любых из трех главных типов волокон – должна рассматриваться совместно с планами более далекой перспективы – применения линий на скорость передачи 10 Гбит/с с использованием последовательно установленных линейных усилителей. Высокая скорость передачи в последнем случае может быть достигнута путем оптимизации длины сегментов между линейными усилителями (приблизительно 70 км).

Волокна SF и DSF позволяют осуществлять наращивание сегментов сетей, волокно NZDSF более перспективно при использовании в новых высокоскоростных магистралях. Сравнение волокон SF и DSF, показало, что SF лучше подходят для сетей, использующих волновое мультиплексирование.

Недостаток SF – большое значение дисперсии в окне 1550 нм – может компенсироваться либо дополнительным участком на основе волокна с компенсирующей дисперсией, либо путем уменьшения спектральной ширина излучаемого сигнала (например, используя передатчики на основе DFB лазеров).

Список использованных источников:

1. Убайдуллаев Р.Р. Волоконно-оптические сети // Москва: Эко-трендз, 2000. – 268 с.

2. Дэвид Бейли и Эдвин Райт, Волоконная оптика. Теория и практика // Москва, 2006 – 308 с.

Крвавич Н.Б.

студент,

Національний університет «Львівська політехніка»

ПАТЕРНИ ДЛЯ ВІЗУАЛІЗАЦІЇ ДАНИХ

У повсякденному житті кожен із нас часто стикається з різноманітною інформацією, її подання є дуже важливим.

Невід’ємним елементом опрацювання інформації є комп’ютер. Він може надзвичайно ефективно і точно виконувати завдання які через велику масштабність чи складність є неможливими для людського мозку. Візуалізація даних – це важливий і потрібний процес, оскільки неструктуровані дані які подані за допомогою символів є важкими для сприйняття людиною [2].

Патерни, або, іншими словами, шаблони програмування – це певні структури коду, не прив’язані до конкретної мови які дозволяють реалізувати якісь конкретні проблеми чи завдання які виникають на практиці [1].

У даній роботі було розроблено ряд патернів для проведення візуалізації даних. Для того щоб провести візуалізацію, необхідно щоб система яка буде виконувати дане завдання вирішувала висвітлені нижче проблеми.

1.) Інтерфейс. Потрібно створити зручний для використання інтерфейс із продуманими усіма, потрібними для використання системи, можливостями.

2.) Взаємодія з користувачем. дозволяє збільшити функціональність системи та збільшити рівень зручності для користувача.

3.) Стратегія подання і відображення даних. Для того щоб система була потрібною, їй необхідно володіти певною запотребованою стратегією [3].

Для того щоб створити систему візуалізації даних, необхідна реалізація класів, які вирішують описані проблеми. Ці класи подані нижче.

Creational pattern розділяє об'єкти на структурні одиниці. Результатом побудови будуть відображення різних об'єктів (класи Figure та MonoFigure).

Для того щоб можна було відображати необхідні фігури, є потрібним клас Figure. Його підкласи визначають усі можливі елементарні графічні елементи.

