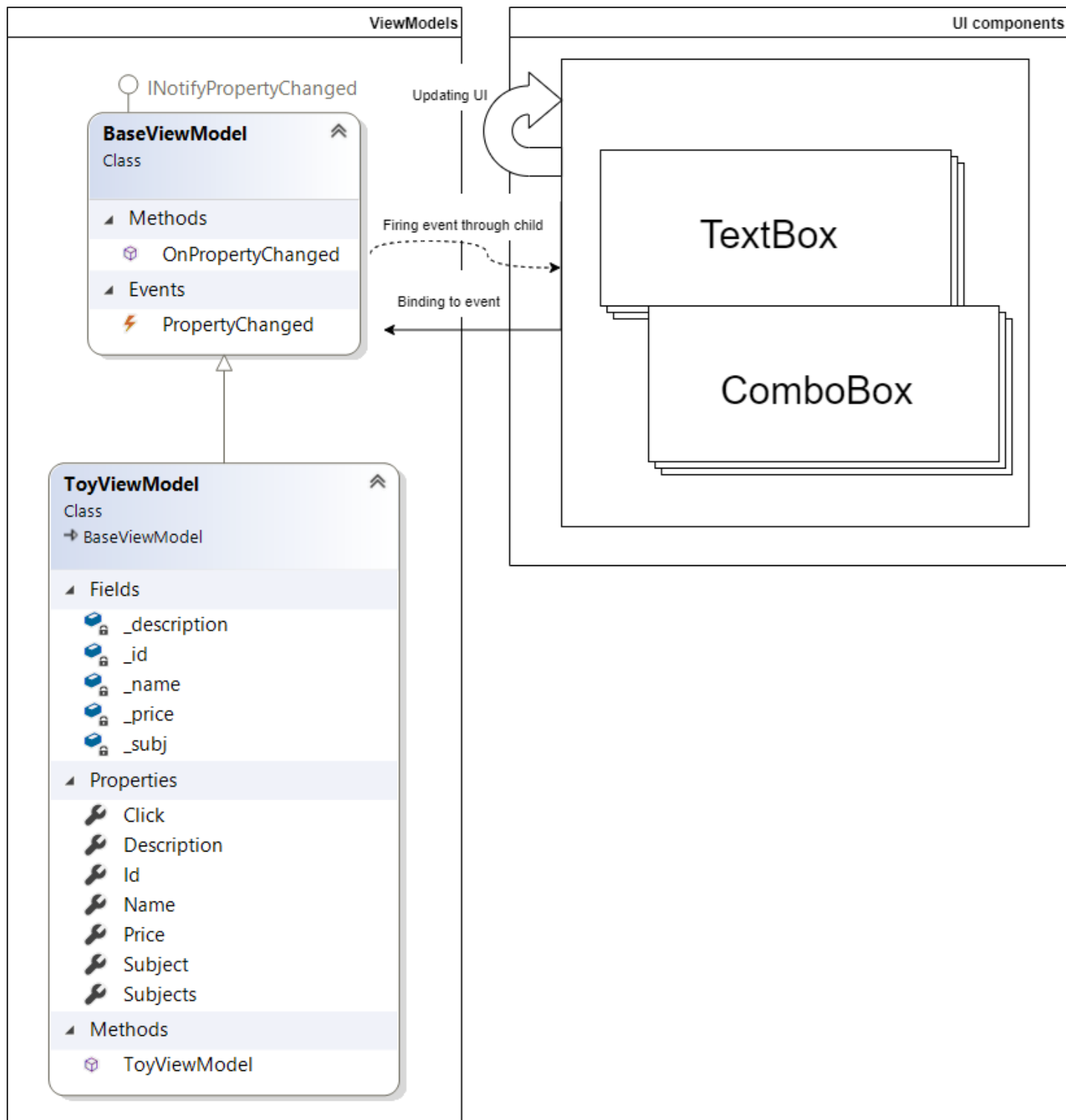
### 4) Спостерігач

Визначення: поведінковий шаблон проєктування, що визначає залежність один до багатьох між об'єктами таким чином, що коли один змінює свій стан, всі залежні є проінформовані та оновлені автоматично.

Проблема: залежність між UI-компонентом та відповідним полем ViewModel повинна бути визначена без тісного зв'язування цих об'єктів. Необхідно переконатися, що, коли одна ViewModel змінює стан, залежні від її властивостей UI-компоненти оновлюються автоматично. Повинно бути можливим, що ViewModel може сповіщати відкриту кількість UI-компонентів.

