МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

## КУРСОВА РОБОТА

з дисципліни «Об’єктно-орієнтоване програмування»   
на тему

## Веб застосунок дослідження мов програмування

Виконав студент

ІІ курсу групи КП-11   
Корнійчук Микола Платонович

залікова книжка КП-1109

Керівник роботи

доцент, к.т.н. Заболотня Т.М. Оцінка

(дата, підпис) 

КИЇВ 2019

## ЗМІСТ

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ** 3

## ВСТУП 4

## СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ ВЕБ ЗАСТОСУНКУ 6

* 1. Модульна організація програми 6
  2. Функціональні характеристики 8
  3. Опис реалізованих класів 10

## ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЕКТУВАННЯ 28

* 1. Обґрунтування вибору та опис шаблонів проектування для

програмної реалізації мікрофреймворку для web-додатків 28

* 1. Діаграма класів 45
  2. Опис результатів роботи програми 47

## ВИСНОВКИ 52

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 53

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

| НТТP - | Hyper Text Transfer Protocol (протокол передавання гіпертекстових документів); |
| --- | --- |
| Web - | (англ. World Wide Web) Глобальний інформаційний простір, заснований на фізичній інфраструктурі Інтернету і протоколі передачі даних HTTP |
| URI - | (англ. Uniform Resource Identifier) Уніфікований ідентифікатор ресурсів компактний рядок літер, який однозначно ідентифікує окремий абстрактний чи фізичний ресурс. |
| URL - | (англ. Uniform Resource Locator) Уніфікований локатор ресурсів або адреса ресурсу. Стандартизована адреса певного ресурсу (такого як документ, чи зображення) в інтернеті (чи деінде) |
| IP - | (від англ. Internet Protocol) Протокол мережевого рівня для передавання датаграм між мережами. |
|  |  |

## ВСТУП

Сучасний світ неможливо уявити без інформаційних технологій, які динамічно розвиваються з кожним днем. Комп'ютерні програми стали необхідною частиною повсякденного життя багатьох людей. Ядром більшості інформаційних систем є програмний код, написаний на одній або декількох мовах програмування. Мови програмування - це засоби взаємодії людини з комп'ютером, інструменти, що дозволяють перетворити ідеї у реальні програми і системи. Вони постійно розвиваються, змінюються, покращуються, з'являються нові.

Цей курсовий проект присвячений дослідженню еволюції мов програмування протягом часу. Ми зануримося в історію створення та розвитку ключових мов програмування, їх вплив на сучасні технології та визначимо основні тенденції, що відображають напрямки їх майбутнього розвитку. Дослідження проводитиметься за допомогою аналізу великого обсягу даних з різних джерел, включаючи наукові статті, статистичні дані, інтерв'ю з експертами в галузі та інші ресурси.

Особлива увага буде приділена створенню веб-сайту, що відображатиме результати цього дослідження. Сайт буде містити візуалізацію даних про еволюцію мов програмування, порівняння їх використання, популярності та впливу на індустрію. Цей інтерактивний ресурс буде корисним для всіх, хто цікавиться історією мов програмування, їх розвитком і впливом на сучасний світ.

Очікується, що ця робота допоможе краще зрозуміти процеси, що відбуваються в світі мов програмування, а також сприятиме поглибленню знань у цій галузі. Результати цього дослідження можуть бути використані як освітній ресурс для студентів, викладачів, науковців та професіоналів у галузі комп'ютерних наук.

*Об’єктом* дослідження є мови програмування протяго часу.

*Метою роботи* є розроблення програмного забезпечення для візуалізації даних про мови програмування.

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 сайту та його логіку
* виконати реалізацію програмного забезпечення відповідно до вимог технічного завдання;
* виконати тестування розробленої програми;
* оформити документацію з курсової роботи.

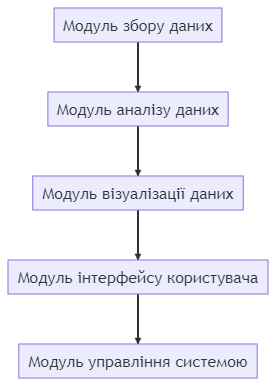
Реалізовані шаблони проектування: Адаптер, Спостерігач, Фабрика, Знімок, Будівельник та Одинак.

До функціональних можливостей програми належать: перегляд графіка на певному часовому проміжками і зображення.

Розроблене програмне забезпечення може бути використане для дослідження мов програмування на часових зрізах.

Пояснювальна записка складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел (6-ти найменувань, з них 2 – іноземною мовою). Робота містить 21 рисунок. Загальний обсяг роботи – 53 друкованих сторінки, з них 46 сторінок основного тексту та 1 сторінка списку використаних джерел.

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



*Рис. 1.1.1. Модульна організація програми*

Для забезпечення гнучкості, масштабованості та ефективного управління програмним кодом веб-сайту, програма створена за принципами модульної архітектури. Модульна організація програми дозволяє розбити загальну задачу на підзадачі, які вирішуються окремо, що спрощує розробку, тестування та підтримку системи.