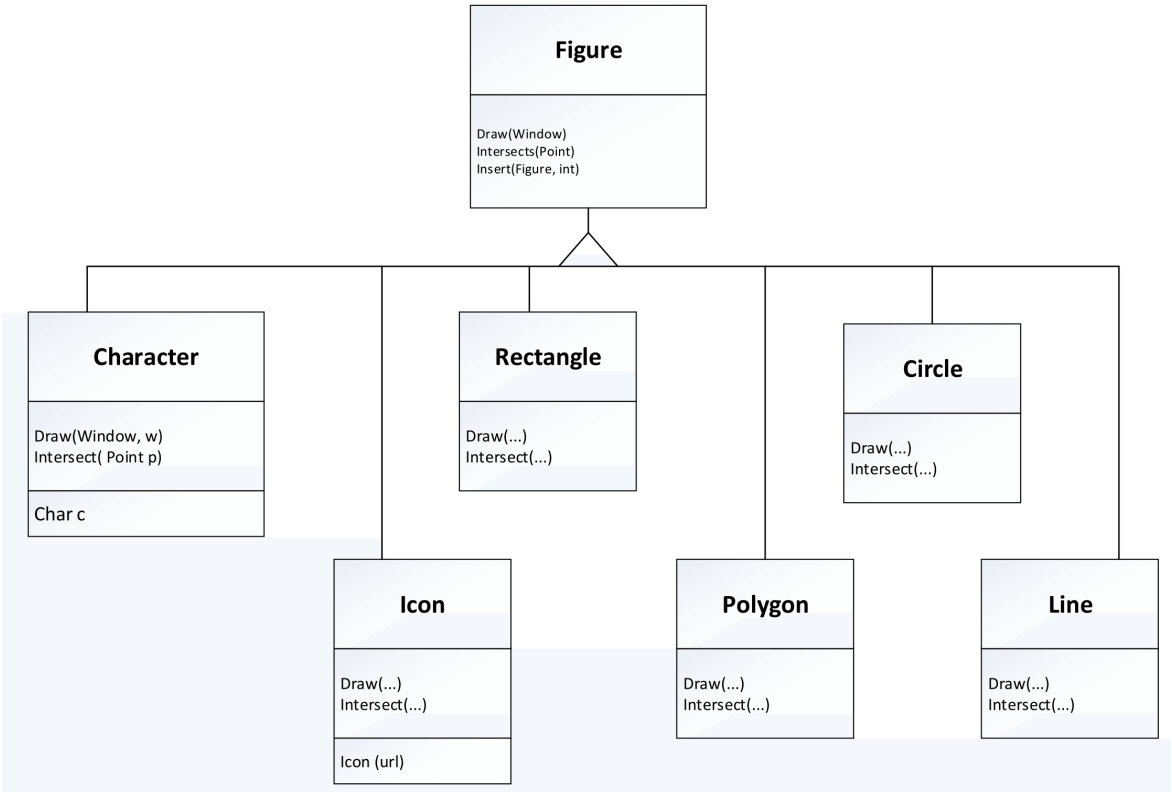

Рис. 1. Ієрархія класу Figure

Джерело: розроблено автором

На рис. 1 зображена ієрархія класу Figure. В цього класу є наступні субкласи: Character, Ractangle, Circle, Polygon, Line and Icon. Усі вони перевизначають функції Draw() та Intersect().

Субклас Rectangle перевизначає Draw() таким чином:

void Rectangle : : Draw (Window w) {w >DrawRect (_x0, _y0, _x1, _y1)},

де x0, y0, x1 і y1 є аргументами даної функції, які визначають два протилежні кути прямокутника. DrawRect – це операція, яка малює прямокутник на екрані.

Для того щоб можна було утворювати композицію кількох фігур потрібен клас MonoFigure. На рис. 2 зображена ієрархія класу MonoFigure.

Цей клас реалізує операцію Draw() таким чином:

void MonoFigure : : Draw (Window w) {_component >Draw (w)}.

Structural pattern поєднує класи і об’єкти у загальну структуру. До цієї категорії відносяться класи Compositor, Composition і Decorator.