Рішення: розділення об'єктів системи: Views, ViewModels - на два типи: Видавці(ViewModels), Підписники(UI-компоненти). Створення базового класу, що пропонує подію “зміна властивості” та метод-тригер, що викликається класом-наслідником при зміні певної властивості з метою повідомлення UI-компонентів про подію для подальшої реакції - перемальовування, або оновлення відображення даних на сторінці.

Для демонстрації реалізації патерну на діаграмах оберемо лише одну ViewModel та схематично окреслити UI-компоненти, з якими працює клас.



*Рис.2.1.4 Діаграма класів і UI-компонентів, що беруть участь у реалізації Спостерігача*

Видавачі - класи ViewModels, складі яких наявні властивості;

Підписники - UI-елементи: текстові рядки, поля множинного вибору.

Результат: залежність між UI-компонентом та відповідним полем ViewModel визначена без тісного зв'язування цих об'єктів. Гарантія того, що, коли одна ViewModel змінює стан, залежні від її властивостей UI-компоненти оновлюються автоматично. Можливим є те, що ViewModel може сповіщати відкриту кількість UI-компонентів.

## **5) Команда**

Визначення: поведінковий шаблон, що інкапсулює запит в об'єкт, таким чином даючи можливість параметризувати клієнтський код різними запитами, залогувати, або закинути в чергу чи підтримувати необроблювані операції.

Проблема: усі кнопки, хоч і виглядають схоже, але виконують різні дії. Виникає запитання: куди розмістити код обробників кліків по цих кнопках? Найпростіше рішення — це створити підкласи для кожної кнопки та перевизначити в них методи дії для різних завдань. Але скоро стало зрозуміло, що такий підхід нікуди не годиться. По-перше, з'являється дуже багато підкласів. По-друге, код кнопок, який відноситься до графічного інтерфейсу, починає залежати від класів бізнес-логіки, яка досить часто змінюється. Проте, найгірше ще попереду, адже деякі операції в перспективі можна викликати з декількох місць: натиснувши кнопку для виходу або натиснувши клавіші Ctrl+E наприклад. Коли в програмі були тільки кнопки, код збереження був тільки у підкласі. Але тепер його доведеться продублювати ще в два класи.

Рішення: пропонує більше не надсилати виклики з кнопок користувацького безпосередньо. Замість цього кожен виклик, що відрізняється від інших, слід звернути у власний клас з єдиним методом, який і здійснюватиме виклик.

Для того, щоб демонстрація реалізації шаблону не була громіздкою, оберемо пару класів команд.