Програма веб-сайту складається з наступних основних модулів:

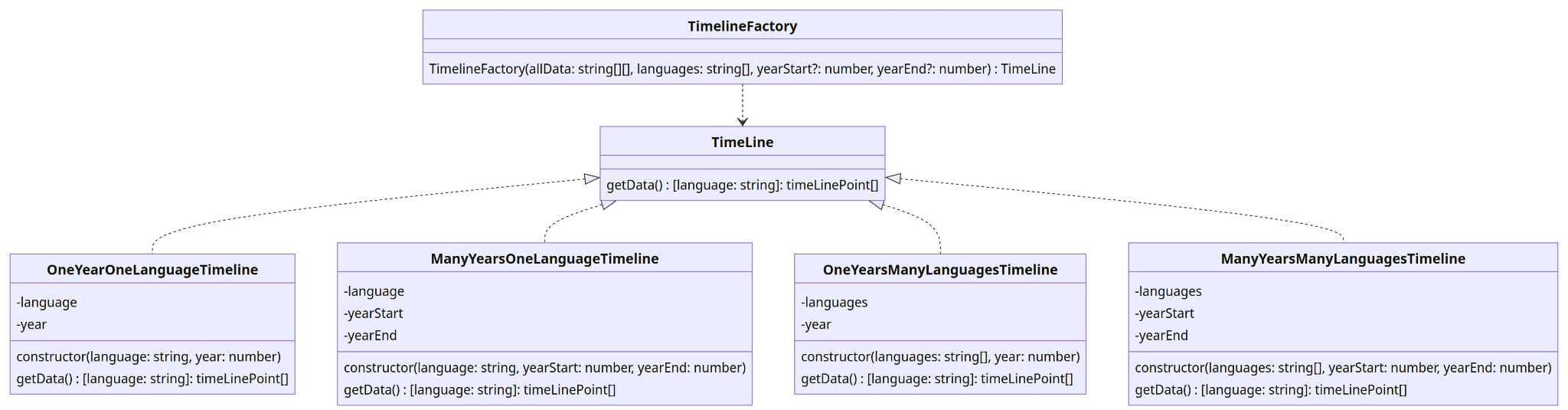
* Модуль збору даних. Цей модуль відповідає за збір даних з різних джерел про мови програмування. До його задач входить отримання, обробка, нормалізація та збереження даних у відповідних структурах даних.
* Модуль аналізу даних. Цей модуль використовує зібрані дані для вирішення конкретних аналітичних задач, пов'язаних з дослідженням еволюції мов програмування.
* Модуль візуалізації даних. Цей модуль відповідає за створення інтерактивних діаграм, графіків та інших візуальних елементів, які допомагають користувачам краще зрозуміти результати аналізу.
* Модуль інтерфейсу користувача. Цей модуль відповідає за взаємодію між користувачем та системою. Він включає розробку веб-сторінок, меню, форм та інших елементів інтерфейсу.
* Модуль управління системою. Цей модуль відповідає за загальне управління програмою, включаючи керування роботою інших модулів, виконання задач по плануванню та координації, обробку помилок, логування та моніторинг системи.

Кожен модуль складається з набору функцій та методів, які реалізують конкретні задачі. Модулі взаємодіють між собою за допомогою визначених інтерфейсів, що дозволяє забезпечити слабку зв'язаність компонентів системи, а також забезпечує можливість їх незалежного оновлення та модифікації.

* 1. **Функціональні характеристики**
* Аналіз даних: Програма проводить аналітичний обробку зібраних даних. Це може включати порівняння популярності різних мов програмування, виявлення трендів та змін використання мов протягом часу, вивчення впливу різних факторів на розвиток та використання мов програмування.
* Візуалізація даних: Програма створює інтерактивні графіки, діаграми та інші візуальні елементи, які демонструють результати аналізу даних. Візуалізації допомагають користувачам легше і швидше зрозуміти отримані результати.
* Інтерактивність: Користувачі можуть інтерактивно працювати з візуалізованими даними, вибираючи конкретні періоди часу, мови програмування або інші параметри для аналізу. Також користувачі можуть задавати власні запити на аналіз.
* Управління системою: Програма має вбудовані механізми для моніторингу стану системи, виявлення та обробки помилок, логування дій, планування та виконання задач.

1. **ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЕКТУВАННЯ** 
   1. **Обґрунтування вибору та опис шаблонів проектування** **для програмної реалізації**
2. **Фабрика**

TimelineFactory генерує різні типи часових шкал на основі вхідних параметрів. Діаграма класів представлена на рис. 2.1.1.



*Рис. 2.1.1. Діаграма класів, які входять до шаблону «Фабрика»*

*Учасники шаблону:*

* timeLinePoint: це оголошення типу, яке представляє точку на часовій шкалі. Це масив з наступною структурою: [number, string, number, number].
* mainData: це змінна, що містить масив елементів timeLinePoint. Спочатку вона не визначена і заповнюється всередині функції TimelineFactory.
* TimelineFactory(allData: string[][], languages: string[], yearStart?: number, yearEnd?: number): це фабрична функція. Вона приймає чотири параметри:
* allData - це масив масивів, кожен з яких містить чотири рядкові елементи, які будуть перетворені на елементи timeLinePoint і збережені в mainData.
* languages - це масив рядків, які представляють мови програмування для врахування на часовій шкалі.
* yearStart - це необов'язкове число, яке представляє початковий рік для часової шкали.
* yearEnd - це необов'язкове число, яке представляє кінцевий рік для часової шкали.
* Залежно від довжини масиву languages і різниці між yearStart і yearEnd, ця функція створює екземпляр одного з чотирьох різних класів: OneYearOneLanguageTimeline, ManyYearsOneLanguageTimeline, OneYearsManyLanguagesTimeline або ManyYearsManyLanguagesTimeline.
* TimeLine: це інтерфейс, який реалізують усі класи часових шкал. Він має один метод, getData(), який повертає об'єкт, де ключі - це мови, а значення - масиви елементів timeLinePoint.

