Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім.»

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему

**Шаблони проектування в ООП. Страхова компанія**

Виконав студент

ІІ курсу групи КП-83

Шкурко Вадим Вячеславович

залікова книжка КП-8327

Керівник роботи

доц., к.т.н. Заболотня Т.М.

Оцінка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, підпис)

Київ-2020

**ЗМІСТ**

**ВСТУП**.........................................................................................................................3

1. **ОПИС СТРУКТУРНО-ЛОГІЧНОЇ ОРГАНІЗАЦІЇ ПРОГРАМИ**.........5
   1. Модульна організація програми............................................................5
   2. Функціональні характеристики.............................................................7
   3. Опис реалізованих класів.......................................................................8
2. **ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЕКТУВАННЯ**................................................................17
   1. Обґрунтування вибору та опис шаблонів проектування для реалізації програмного забезпечення..................................17
   2. Діаграма класів......................................................................................24
   3. Результати роботи програми................................................................25

**ВИСНОВКИ**..............................................................................................................27

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**......................................................28

**ВСТУП**

Дана курсова робота присвячена розробці програмного забезпечення для роботи страхової компанії за допомогою використання шаблонів проектування. Задача моделювання програмного забезпечення для проведення роботи страхової компанії є дуже актуальною, адже страхування – невід'ємна частина володіння майном, і для організації швидкої та надійної системи, необхідно розробити певні рішення, які представлять основні потоки даних об’єктів, з якими взаємодіють працівники компанії. Дана тематика вибрана для курсової роботи тому, що результати абстрагування об’єктів у даній спроектованій системі дозволяють застосувати всі вивчені методи та принципи об’єктно-орієнтованого програмування для створення програмного забезпечення, зокрема мати змогу правильно організувати код за допомогою шаблонів проектування.

**Об’єктом** курсової роботи є *процес роботи* *страхової компанії*.

**Метою** роботи є *розроблення програмного забезпечення* *для роботи страхової компанії* з використанням шаблонів проектування.

Для досягнення описаної мети необхідно виконати такі **пункти завдань**:

* Абстрагувати об’єкти предметної галузі;
* Розробити структурну організацію програмного забезпечення за допомогою використання основних принципів об’єктно-орієнтованого програмування та шаблонів проектування;
* Визначити та описати функціональні характеристики програми;
* Обґрунтувати вибір шаблонів проектування;
* Розробити консольний інтерфейс користувача;
* Виконати реалізацію програмного забезпечення відповідно до технічного завдання;
* Виконати тестування розробленої програми;
* Оформити документацію з курсової роботи.

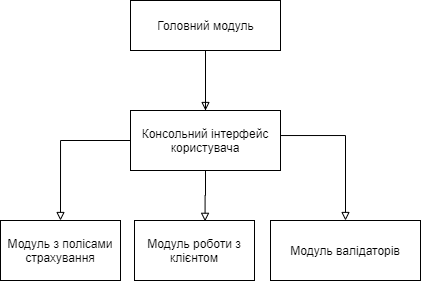
Розроблене програмне забезпечення складається з таких логічних частин: інтерфейсу користувача, користувачів(клієнтів) компанії, полісів страхування.

Використані шаблони проектування: **Міст, Шаблонний метод, Ланцюжок обов’язків, Стратегія, Замісник,** **Декоратор, Спостерігач.**

Розроблене програмне забезпечення може бути використане страховими компаніями.

Пояснювальна записка складається зі вступу, двох розділів, загальних висновків та списку використаних джерел (*три* найменування). Робота містить \_\_ рисунків. Загальний обсяг роботи – \_\_ друкованих сторінок.

1. **ОПИС СТРУКТУРНО-ЛОГІЧНОЇ СХЕМИ ПРОГРАМИ**
   1. **Модульна організація програми**



*Рис. 1.1.1. Модульна організація програми*

**Головний модуль системи** – містить в собі ініціалізацію консольного інтерфейсу користувача.

**Консольний інтерфейс користувача** – модуль консольного терміналу, який містить основну логіку роботи компанії та модулі з полісами страхування, роботою з клієнтом та валідатором.