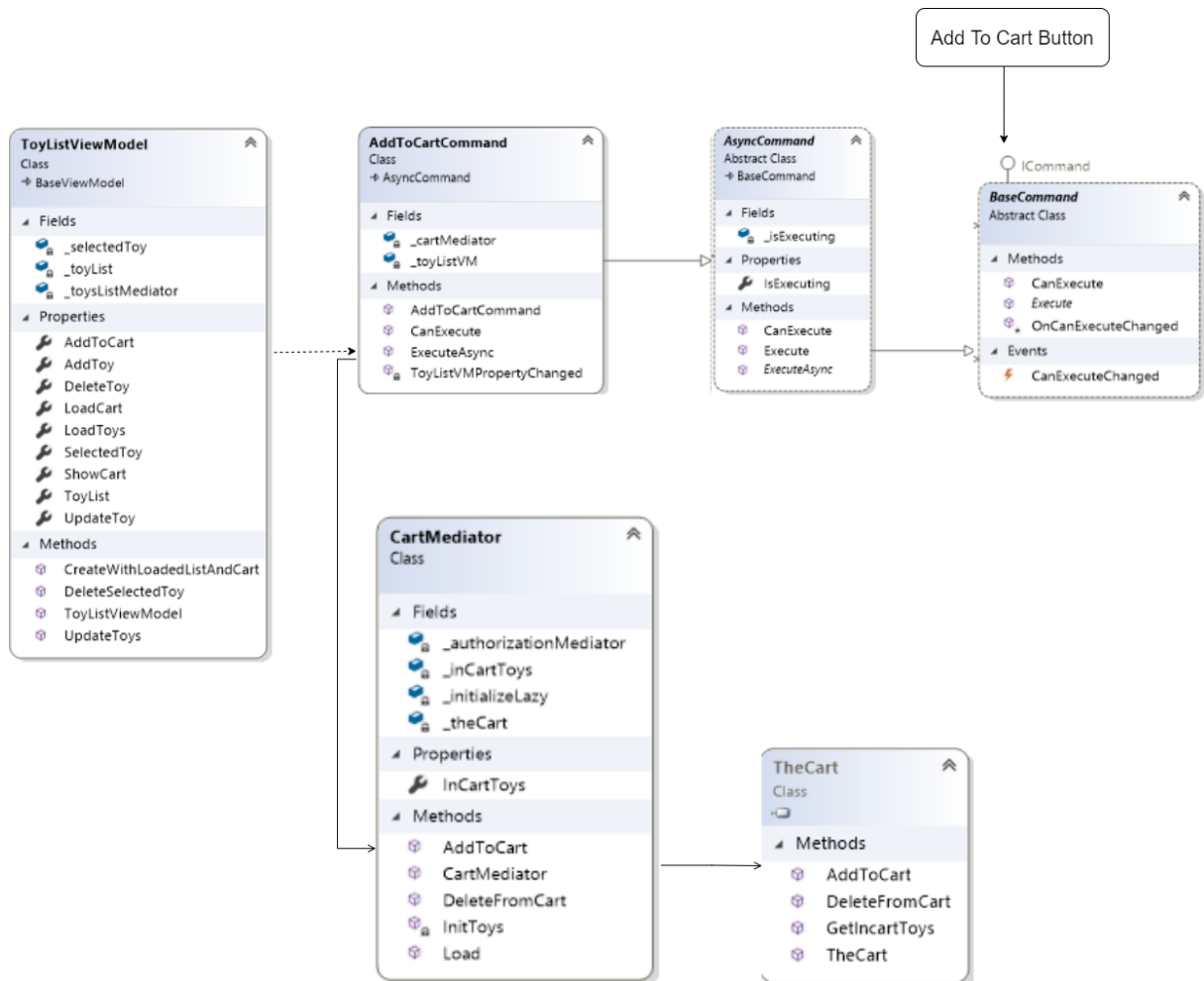


Рис.2.1.5 Діаграма частини класів, що беруть участь у реалізації Команди

*Invoker* - кнопка “Add to cart” у користувацькому інтерфейсі.

*Command* - інтерфейс **ICommand**, що специфікує методи `Execute` & `CanExecute`.

*Concrete Command* - клас **AddToCartCommand**, що викликає на виконання метод **CartMediator** - `AddToCart`.



Receiver - клас CartMediator, що викликає на виконання метод бізнес-логіки.

Результат: більше не потрібно буде створювати безліч підкласів кнопок для різних дій. Буде достатньо одного класу з полем для зберігання об'єкта команди. Виконується принцип ОСР. Відсутність прямої залежності між об'єктами, що викликають операції, та об'єктами, які їх безпосередньо виконують.

## **6) Легковаговик**

Визначення: поведінковий шаблон, що інкапсулює запит в об'єкт, таким чином даючи можливість параметризувати клієнтський код різними запитами, залогувати, або закинути в чергу чи підтримувати необроблювані операції.

Проблема: для кожного екземпляру ToyViewModel, що відображає певний товар у магазині, потрібно отримувати колекцію значень перелічення "Тематики"(Subject), яка відображається у полі множинного вибору(combobox) . Якщо таких товарів 10 - витрати пам'яті та часу на перебір і конвертування значень перелічення є незначними. А якщо таких товарів 1000?

Рішення: при першому зверненні на підвантаження певного перелічення, проводити операції напрямлені на отримання колекції та зберігати її у словник, ключем якого є тип перелічення, а значенням - назви констант перелічення.

Для компактності прикладу на діаграмі, оберемо перелічення тематики іграшки - Subject.