*Класи, представлені в коді, функціонують наступним чином::*

* OneYearOneLanguageTimeline: Цей клас представляє часову шкалу для однієї мови програмування за один рік. Він має властивості "мова" (language) та "рік" (year). У його методі getData() він фільтрує основні дані (mainData) за відповідним роком та мовою​​.
* ManyYearsOneLanguageTimeline: Цей клас представляє часову шкалу для однієї мови програмування за багато років. Він має властивості "мова" (language), "початковий рік" (yearStart) та "кінцевий рік" (yearEnd). У його методі getData() він фільтрує основні дані (mainData) за відповідною мовою та діапазоном років​​.
* OneYearsManyLanguagesTimeline: Цей клас представляє часову шкалу для багатьох мов програмування за один рік. Він має властивості "мови" (languages) та "рік" (year). У його методі getData() він фільтрує основні дані (mainData) за відповідним роком та набором мов, а потім групує дані за мовами​​.
* ManyYearsManyLanguagesTimeline: Цей клас представляє часову шкалу для багатьох мов програмування за багато років. Він має властивості "мови" (languages), "початковий рік" (yearStart) та "кінцевий рік" (yearEnd). У його методі getData() він фільтрує основні дані (mainData) за відповідним діапазоном років та набором мов, а потім групує дані за мовами​.

*Обґрунтування використання шаблону:*

Шаблон проектування, який тут використовується, відноситься до відомого шаблону "Фабрика" (Factory). Фабричний шаблон проектування використовується, коли у нас є суперклас з декількома підкласами і на основі вхідних даних нам потрібно повернути об'єкт одного з підкласів.

Ідея в тому, що ми маємо різні типи часових шкал (один рік, одна мова; багато років, одна мова; один рік, багато мов; багато років, багато мов), і замість створення цих об'єктів напряму, ми використовуємо фабричний метод. Цей метод приймає вхідні дані (мови і роки) і вирішує, який тип часової шкали створити.

Основні переваги використання цього шаблону тут:

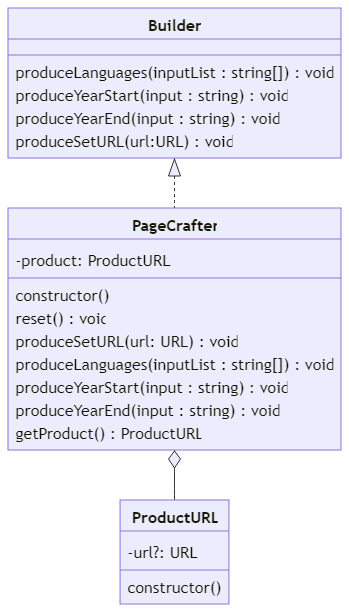
* Гнучкість: Фабричний метод може легко впоратися з новими типами часових шкал, якщо вони будуть додані в майбутньому. Нам просто потрібно додати новий підклас і оновити фабричний метод, щоб він міг створювати новий тип часової шкали.
* Інкапсуляція логіки створення: Логіка визначення, який тип часової шкали створити, інкапсульована в самому фабричному методі. Це робить основний код більш чистим і легшим для розуміння.
* Поліморфізм: Фабричний метод повертає об'єкт суперкласу (в даному випадку TimeLine), тому ми можемо використовувати поліморфізм. Ми можемо взаємодіяти з об'єктами різних типів через їх загальний інтерфейс.

1. **Будівельник**

PageCrafter є реалізацією патерну "Будівельник" (Builder), який використовується для створення складних об'єктів, які потребують багато кроків для їх конфігурації. В даному випадку, PageCrafter використовується для створення URL. Діаграма класів представлена на рис. 2.1.2.

*Учасники шаблону:*

* Builder (Будівельник): Інтерфейс, який визначає кроки для створення продукту.
* PageCrafter: Конкретний будівельник, який реалізує кроки, визначені в Builder.
* ProductURL: Продукт, який створюється за допомогою PageCrafter.