**Модуль роботи з полісами страхування** – модуль, який відповідає за роботу з полісами(створення, зміна властивостей).

**Модуль для роботи з клієнтом** – модуль, який відповідає за функції роботи безпосередньо з сутністю користувача.

**Модуль з валідаторами –** модуль, який призначений перевіряти введені користувачем дані та, чи має користувач можливість зареєструвати поліс.

* 1. **Функціональні характеристики**

Робота з програмою розпочинається з *Main Menu* (головного меню). Користувачу може вибрати один з декількох його пунктів а саме він може зареєструвати нового клієнта, увійти в акаунт та переглянути інформацію про цього клієнта.

Якщо користувач вибере пункт додати клієнта він перейде до меню реєстрації де йому потрібно ввести своє ім’я, вік та, чи бажає він стати VIP-клієнтом.

Якщо користувач вибере пункт входу в акаунт, то залежно від того, чи користувач VIP, чи стандартний йому виведуть інформацію про користувача у різних форматах.

* 1. **Опис реалізованих класів**

1. **CUI –** консольний інтерфейс користувача. Для його реалізації використовуються такий патерн програмування, як **Ланцюжок обов’язків.** Він використовується для обробки запитів користувача. Прикладом класу-обробника є абстрактний **AbstractCommand** та його наслідувальні **ExitCommand, HomeCommand** та багато інших.

|  |
| --- |
| **CUI.cs | AbstractCommand | ExitCommand** |
| class Cui      {          private static Cui \_instance = null;          private HomeCommand homeCommand = new HomeCommand();          private HelpCommand helpCommand = new HelpCommand();          private ExitCommand exitCommand = new ExitCommand();          private LoginCommand loginCommand = new LoginCommand();          private NewPolicyCommand newPolicyCommand = new NewPolicyCommand();          private RegisterUser registerUser = new RegisterUser();          private AddPossessionCommand addPossessionCommand = new AddPossessionCommand();          private ShowCommand showCommand = new ShowCommand();          private AddDocument addDocument = new AddDocument();          private ChangePricesCommand changePricesCommand = new ChangePricesCommand();          private Cui()          {              homeCommand.SetNext(helpCommand).SetNext(exitCommand).SetNext(loginCommand).SetNext(newPolicyCommand).SetNext(registerUser).              SetNext(addPossessionCommand).SetNext(showCommand).SetNext(addDocument).SetNext(changePricesCommand);          }          public void Start()          {              Console.Clear();              bool isProgramActive = true;              Console.WriteLine(homeCommand.HandleRequest("-home"));              while (isProgramActive)              {                  string req = Console.ReadLine();                  object res = homeCommand.HandleRequest(req);                  if (res == null)                  {                      Console.WriteLine($"{req} is not an available command! Try something else");                  }                  else if (res.ToString().Equals("False"))                  {                      isProgramActive = false;                  }                  else                  {                      Console.WriteLine(res);                  }              }          }          public static Cui GetInstance()          {              if (\_instance == null)              {                  \_instance = new Cui();              }              return \_instance;          }      }  abstract class AbstractCommand : ICommand      {          private ICommand \_nextCommand;          public ICommand SetNext(ICommand command)          {              this.\_nextCommand = command;              return command;          }          public virtual object HandleRequest(string req)          {              if (this.\_nextCommand != null)              {                  return this.\_nextCommand.HandleRequest(req);              }              else              {                  return null;              }          }      }  class ExitCommand : AbstractCommand      {          public override object HandleRequest(string req)          {              if (req.Equals("-exit"))              {                  return false;              }              return base.HandleRequest(req);          }      } |

1. **User** —абстрактний клас для оголошення сутності, яка відображатиме користувача страхової компанії. Цей клас є імплементацією патерну «міст».