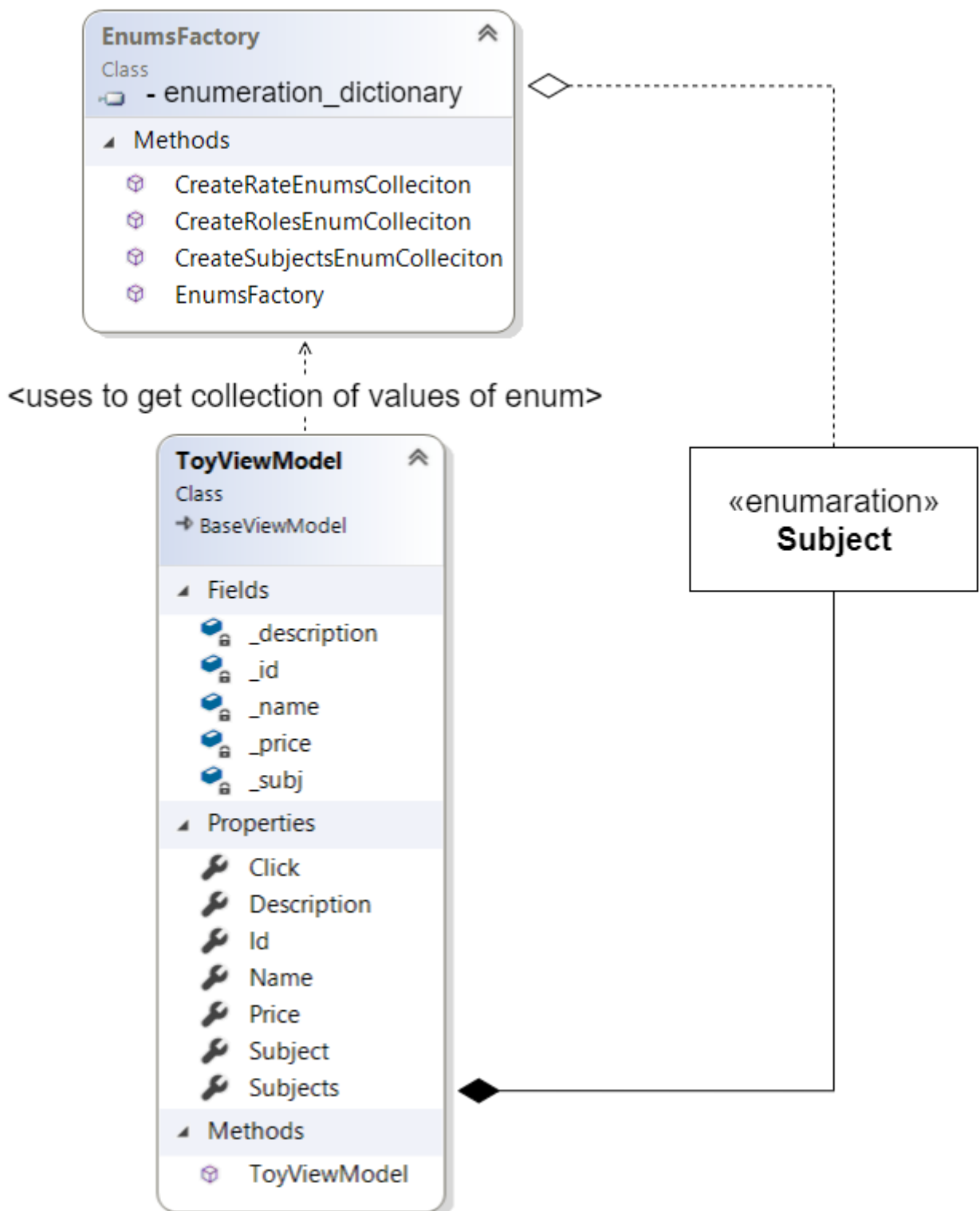


Рис.2.1.6 Діаграма частини класів, що беруть участь у реалізації  
Легковаговика

*Flyweight factory* - EnumsFactory, клас зі статичним полем-посиланням на словник, що виконує перетворення перелічень та зберігає отриманні значення в словник;

Flyweight - перелічення тематики іграшки;

Клієнт - код класу ToyViewModel.

Результат: економія оперативної пам'яті, заощадження процесорного часу.

## 7) Посередник

Визначення: поведінковий шаблон, що централізує взаємодію між компонентами, таким чином послаблюючи їхню зв'язність.

Проблема: список з іграшками знаходиться на одній сторінці, в той час як поля для додавання нової іграшки разом із кнопкою на виконання - на іншій. Виникає таке питання: як через сторінку, що характеризується класом CreateToyViewModel, додавати іграшки на сторінку, що характеризується класом ToyListViewModel. Іншими словами, як комунікувати між ViewModels? Можна за допомогою ін'єкції однієї viewmodel в іншу налагодити додавання нової іграшки в БД і оновлювати список ToyListViewModel напряму. Але таким чином утворимо тісну зв'язність. Ризикуємо порушити принцип SRP та утворити "спагеті-код".