**

*Рис. 2.1.2. Діаграма класів, які входять до шаблону «Будівельник»*

*Результати використання конструкції:*

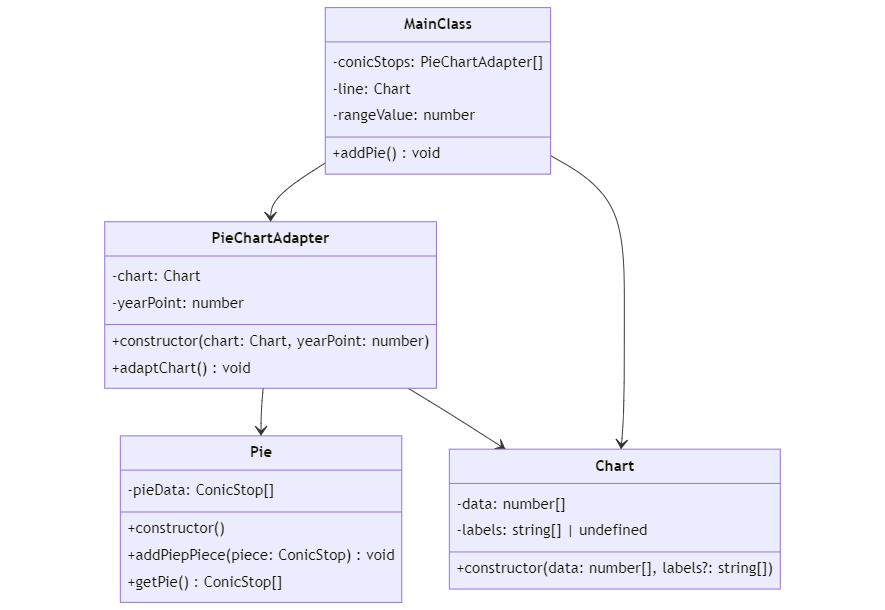
* Ізоляція складного процесу створення об'єкта від його основного представлення. Це означає, що той самий процес створення може бути використаний для створення різних представлень.
* Можливість контролювати процес створення об'єкта, використовуючи специфічні методи PageCrafter.
* Можливість створення різних продуктів, використовуючи той самий процес.

*Обґрунтування використання шаблону:*

Патерн "Будівельник" використовується, коли процес створення об'єкта є складним і потребує багато кроків. В даному випадку, створення URL вимагає встановлення різних параметрів (languages, yearStart, yearEnd), що робить цей процес ідеальним кандидатом для використання патерну "Будівельник". Крім того, цей патерн дозволяє ізолювати процес створення від представлення продукту, що забезпечує більшу гнучкість і повторне використання коду.

1. **Адаптер**

PieChartAdapter є реалізацією патерну "Адаптер" (Adapter), який використовується для перетворення інтерфейсу одного класу на інтерфейс, який очікується клієнтом. В даному випадку, PieChartAdapter адаптує дані з діаграми ChartJS до формату, який може бути використаний для створення кругової діаграми.



*Рис. 2.1.3. Діаграма класів, які входять до шаблону «Адаптер»*

*Учасники шаблону:*

* Pie: Цільовий інтерфейс, який очікується клієнтом.
* PieChartAdapter: Адаптер, який перетворює інтерфейс ChartJS на інтерфейс Pie.

*Результати використання конструкції:*

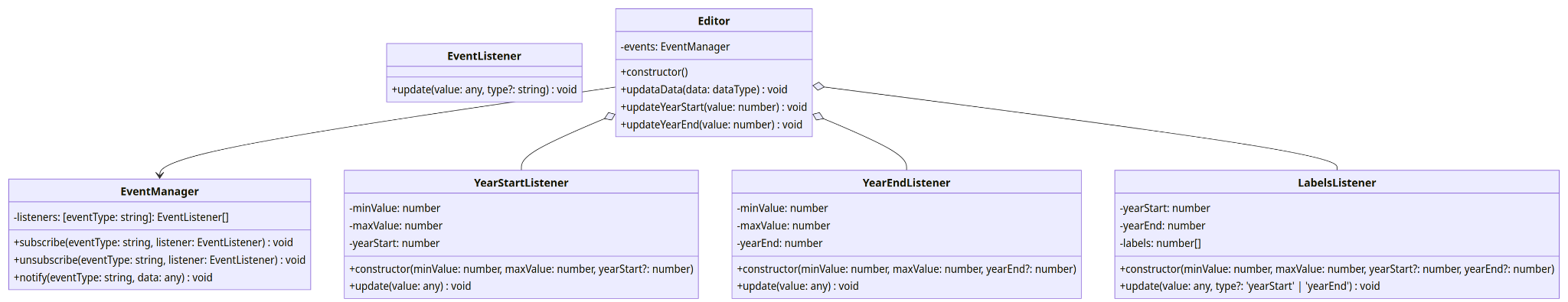
* Можливість використовувати дані з діаграми ChartJS для створення кругової діаграми без необхідності зміни клієнтського коду.
* Підвищення гнучкості системи за рахунок можливості використання різних типів діаграм з однаковим інтерфейсом.

*Обґрунтування використання шаблону:*

Патерн "Адаптер" використовується, коли існуючий клас має інтерфейс, який не відповідає вимогам клієнтського коду. В даному випадку, дані з діаграми ChartJS не можуть бути безпосередньо використані для створення кругової діаграми. Замість зміни клієнтського коду або самого класу ChartJS, використовується PieChartAdapter для адаптації інтерфейсу ChartJS до інтерфейсу, який очікується клієнтом.

1. **Спостерігач**

* EventManager є реалізацією патерну "Спостерігач" (Observer), який використовується для створення механізму підписки, що дозволяє об'єктам слідкувати за подіями, які відбуваються в інших об'єктах. В даному випадку, EventManager дозволяє об'єктам підписуватися на події та отримувати повідомлення про їх відбування.



*Рис. 2.1.4. Діаграма класів, які входять до шаблонів «Спостерігач»*

*Учасники шаблону:*

* Subject (Суб'єкт): Клас, який має механізми для управління підписниками.
* EventManager: Конкретний суб'єкт, який дозволяє об'єктам підписуватися на події та отримувати повідомлення про їх відбування.
* Observer (Спостерігач): Інтерфейс, який визначає метод update, який викликається при відбуванні події.
* YearStartListener, YearEndListener, LabelsListener: Конкретні спостерігачі, які реалізують метод update.