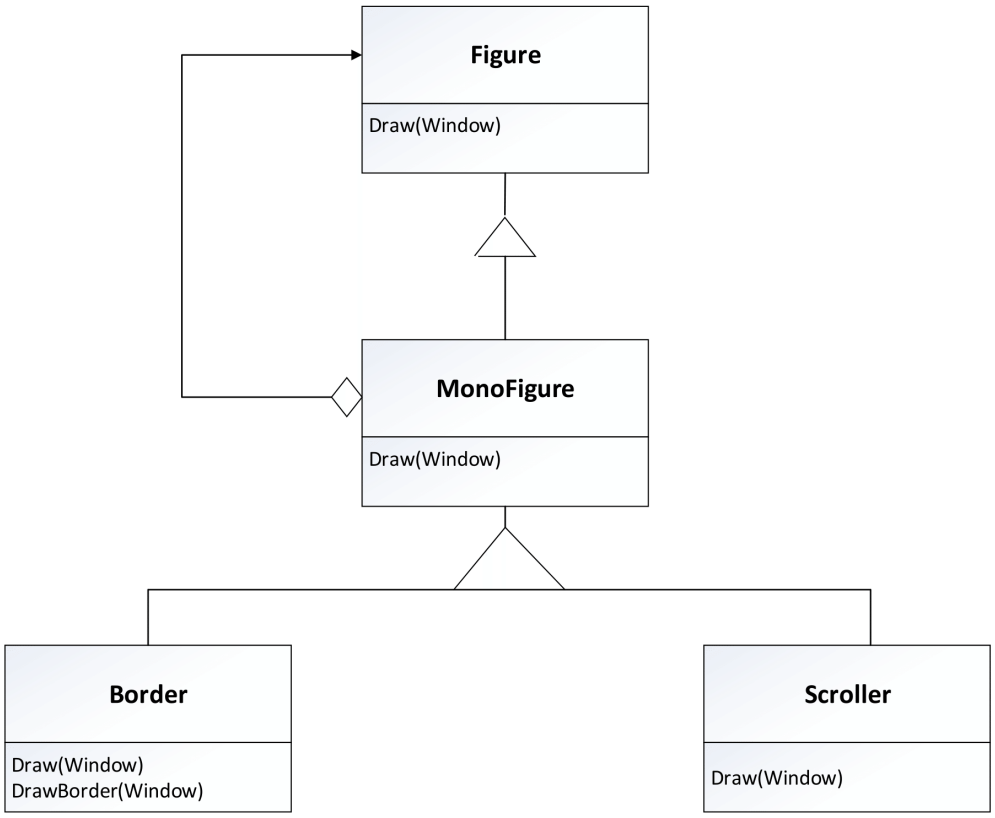

Рис. 2. Ієрархія класу MonoFigure

Джерело: розроблено автором

Для визначення ієрархії класів які складаються із примітивних та компонованих об’єктів можна використовувати класи Compositor і Composition. Примітивні об’єкти можуть бути поєднані в більш складні структури, а ці в свою чергу також із іншими, це може повторюватись рекурсивно.

Реалізація цього патерну також може сприяти легкому додаванню чи видаленню об’єктів які належать до певної структури, а також виконанню композиції оптимальним способом.

На рис. 3 зображено взаємозв’язок класів Compositor і Composition. Модель класу Decorator фіксує взаємо’звязки класів і об’єктів, а також елементи їх декору. З його допомогою можна додавати чи змінювати обов’язки окремих об’єктів динамічно. Декоратор застосовується до інтерфейсу компонента і прикрашає його таким чином, щоб його присутність була зрозумілою та прозорою. Декоратор може виконувати додаткові дії (такі як малювання полів чи смуг прокрутки). Використовуючи рекурсивне використання декоратора, забезпечується необмежена кількість доданих обов’язків чи компонентів. На рис. 4 зображена ієрархія класу Decorator.