Рішення: змусимо об'єкти спілкуватися через окремий об'єкт-посередник, який знає, кому потрібно перенаправити той або інший запит, які сервіси для виконання потрібні. Завдяки цьому компоненти системи залежатимуть тільки від посередника, а не від десятків інших компонентів. Таким чином задачею медіатора буде:

- надання єдиної колекції іграшок, яку споживатимуть ViewModels, що агрегують медіатор;
- надання методів маніпулювання цією колекцією;
- синхронізації колекції з тим, що знаходиться в БД;
- синхронізація єдиної колекції з відображувальною колекцією в ToyListViewModel;

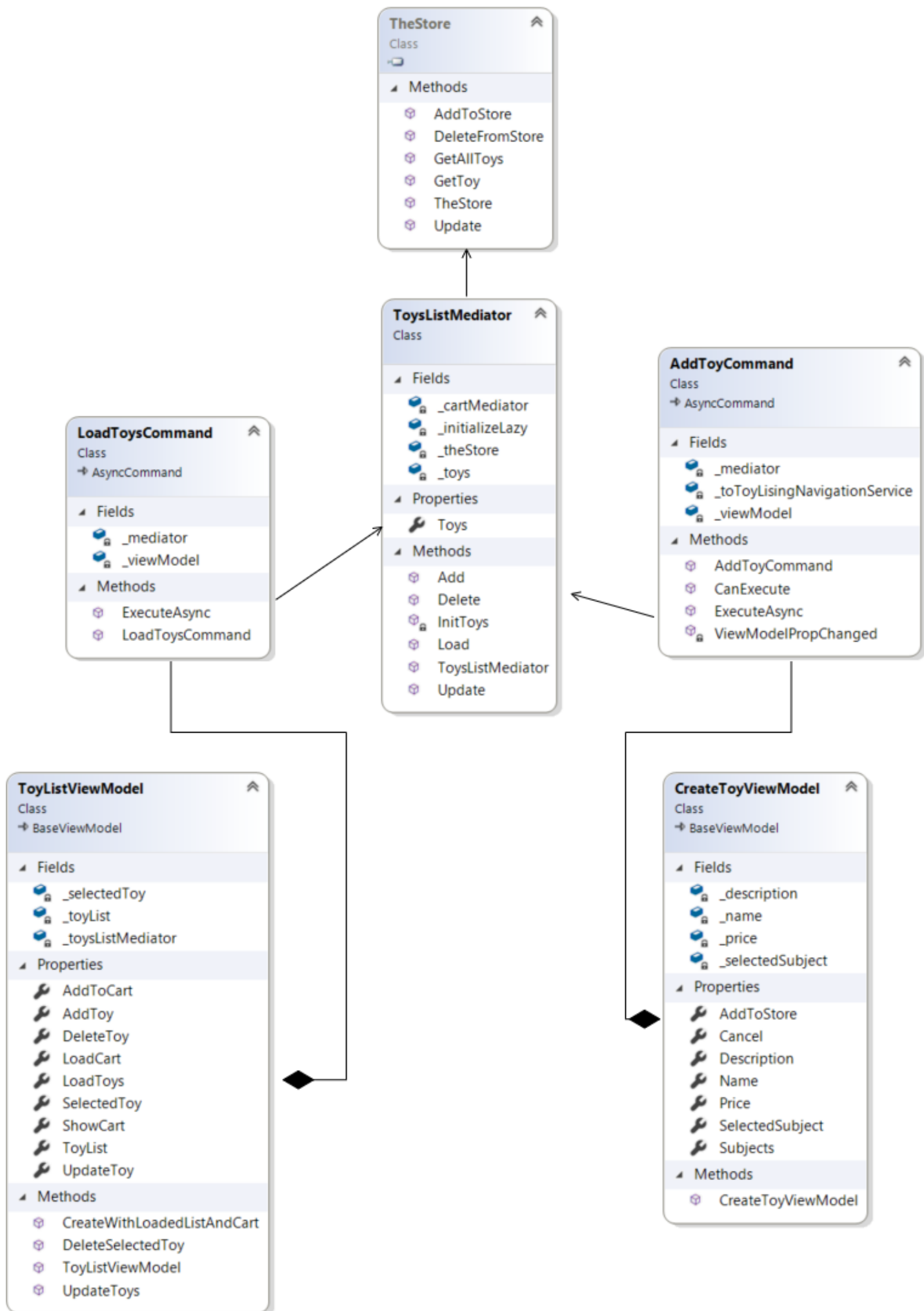


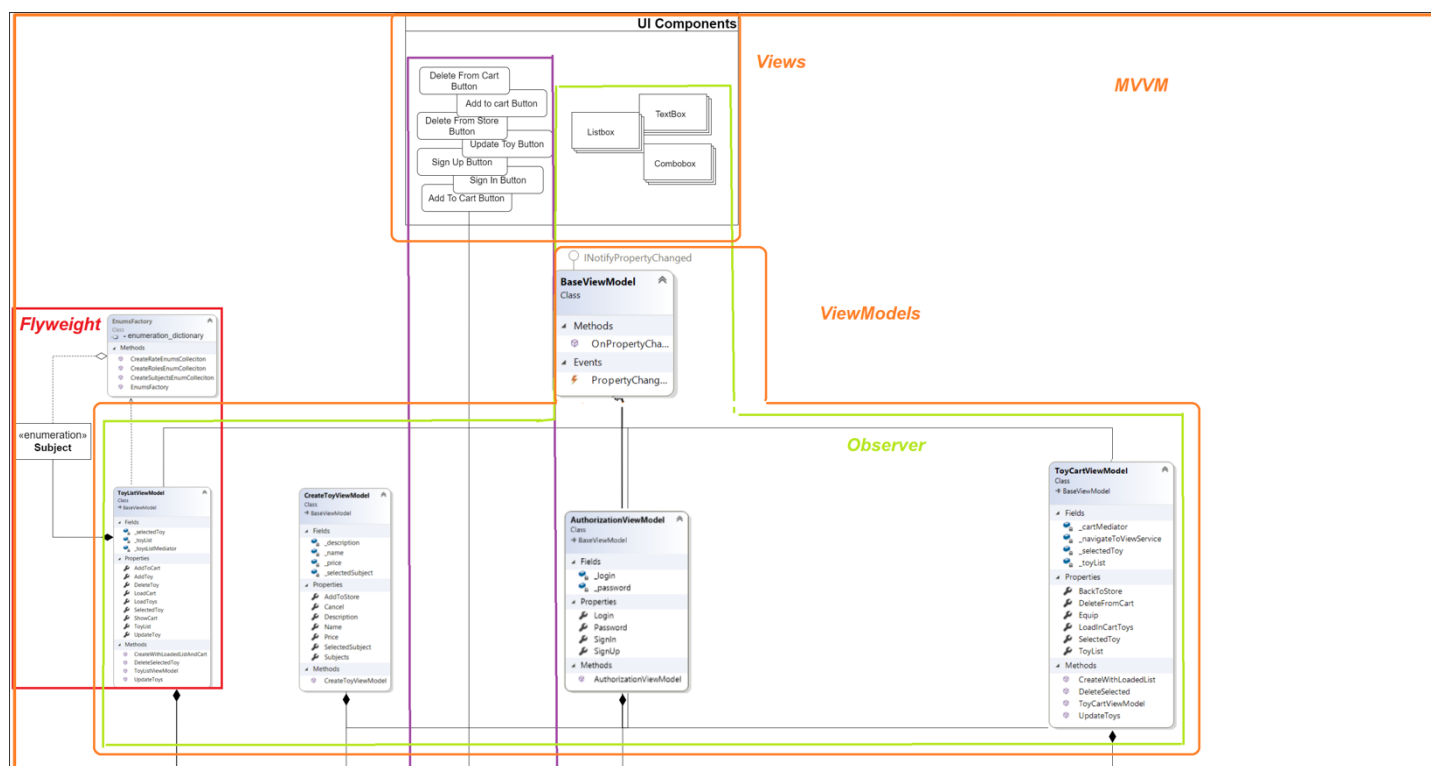
Рис.2.1.7 Діаграма частини класів, що беруть участь у реалізації Посередника