Разом ізкласом **User** патерну «міст» для реалізації використовуються класи: **StandartUser, VIPUser.**

|  |
| --- |
| **User | StandartUser | VIPUser** |
| abstract class User      {          public string Name;          public Dictionary<string, UserPossession> possessions = new Dictionary<string, UserPossession>();          public Dictionary<string, Document> documents = new Dictionary<string, Document>();          public int Age;          public User(string name, int age)          {              UserPossession life = new UserPossession("life", false);              UserPossession health = new UserPossession("health", false);              possessions.Add("life", life);              possessions.Add("health", health);              Name = name;              Age = age;          }          abstract public void ShowInfo();      }  class StandartUser : User      {          public StandartUser(string name, int age) : base(name, age) { }          public override void ShowInfo()          {              Console.WriteLine($"User {Name} owns:");              if (possessions != null && possessions.Count != 0)              {                  foreach (UserPossession possession in possessions.Values)                  {                      Console.WriteLine($"----{possession.Name}");                  }              }              else              {                  Console.WriteLine("-----------nothing-----------");              }              Console.WriteLine($"Also {Name} has given our company these documents:");              if (documents != null && documents.Count != 0)              {                  foreach (Document document in documents.Values)                  {                      Console.WriteLine($"----{document.DocName}");                  }              }              else              {                  Console.WriteLine("-----------nothing-----------");              }          }      }  class VIPUser : User      {          public VIPUser(string name, int age) : base(name, age) { }          public override void ShowInfo()          {              Console.WriteLine($"VIP User {Name} owns:");              if (possessions != null && possessions.Count != 0)              {                  foreach (UserPossession possession in possessions.Values)                  {                      Console.WriteLine($"----{possession.Name}");                  }              }              else              {                  Console.WriteLine("-----------nothing-----------");              }              Console.WriteLine($"Also {Name} has given our company these documents:");              if (documents != null && documents.Count != 0)              {                  foreach (Document document in documents.Values)                  {                      Console.WriteLine($"----{document.DocName}");                  }              }              else              {                  Console.WriteLine("-----------nothing-----------");              }          }      } |

1. **AbstractInsurance–** абстрактний клас, який визначає структуру полісу страхування. Також допомагає реалізувати такі патерни: **Шаблонний метод, Декоратор**. Від цього класу наслідуються **LifeInsurance, HealthInsurance, CarInsurance, HouseInsurance**

|  |
| --- |
| **AbstractInsurance | LifeInsurance** |
| abstract class AbstractInsurance : Document      {          protected string InsuredItem;          public float Price = 1;          public float MaxPayOut = 1;          abstract protected void Calculate();          private bool IsUserVIP = false;          public AbstractInsurance(string clientName, string docName, string insuredItem, bool isVIPuser) : base(docName, clientName)          {              OwnerName = clientName;              InsuredItem = insuredItem;              IsUserVIP = isVIPuser;              if (isVIPuser)              {                  MaxPayOut = 2;                  Price = (float)1.5;              }          }          public override void FormDocument()          {              Console.WriteLine($"{OwnerName} has insured {InsuredItem} using {DocName}. It cost him {Price}uah and max sum of payout reaches {MaxPayOut}uah. Insurance period started at {Date}");          }          public void CreatePolicy()          {              Calculate();              RegisterPolicy();              FormDocument();          }          public void UpdatePricePolitics(float newPrice, float newMaxPayout)          {              this.Price = newPrice;              this.MaxPayOut = newMaxPayout;              if (IsUserVIP)              {                  Price \*= (float)1.5;                  MaxPayOut \*= 2;              }              CreatePolicy();          }      }  class LifeInsurance : AbstractInsurance      {          public LifeInsurance(string clientName, string docName, string insuredItem, bool isVIPuser) : base(clientName, docName, insuredItem, isVIPuser) { }          protected override void Calculate()          {              DocName = "Life insurance";              Price \*= 1200;              MaxPayOut \*= 200000;          }      } |

1. **AuthSystem** – клас, реалізуючий валідацію реєстрації, використання бази даних. Клас реалізує шаблон **Замісник.**

|  |
| --- |
| **AuthSystem** |
| class AuthSystem      {          public static Dictionary<string, User> UsersDB = new Dictionary<string, User>();          public static User СurrentUser = null;          public static bool RegisterUser(User user)          {              if (!UsersDB.ContainsKey(user.Name))              {                  UsersDB.Add(user.Name, user);                  return true;              }              return false;          }          public static bool CheckIfUserHasPermission(User user)          {              if (user.Age >= 18)              {                  return true;              }              return false;          }          public static User AuthorizeUser(string userName)          {              СurrentUser = UsersDB[userName];              return СurrentUser;          }      } |