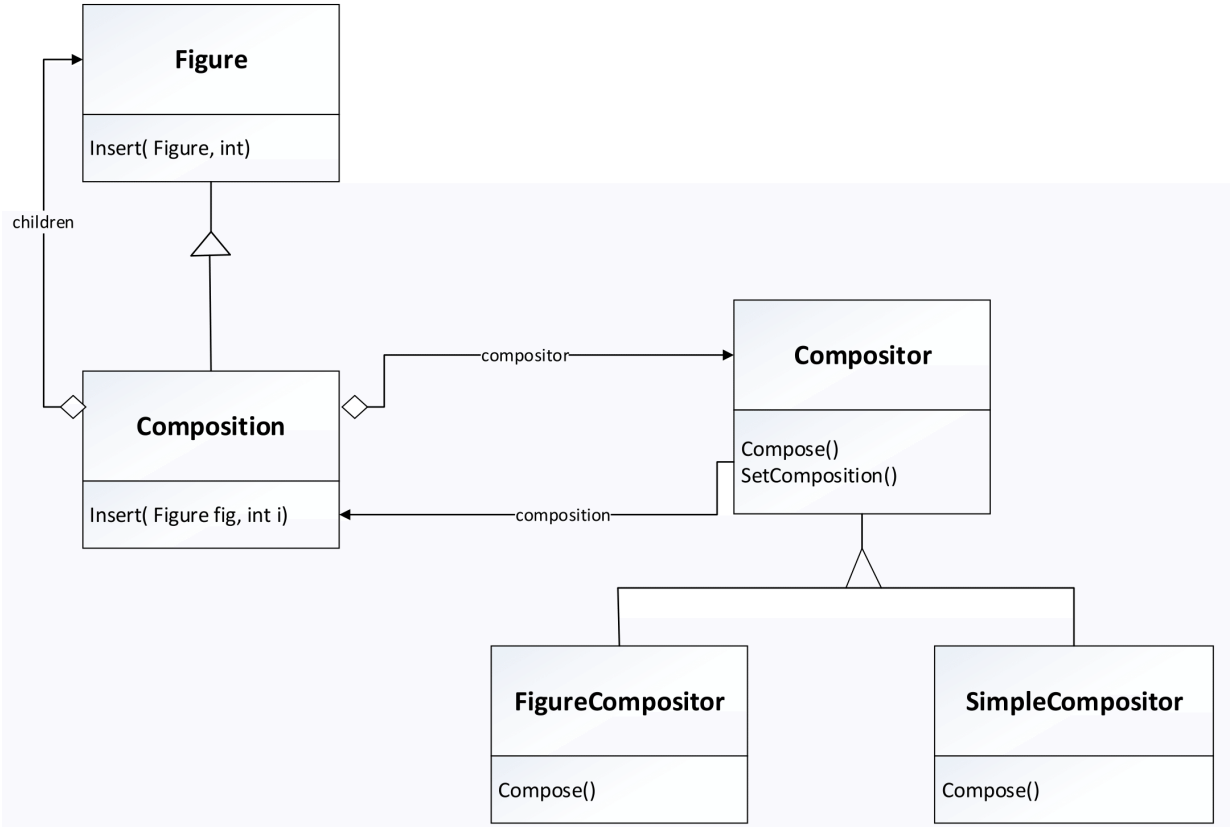

Рис. 3. Взаємозв'язок класів Compositor і Composition

Джерело: розроблено автором

Behavioral pattern пов'язаний із алгоритмами та розподілом завдань між об'єктими. Він описує не лише шаблони об'єктів чи класів, але й взаємозв'язок між ними.

Інкапсуляція об'єктів в алгоритмі є ціллю моделі класу Strategy. Ключовими учасниками в структурі є об'єктами стратегії (яка інкапсулює різні алгоритми) і контексту, в якому вони працюють.

Метою застосування класу Strategy є розробка інтерфейсів стратегії і їх контекст, що є досить загальним підтриманням алгоритмів. У даному прикладі, базова підтримка інтерфейсу Figure для доступу, вставки і видалення. Вцілому фізична структура об'єкта, незалежно від алгоритму використовує список завдань для нього.

На рис. 5 зображеноа ієрархія класу Strategy.

Дана тема є актуальною у наш час і потребує додаткового дослідження й розробки нових методів та засобів, оскільки з рівнем інформатизації суспільства зростає зацікавленість у якості подання та розуміння даних які нас оточують.