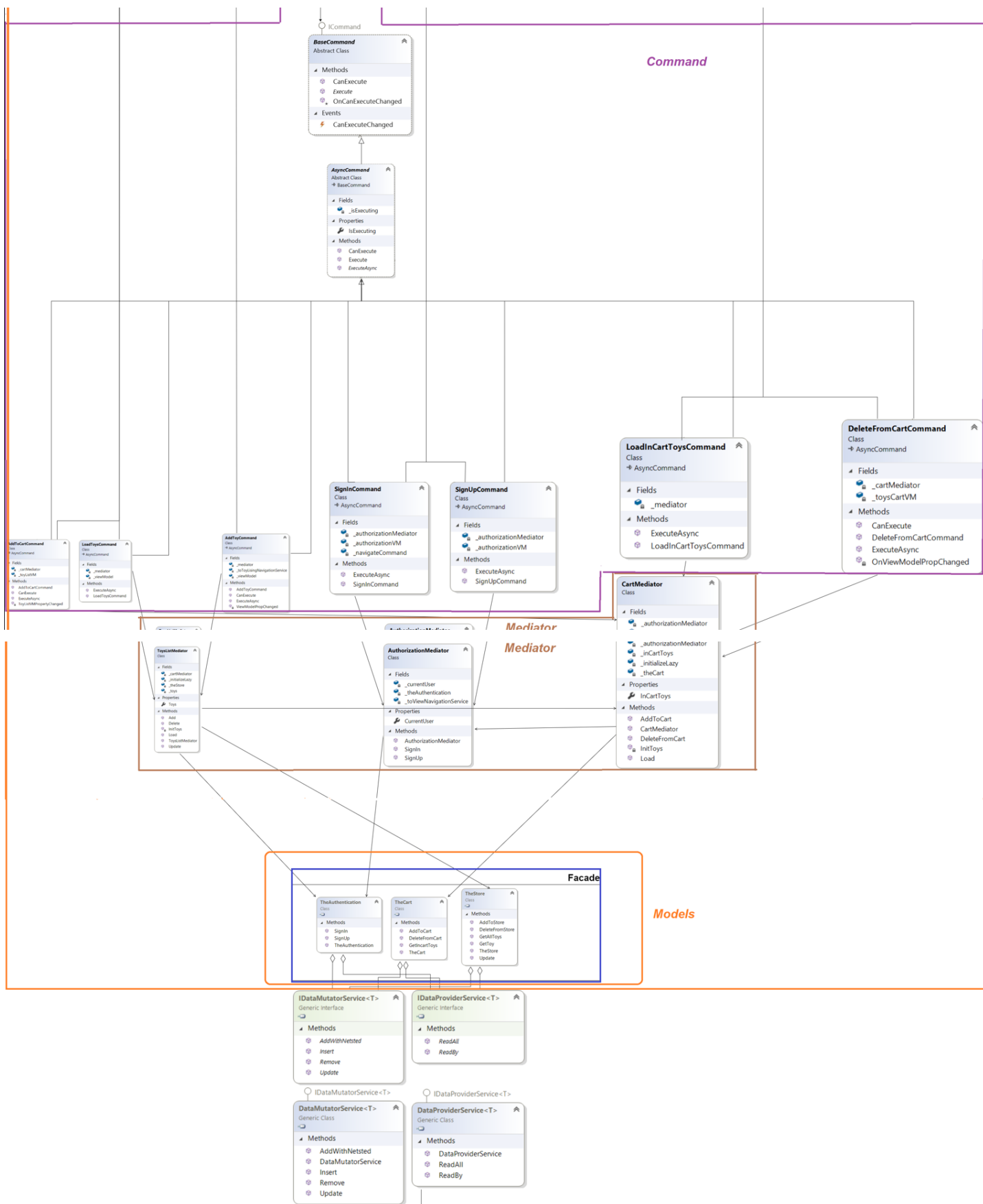
*Components* - компонентами можна вважати композицію ViewModel-Command, а саме: *ToyListViewModel-LoadToysCommand*, *CreateToyViewModel-AddToyCommand* - та клас-фасад бізнес-логіки: *TheStore*;

*Mediator* - посередником між компонентами є клас *ToyListMediator*.

Результат: спрощення взаємодії між компонентами, централізована взаємодія в одній ділянці коду, усунення залежності між компонентами, як наслідок можливість їх повторного використання.

## 2.2. Діаграма класів





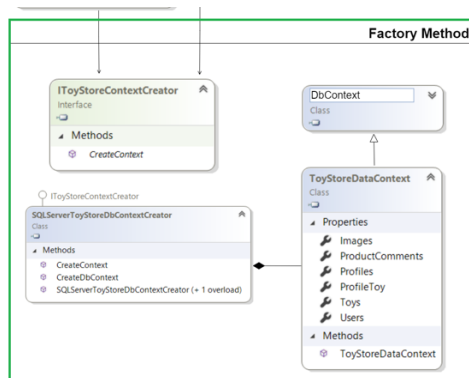


Рис.2.2.1 Загальна діаграма класів

## 2.3. Опис результатів роботи програми

### Authorization

Login

Password

Sign In

Sign Up


Рис.2.3.1 Вікно реєстрації та входу в систему

Користувач вводить логін та пароль натискає на кнопку реєстрації, далі на кнопку входу.