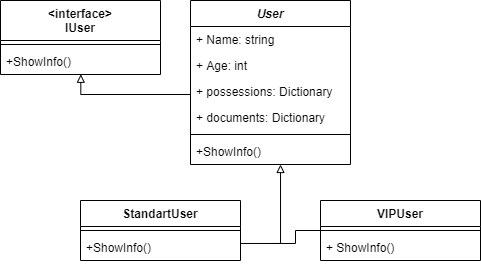
1. **NewPolicyCommand —** клас-обробник запитів користувача, реалізуючий створення полісу страхування. Також це клас реалізації шаблону **Стратегія**.

|  |
| --- |
| **NewPolicyCommand** |
| class NewPolicyCommand : AbstractCommand      {          public override object HandleRequest(string req)          {              if (req.Equals("-newpolicy"))              {                  if (AuthSystem.СurrentUser == null)                  {                      Console.WriteLine("No user is logged in!");                  }                  else                  {                      Console.WriteLine("Enter the subject that you want to insure");                      string subject = Console.ReadLine();                      User user = AuthSystem.СurrentUser;                      if (AuthSystem.СurrentUser.possessions != null && AuthSystem.СurrentUser.possessions.ContainsKey(subject))                      {                          if (PolicyValidator.CheckIfPolicyAvailable(subject, user))                          {                              AbstractInsurance insurance = null;                              bool isUserVIP = user.ToString().Equals("oop\_coursework.VIPUser");                              if (subject.Equals("car"))                              {                                  insurance = new CarInsurance(user.Name, "Car insurance", subject, isUserVIP);                              }                              else if (subject.Equals("life"))                              {                                  insurance = new LifeInsurance(user.Name, "Life insurance", subject, isUserVIP);                              }                              else if (subject.Equals("health"))                              {                                  insurance = new HealthInsurance(user.Name, "health", subject, isUserVIP);                              }                              else if (subject.Equals("house"))                              {                                  insurance = new HouseInsurance(user.Name, "house insurance", subject, isUserVIP);                              }                              if (insurance != null)                              {                                  insurance.CreatePolicy();                                  Company.registeredPolicies.Add(insurance);                              }                              else                              {                                  Console.WriteLine($"{subject} is not supposed to be insured");                              }                          }                      }                      else                      {                          Console.WriteLine($"{user.Name} does not have {subject}");                      }                  }                  return $"";              }              return base.HandleRequest(req);          }      } |
|  |

1. **Company —** клас, реалізуючий можливість зміни тарифів страхування та миттєвого редагування вже існуючих полісів. Реалізує шаблон **Спостерігач**.

|  |
| --- |
| **Company** |
| class Company      {          public static List<AbstractInsurance> registeredPolicies = new List<AbstractInsurance>();          public static void NotifyCreatedPolicies(float newPrice, float newMaxPayOut)          {              foreach (AbstractInsurance insurance in registeredPolicies)              {                  insurance.UpdatePricePolitics(newPrice, newMaxPayOut);}}} |

1. **ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЕКТУВАННЯ**
   1. **Обґрунтування вибору та опис шаблонів проектування для реалізації програмного забезпечення автомату**
2. **Міст**

****

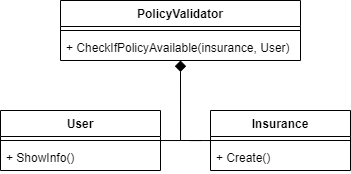
*Рис. 2.1.1. UML-діаграма шаблону* ***Міст***

*Структурний шаблон*. Це структурний шаблон проектування, що дозволяє розділяти абстракцію і реалізацію таким чином, щоб вони могли змінюватися незалежно. Шаблон bridge (від англ. – «міст») використовує інкапсуляцію, агрегування та успадкування для того, щоб розділити відповідальність між класами.