*Результати використання конструкції:*

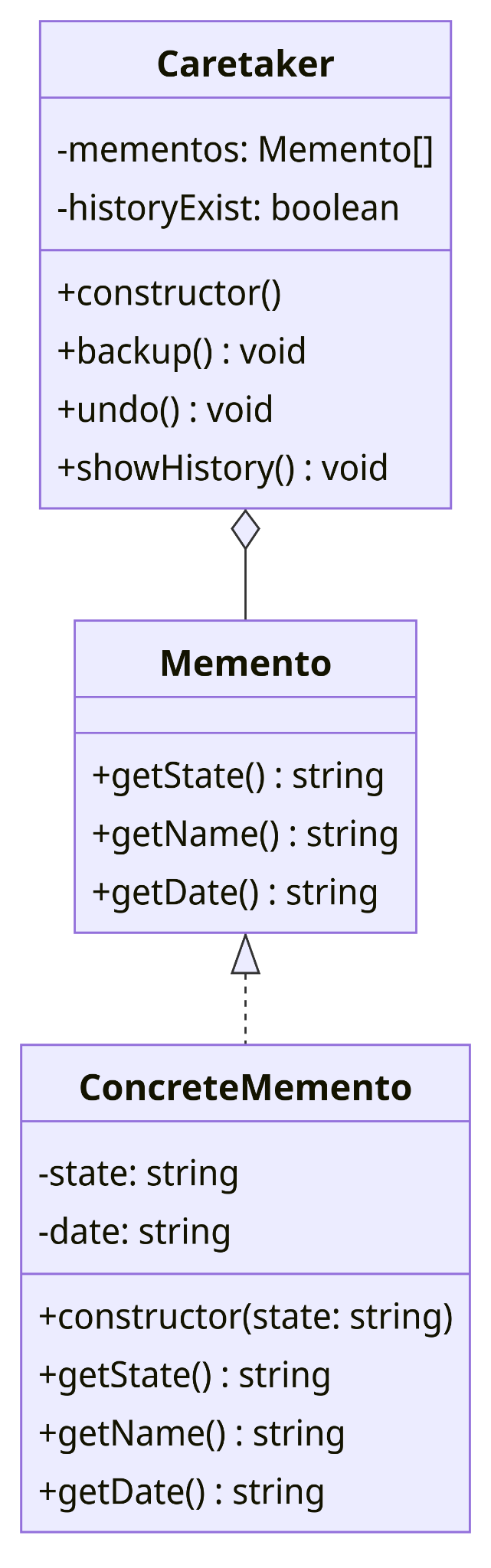
* Можливість динамічно спостерігати за станом об'єктів та реагувати на зміни в їх стані.
* Підвищення гнучкості системи за рахунок можливості додавати нові типи спостерігачів без зміни коду суб'єкта.

*Обґрунтування використання шаблону:*

Патерн "Спостерігач" використовується, коли існує необхідність в установленні відносин "один до багатьох" між об'єктами, таким чином, що при зміні стану одного об'єкта всі його залежні об'єкти повідомляються та оновлюються автоматично. В даному випадку, EventManager дозволяє об'єктам слідкувати за подіями, які відбуваються

1. **Знімок**

Caretaker є реалізацією патерну "Знімок" (Memento), який використовується для збереження та відновлення попередніх станів об'єкта без порушення принципів інкапсуляції. В даному випадку, Caretaker зберігає знімки стану глобального стану додатка.



*Рис. 2.1.5. Діаграма класів, які входять до шаблону «Знімок»*

*Учасники шаблону:*

* Memento (Знімок): Інтерфейс, який визначає методи для отримання стану об'єкта.
* ConcreteMemento: Конкретний знімок, який реалізує Memento і зберігає стан об'єкта.
* Caretaker: Клас, який зберігає знімки і відновлює стан об'єкта з цих знімків.