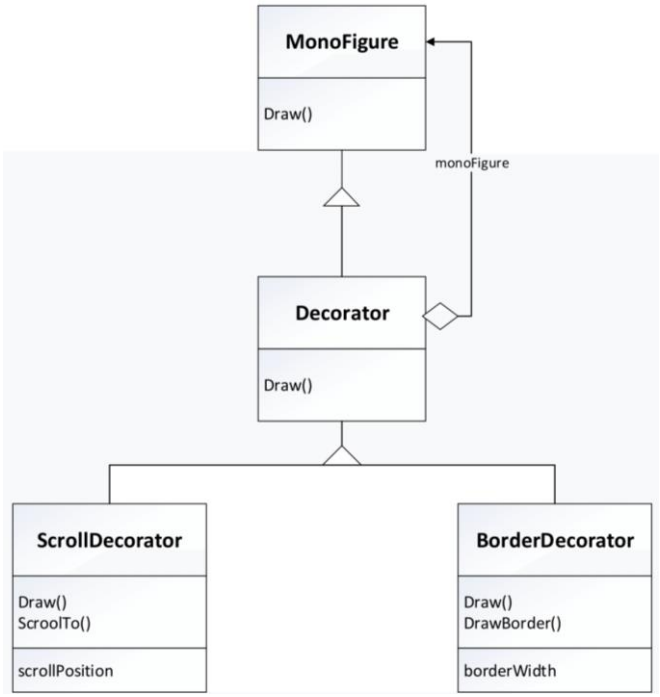

Рис. 4. Ієрархія класу Decorator

Джерело: розроблено автором

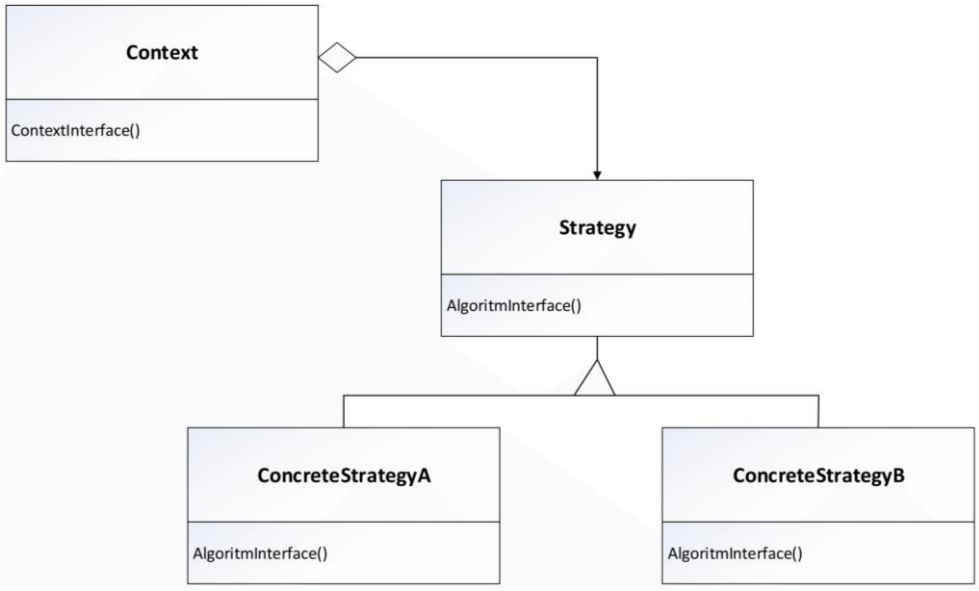

Рис. 5. Ієрархія класу Strategy

Джерело: розроблено автором

Список використаних джерел:

1. Allen Holub Holub on Patterns.Learning design patterns by looking at code / A. Holub – USA : Apress-2011. – 380 p.

2. Scott Murray Interactive Data Visualization for the Web / S. Murray. – Canada : O'Reilly Media, 2013 – 268 p.

3. Stoyan Stefanov JavaScript Patterns / S. Stefanov. – USA : O'Reilly Media, 2010 – 262 p.