*Структура*.Реалізований для класу **User**. Цей клас містить в собі інтерфейс для якого є дві реалізації.

*Обґрунтування використання даного шаблону*. У користувача є два види акаунту: стандартний та VIP. VIP дає можливість платити за страхування в 1.5 рази більше, але максимальна сума виплати збільшиться у 2 рази.

1. **Заступник**

**

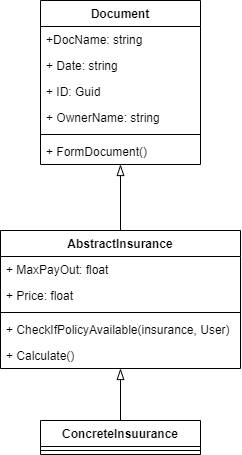
*Рис. 2.1.2. UML-діаграма шаблону “***Заступник***”*

*Структурний шаблон*. Шаблон який забезпечує створення заступника об’єкта для контролю доступу до останнього через перехоплення всіх викликів.

*Структура*. **PolicyValidator –** клас-замісник, через який дається доступ класам **Insurance**.

*Обґрунтування використання даного шаблону*. Реалізований для того, щоб перевірити, чи можна зареєструвати поліс страхування. Перевіряється наявність документів користувача, вік для деяких об’єктів страхування та наявність об’єкту страхування у користувача.

1. **Декоратор**

****

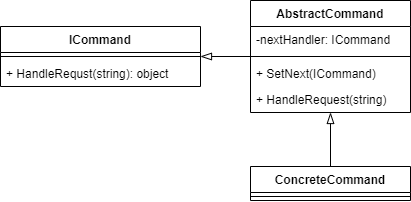
*Рис. 2.1.3. UML-діаграма шаблону “Декоратор”*

*Структурний шаблон*. Визначає інтерфейс для створення об'єктів, але дозволяє підкласам змінювати тип створюваних об'єктів.

*Структура*. **MarkDescription** – головний інтерфейс даного шаблону. Має у середині один-єдиний метод: Description(). Цей інтерфейс реалізують чотири інших класів, представлених на скріншоті зверху.

*Обґрунтування використання даного шаблону*. За допомогою даного шаблона можна організувати гнучку реалізацію різновидів певного класу, і в даній програмі така реалізація потребувалася для опису отриманої учасником оцінки.

1. **Ланцюжок обов’язків**

****

*Рис. 2.1.4. UML-діаграма шаблону “****Ланцюжок обов’язків****”*

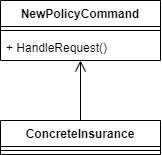
*Поведінковий шаблон*. поведінковий патерн проектування, що дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком.

*Структура*. **CUI** використувоє класи-обробники **AbstractCommand**

*Обґрунтування використання даного шаблону*. Інтерфейс користувача не повинен на пряму взаємодіяти з роботами учасників тому в дію вступає посередник який ослабить зв’язність системи, позбавляючи об'єкти від необхідності явно посилатися один на одного і дозволяючи тим самим незалежно змінювати взаємодії між ними;

.

1. **Стратегія**

****

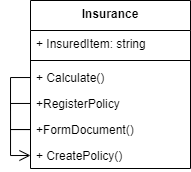
*Рис. 2.1.5. UML-діаграма шаблону “***Стратегія***”*

*Поведінковий шаблон*. поведінковий патерн проектування, який визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна заміняти один на інший прямо під час виконання програми.

*Структура*. **NewPolicyCommand** – клас-обробник, в якому користувач за допомогою консольного інтерфейсу може обрати, що саме він хоче застрахувати.

*Обґрунтування використання даного шаблону*.Клієнт страхової компанії хоче щось застрахувати і має обрати, що саме страхувати.

1. **Шаблонний метод**

****

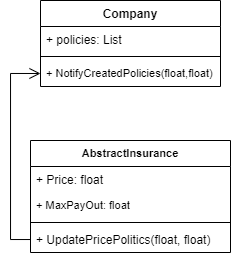
*Рис. 2.1.6. UML-діаграма шаблону “****Шаблонний метод****”*

*Поведінковий шаблон*. поведінковий патерн проектування, який визначає кістяк алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи. Патерн дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури.