*Результати використання конструкції:*

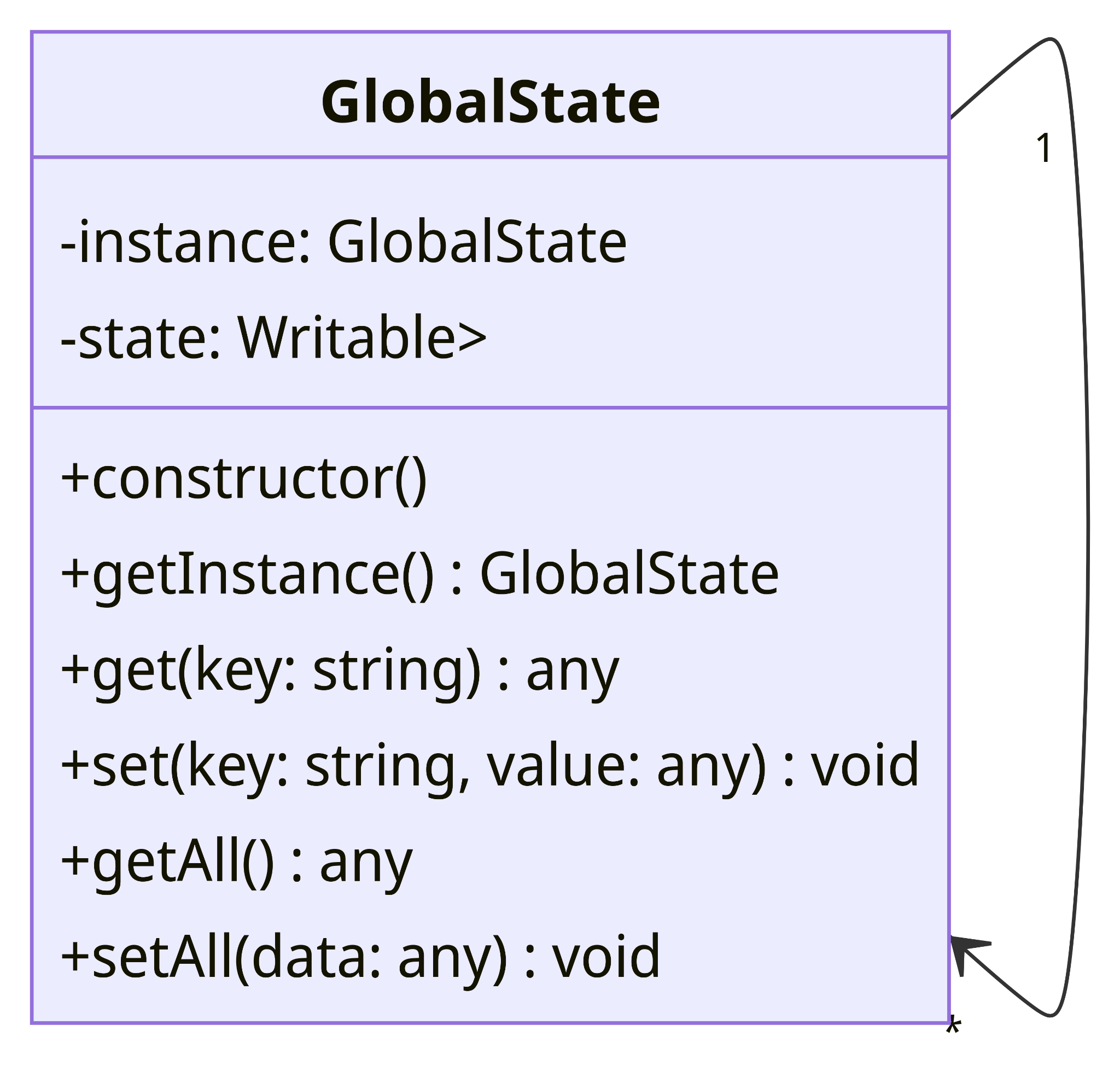
* Можливість зберігати знімки стану об'єкта і відновлювати ці стани пізніше.
* Збереження історії змін стану об'єкта.
* Відновлення стану об'єкта до попереднього стану без порушення принципів інкапсуляції.

*Обґрунтування використання шаблону:*

Патерн "Знімок" використовується, коли необхідно зберегти моментальний знімок (або знімки) стану об'єкта, що може бути відновлений пізніше. В даному випадку, Caretaker зберігає знімки стану глобального стану додатка, дозволяючи відновити ці стани, коли це необхідно.

1. **Одинак**

* GlobalState є реалізацією патерну "Одинак" (Singleton), який гарантує, що клас має лише один екземпляр, і надає глобальну точку доступу до нього. В даному випадку, GlobalState використовується для зберігання глобального стану додатка.



*Рис. 2.1.6. Діаграма класів, які входять до шаблону «Одинак»*

*Учасники шаблону:*

* Singleton (Одинак): Клас, який має лише один екземпляр. В даному випадку, це GlobalState.