Code	Name	Subject
19003	Динозаври	horror
30002	Самокат Nixor Sports Pro	educational
30003	Набір-сюрприз	anime
30004	Лялковий набір LOL Surprise	anime
30005	Hot Wheels	military
30006	Конструктор Lego City	educational
30007	Ігровий набір Hot Wheels	educational
30008	Самокат біговел Scoot and Ride	educational

Delete From Store

Update Toy



Name

Лялковий набір LOL Surprise

Price

598

Subject

anime

Description

Лялковий набір Баскетболістів

Рис.2.3.2 Головне вікно системи


Після входу користувач здійснює автоматичний перехід на головну сторінку додатка. Надається можливість переглядати наявні в асортименті товари. Обравши одну іграшку, можна переглянути її детальну інформацію. Надається можливість видалення іграшки, з магазину. Змінивши значення поля, натискаючи кнопку “Оновити іграшку”, користувач змінює дані іграшки.



Code	Name	Subject
30003	Набір-сюрприз	anime
30004	Лялковий набір LOL Surprise	anime
30005	Hot Wheels	military
30006	Конструктор Lego City	educational
30007	Ігровий набір Hot Wheels	educational
30008	Самокат біговел Scoot and Ride	educational
30009	Лялька Леді Бар	military
30010	Динозаври лего Hello world	military

Delete From Store

Update Toy



Name

Price

Subject

Description

*Рис.2.3.3 Вікно після видалення товару та додавання іншого товару в кошик*

За допомогою кнопки “додати в кошик”, що визначається відповідною картинкою, кнопка на додавання цього товару не доступна. Можна переглянути кошик, користуючись відповідною кнопкою у верхній частині сторінки, що позначається картинкою кошику.

Toy Store

Code	Name	Subject
30005	Hot Wheels	military
30004	Лялковий набір LOL Surprise	anime

Delete from cart

Name

Лялковий набір LOL Surprise

Price

598

Subject

anime

Description

Лялковий набір Баскетболістів

Рис.2.3.4 Ілюстрація вмісту кошику

На сторінці з товарами у кошику можна переглянути додані товари. Надається можливість видалення товару з кошику. При цьому можливість його додавання на головній сторінці повертається.

Toy Store

Code	Name	Subject
------	------	---------

Delete from cart

Name

Price

Subject

Description

Рис.2.3.6 Пустий кошик

Create toy

Name

Price

0

Subject

horror

Description

Add to store

Cancel

Рис.2.3.7 Вікно створення нового товару

Create toy

Name

Конструктор LEGO Technic Jam Grave Digge

Price

123

Subject

educational

Description

поціновувачів механіки й конструювання.

Add to store

Cancel

Рис.2.3.8 Вікно з внесеними даними про новий товар

## ВИСНОВКИ

*Метою роботи* було розроблення програмного забезпечення магазину дитячих іграшок з використанням шаблонів проектування.

Для досягнення визначеної мети було виконано в повному обсязі такі завдання:

- абстрагування об'єктів предметної галузі;
- розробка структурної організації ПЗ за допомогою застосування основних принципів ООП та шаблонів проектування;
- визначення та опис функціональних характеристик програми;
- обґрунтування вибору шаблонів проектування, використаних для побудови програми;
- розроблення дизайну інтерфейсу користувача;
- виконано реалізацію програмного забезпечення відповідно до вимог технічного завдання;
- виконано тестування розробленої програми;
- оформлення документації з курсової роботи.

Розроблений застосунок вирішить на ряд наступних проблем як-от: наявність повної інформації про товар, актуальність цієї інформації (параметри іграшки, наявність в асортименті), помилки під час реалізації товарів за невірною ціною, крадіжки, швидкість продажів.

Для функціонування розробленої програми необхідна наявність на комп'ютері 2 ГБ оперативної пам'яті, від 800 МБ до 210 ГБ вільного місця на жорсткому диску, встановлений .Net 5.

Розроблене програмне забезпечення може буде використане магазинами іграшок для вирішення ряду проблем з метою заощадження та оптимізації бізнесу.

## **СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. Прийоми об'єктно орієнтованого проєктування. Паттерни проєктування / Еріх Гама, Річард Гелм, Ральф Джонсон, Джон Влісідес. – 1994. – 395 с.
2. Дизайн-патерни - просто як двері /Андрій будай. 2012. — 90с.
3. Refactoring Guru [Електронний ресурс]: [Веб-сайт]. - Режим доступу: <https://refactoring.guru/uk> (дата звернення 26.04.2022) - Назва з екрана.