*Структура*. **AbstractInsurance —** абстрактний клас, реалізує шаблонне виконання страхування.

*Обґрунтування використання даного шаблону*.Для шаблонного виконання страхування доцільно використати шаблонний метод.

1. **Спостерігач**

****

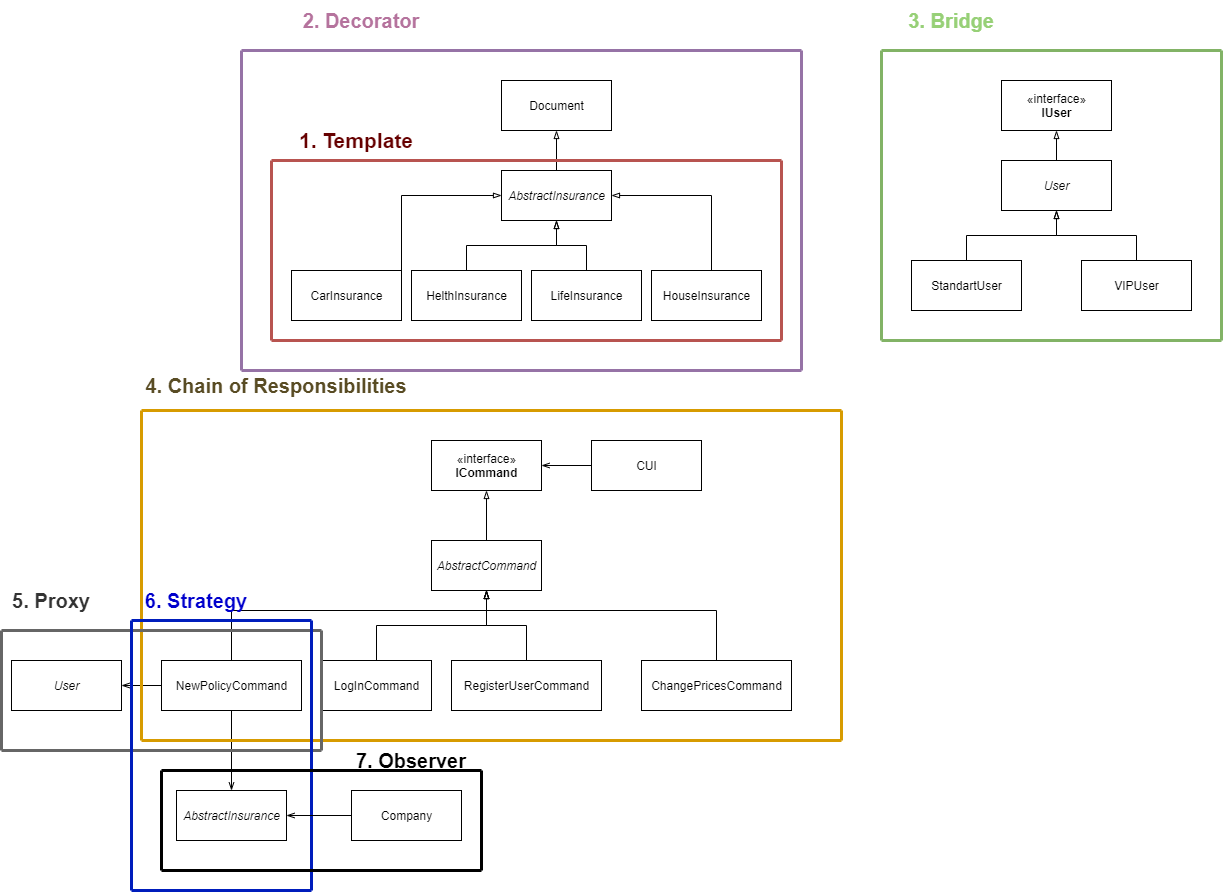
*Рис. 2.1.7. UML-діаграма шаблону “***Спостерігач***”*

*Поведінковий шаблон*. поведінковий патерн, який дозволяє об’єктам повідомляти інші об’єкти про зміни свого стану.