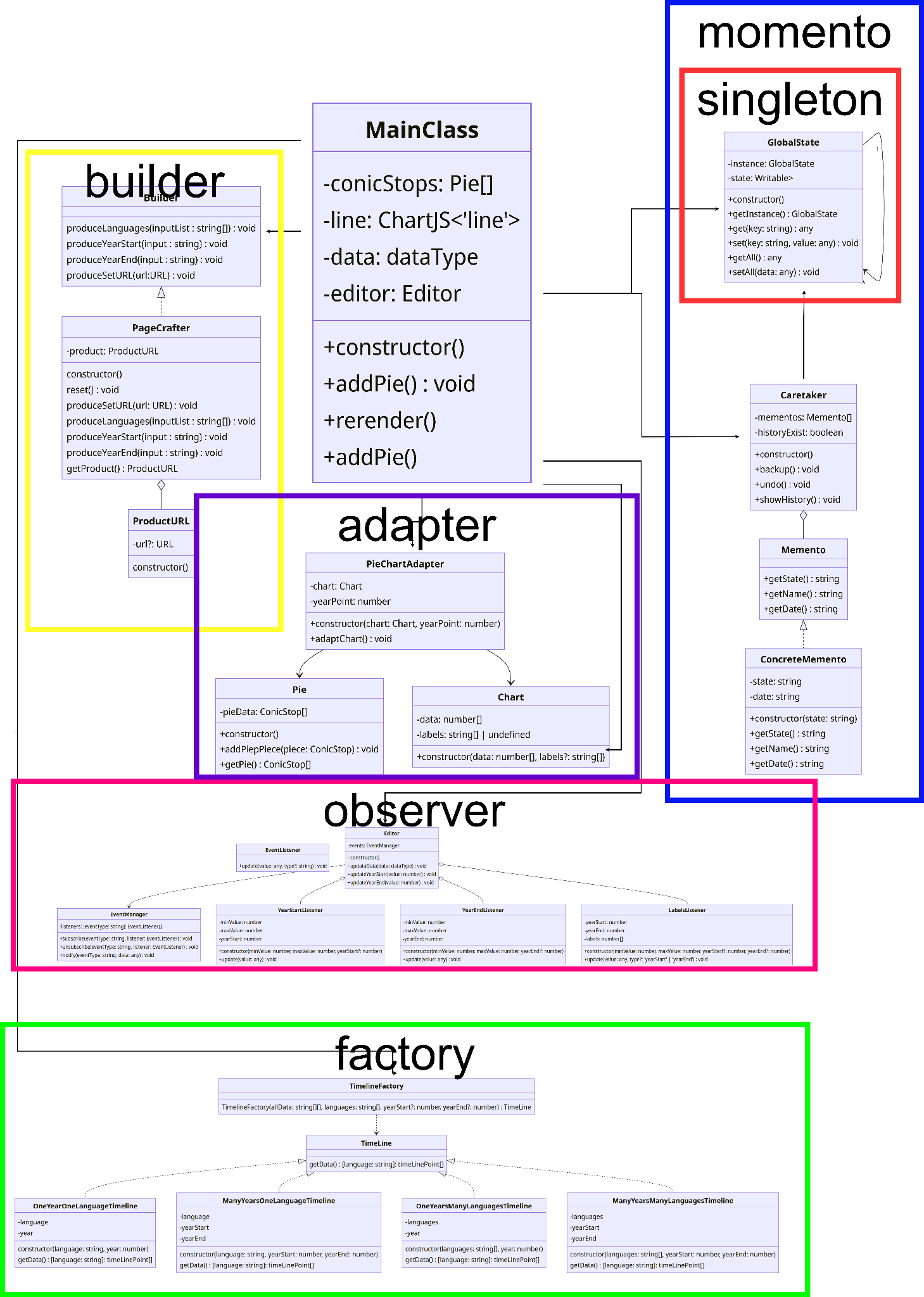
*Результати використання конструкції:*

* Гарантоване існування лише одного екземпляра класу GlobalState в системі.
* Глобальна точка доступу до цього екземпляра.
* Можливість контролювати доступ до глобального стану додатка.

*Обґрунтування використання шаблону:*

Патерн "Одинак" використовується, коли система повинна мати лише один екземпляр класу, доступний глобально. В даному випадку, GlobalState використовується для зберігання глобального стану додатка, і має бути доступний з будь-якої точки додатка. Це робить його ідеальним кандидатом для використання патерну "Одинак".