*Структура*. **Company –** клас, який містить всі зареєстровані поліси страхування.

*Обґрунтування використання даного шаблону*. Якщо компанія хоче змінити цінову політику, клас може пройтись по всім зареєстрованим полісам та змінити їх властивості.

* 1. **Діаграма класів**



*Рис. 2.2.1. Діаграма класів програми*

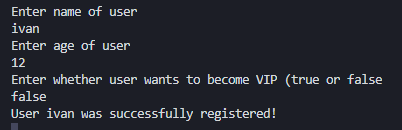
* 1. **Результати роботи програми**

Початкове меню де користувач продовжити роботу з програмою або дізнатися про доступні команди:



*Рис. 2.3.1. Головне меню*

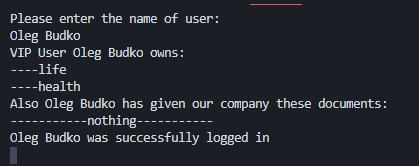
Якщо користувач введе **-register** його направить в меню реєстрації, де йому буде потрібно ввести своє ім’я, вік та, чи хоче він обслуговуватися, як VIP.



*Рис 2.3.2. Меню реєстрації*

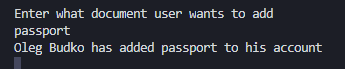
Після проходження користувачем реєстрації користувач може спробувати увійти в свій акаунт.

Якщо користувач введе **–login,** користувач перейде до меню авторизації, де має ввести своє ім’я. Система його авторизує. У випадку успішної авторизації виведеться інформація про цього користувача



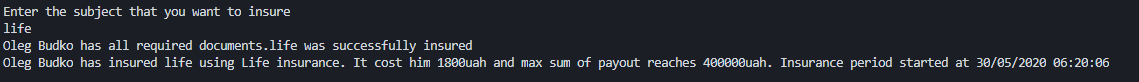
*Рис 2.3.4. Результат авторизації*

При вводі **–adddocument** користувач може додати документ до його акаунту. Наприклад, для страхування в компанії обов’язково мають бути такі документи: паспорт та ідентифікаційний код.



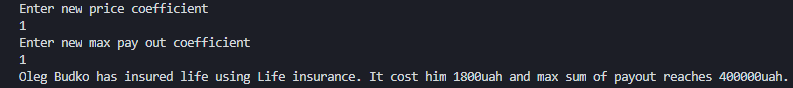
*Рис 2.3.4. Додавання документів до акаунту користувача*

При введенні **–newpolicy**, користувач переходить до меню страхування. Там він обирає, що хоче страхувати і, якщо компанія має всі необхідні документи, то створюється новий поліс.



*Рис 2.3.5. Страхування майна*

При введенні **–changeprices** користувач може змінити тарифи всієї компанії, також вже створені поліси перейдуть на нові тарифи.



*Рис 2.3.6. Сповіщення про змінуі тарифу*

**ВИСНОВКИ**

Метою даної курсової роботи було розроблення програмного забезпечення для проведення олімпіад з використанням шаблонів проектування. Підставою для розроблення стало завдання на виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування» студентами ІІ курсу кафедри ПЗКС НТУУ «КПІ».

Для досягнення поставленої мети у повному обсязі виконано завдання, визначені у аркуші завдання на курсову роботу; розроблено графічні матеріали; реалізовано всі вимоги до програмного продукту, наведені у технічному завданні; створено відповідну документацію. Розроблене програмне забезпечення дозволяє користувачу реєструвати нових клієнтів, страхувати майно клієнтів та змінювати тарифи компанії.

Програму створено на основі використання шаблонів проектування. Зокрема, до структури програмного забезпечення входить реалізація семи шаблонів, які належать до різних груп.

Для розроблення програмного забезпечення була використана мова програмування C#.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. Язык шаблонов. Города. Здания. Строительство. / Кристофер Вольфганг Александер. – 1977. – 1096 с.
2. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес. – 1994. – 395 с.
3. Руководство Microsoft по проектированию архитектуры приложений. / С. Сомасегар, Скотт Гатри, Дэвид Хилл. – 2009. – 529 с.