* 1. **Діаграма класів**

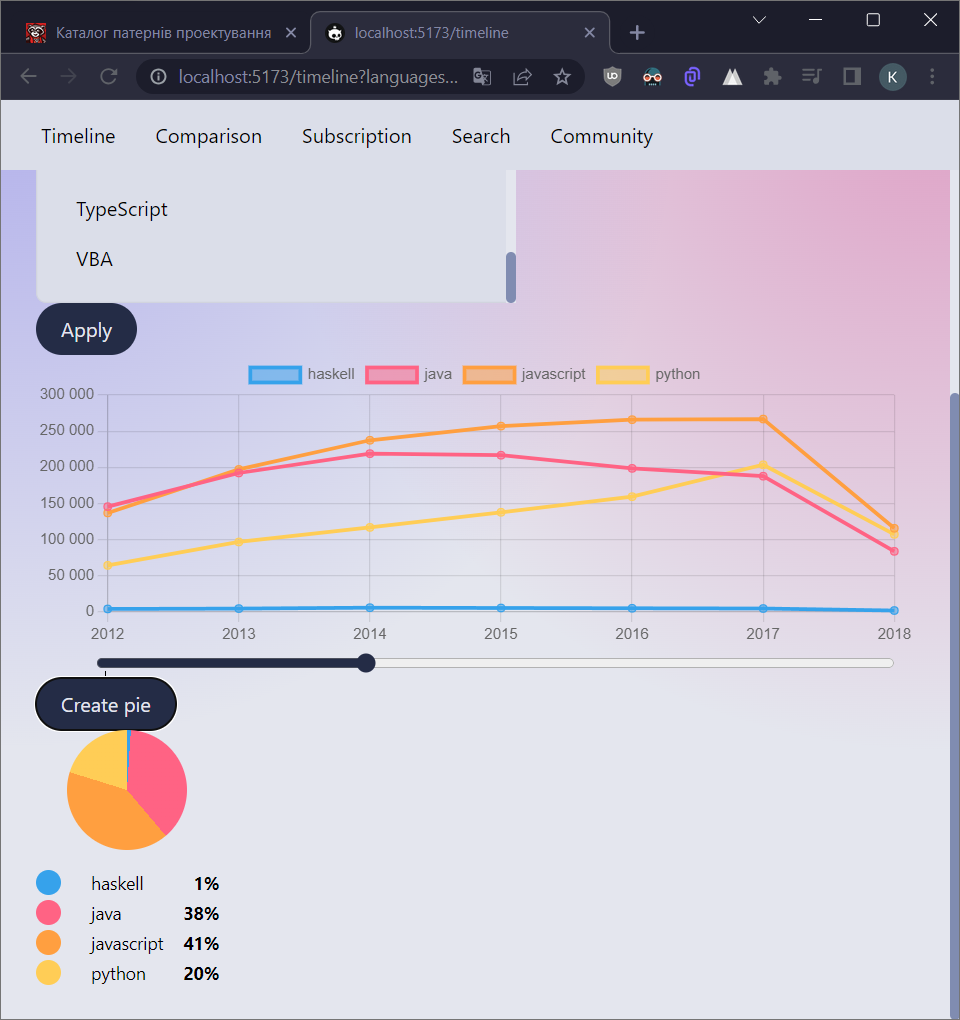
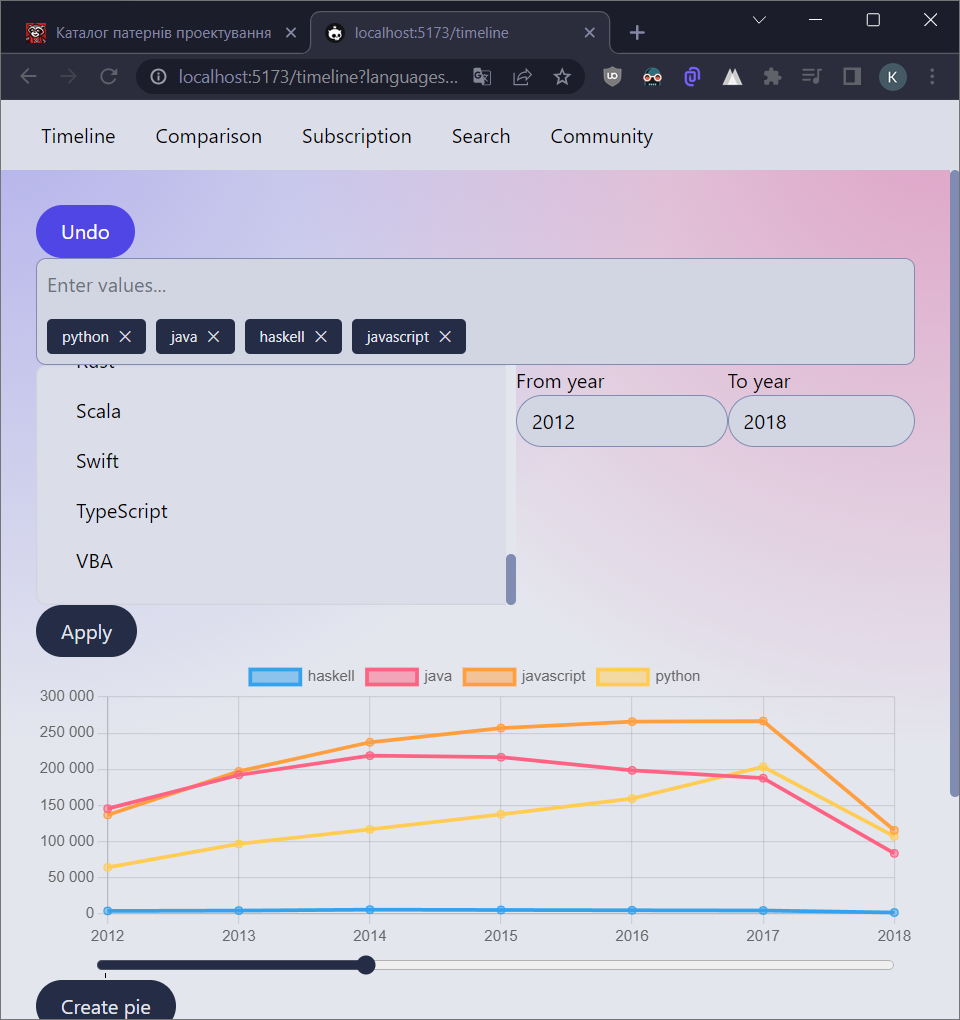


*Рис. 2.2, Діаграма класів web-фреймворку*

* 1. **Опис результатів роботи програми**

Демонстрування результатів роботи розробленого сайту.

Вигляд головної сторінки даного веб-порталу подано на рис. 2.3.1.



*Рис. 2.3.1. Головна сторінка*

## ВИСНОВКИ

Метою даної курсової роботи було розроблення програмного забезпечення для дослідження популярності мов програмування на часових проміжках. Підставою для розроблення стало завдання на виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування» студентами ІІ курсу кафедри ПЗКС НТУУ «КПІ ім. І.Сікорського».

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, програмного та апаратного забезпечення, наведені у технічному завданні; створено відповідну документацію.

Розроблене програмне забезпечення дозволяє просто слідкувати за трендами мов програмування на різних проміжках часу і аналізувати зрізи певної вибірки на певному році.

Програму створено на основі використання шаблонів проектування: зокрема до структури розробленого програмного забезпечення входить реалізація шести шаблонів, які належать до різних груп шаблонів проектування (структурних, породжувальних та поведінкових).

Для розроблення програмного забезпечення використано мову програмування typescript(framework svelte). Платформи: Website.

Перспективним напрямом подальшого дослідження даної тематики є розширення функціоналу даного сайту шляхом денмічного оновлення даних.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. Каталог патернів проектування (<https://refactoring.guru/uk/design-patterns/catalog>)