**Практична робота №2-1 Побудова діаграм класів.**

**Мета:** Ознайомитися з принципами побудови діаграм класів засобами мови UML та застосувати їх на практиці.

**Завдання:**

1. Опрацювати теоретичні відомості. Перевірити засвоєння матеріалу на контрольних запитаннях.
2. Розробити діаграми класів для застосунку, який реалізуємо в ході вивчення ООП: застосунок до власної курсової роботи з ООП.

На діаграмі надати нотатку, де вказати тему курсової, групу і прізвище.

При командній розробці застосунку на діаграмі виділити класи, які є вашою сферою особистої відповідальності.

1. Оформити звіт, до якого включити розроблені діаграми та письмові відповіді на запитання в кінці цієї ПР.

Файл зі звітом надавати з іменем у форматі

**ОPI<Номер групи><Номер лекції / практичної / лабораторної>[-<Номер завдання>][літера позначення типу роботи L – лекція, P –практична, R – лабораторна]<Прізвище англійською>**. Наприклад, **ОPI3104L**buts.doc. Відповіді на запитання повинні бути не довгими і змістовними. Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності відповідей-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

**Строк виконання цієї роботи ІПЗ-31 27.03.2024**

**ІПЗ-32**

## Теоретичні відомості.

Діаграма класів відображає класи та їх взаємовідносини, тим самим представляючи логічний аспект проекту. Кожна діаграма класів представляє певний ракурс структури класів. На стадії аналізу діаграми класів використовуються, щоб виділити загальні ролі й обов'язки сутностей, які забезпечують необхідну поведінку системи. На стадії проектування діаграми класів застосовують, щоб передати структуру класів, які формують архітектуру системи.

Кожен клас повинен мати ім'я і, якщо ім'я занадто довге, його можна скоротити або збільшити сам значок на діаграмі. Ім'я кожного класу повинно бути унікальним в поточному проекті.

Діаграма класів визначає типи об'єктів системи й різного роду статичні зв'язки, які існують між ними. Є два основних види статичних зв'язків:

• асоціації (наприклад, менеджер може вести кілька проектів);

• підтипи (працівник є різновидом особистості).

На діаграмах класів зображуються також атрибути класів, операції та обмеження, які накладаються на зв'язки між об'єктами. типова діаграма класів зображена на рис. 1.

*Асоціації.* Асоціації являють собою зв'язки між екземплярами класів (особистість працює в компанії, компанія має ряд офісів).



Рис. 1. Типова діаграма класів

Будь-яка асоціація володіє двома ролями; кожна роль являє собою напрямок асоціації. Таким чином, асоціація між «Виконавцем» і «Звітом» містить дві ролі: одна від «Виконавця» до «Звіту»; інша – від «Звіту» до «Виконавця». Роль може бути явно пойменована за допомогою позначки (мітки). Якщо позначка відсутня, ролі привласнюється ім'я класу-мети. таким чином, роль асоціації від «Виконавця» до «Звіту» може бути названа «Звітом».

Роль також має множинність, яка вказує, скільки об'єктів може брати участь у даному зв'язку. На рис. 1 символ «0..\*» над асоціацією між «Менеджером» й «Контрактом» вказує, що з одним «Менеджером» може бути зв'язано багато «Контрактів», а символ «1» показує, що будь-який «Контракт» управляється одним «Менеджером».

У загальному випадку множинність вказує нижню й верхню границі кількості об'єктів, які можуть брати участь у зв'язку. Для цього можуть використовуватися однина, діапазон або дискретна комбінація із чисел і діапазонів.

Для асоціації може бути зазначений напрямок навігації. Якщо навігація зазначена тільки в одному напрямку, то така асоціація називається односпрямованою (рис. 1, асоціація між «Менеджером» та «Звітом»). У двоспрямованій асоціації навігація зазначена в обох напрямках. У мові UML відсутність стрілок в асоціації трактується наступним чином: напрямок навігації невідомий або асоціація є двоспрямованою.

*Атрибути*

Атрибути багато в чому подібні до асоціацій. Різниця між ними полягає в тому, що атрибути припускають єдиний напрямок навігації – від типу до атрибута.

На рис. 1 атрибути зазначені для класів «Контракт» і «Звіт». Залежно від ступеня деталізації діаграми, позначення атрибута може включати ім'я атрибута, тип і значення за замовчуванням. У синтаксисі UML це виглядає так: <ознака видимості> <ім'я> : <тип> = <значення за замовчуванням>, де ознака видимості може набувати одне з чотирьох значень:

• загальний (public) - атрибут доступний для всіх клієнтів класу;

• захищений (protected) - атрибут доступний тільки для підкласів і друзів класу;

• закритий (private) - доступний тільки для друзів класу;

• реалізація (implementation) - атрибут доступний тільки усередині пакета, що обрамляє.

*Операції*

Операції являють собою процеси, реалізовані класом. Найбільш очевидна відповідність існує між операціями й методами класу. Повний синтаксис UML для операцій виглядає так: <ознака видимості> <ім'я> (<список-параметрів>) : <тип виразу, що повертає значення> = <рядок-властивостей>, де:

• ознака видимості може набувати ті ж значення, що й для атрибутів;

• ім'я являє собою символьний рядок;

• список параметрів містить необов'язкові аргументи, синтаксис яких збігається із синтаксисом атрибутів;

• тип виразу, що повертає значення, є необов'язковою специфікацією й залежить від конкретної мови програмування;

• рядок властивостей показує значення властивостей, які застосовуються до даної операції. Прикладом операції на рис.1 є операція закриття класу «Контракт».

*Узагальнення*

Типовий приклад узагальнення містить «Команду проекту» й «Субпідрядника» (див. рис. 1). Вони мають деякі розходження, однак у них є також багато спільного. Однакові характеристики можна помістити в узагальнений клас «Виконавець» (супертип), при цьому класи «Команда проекту» й «Субпідрядник» будуть виступати як підтипи.

Зміст узагальнення полягає в тім, що інтерфейс підтипу повинен включати всі елементи інтерфейсу супертипу. Інша сторона узагальнення пов'язана із принципом підстановки. Субпідрядника можна підставити в будь-який код, де потрібен «Виконавець», і при цьому все повинно нормально працювати. Це означає, що, розробивши код, який припускає використання «Виконавця», можна вільно вживати екземпляр будь-якого підтипу «Виконавця». Субпідрядник може реагувати на деякі команди відмінним від іншого «Виконавця» чином (відповідно до принципу поліморфізму), але ця відмінність не повинна турбувати об'єкт, що викликає «Виконавця».

Узагальнення з погляду реалізації пов'язане з поняттям успадкування в мовах програмування. Підклас успадковує всі методи й поля суперкласу і може перевизначити наслідувані методи. Підтип можна також реалізувати, використовуючи механізм делегування.

*Обмеження*

При побудові діаграм класів основним завданням є відображення різних обмежень. На рис. 1 показано, що «Контракт» може управлятися тільки одним «Менеджером».

За допомогою конструкцій асоціації, атрибута й узагальнення можна специфікувати найбільш важливі обмеження, але неможливо виразити їх усі.

**Контрольні запитання**.

1. Який основний вид діаграм у концептуальній моделі
2. Яке призначення логічної моделі?
3. Назвіть основний вид діаграм у фізичній моделі.
4. Що являє собою діаграма класів?
5. Яке призначення діаграми класів?
6. Дайте визначення поняттю «клас».
7. Поясніть базові відношення в UML. Що являє собою асоціація? У чому зміст множинності асоціацій? У чому відмінність атрибутів від асоціацій?
8. Що являє собою операція класу? У чому зміст узагальнення?

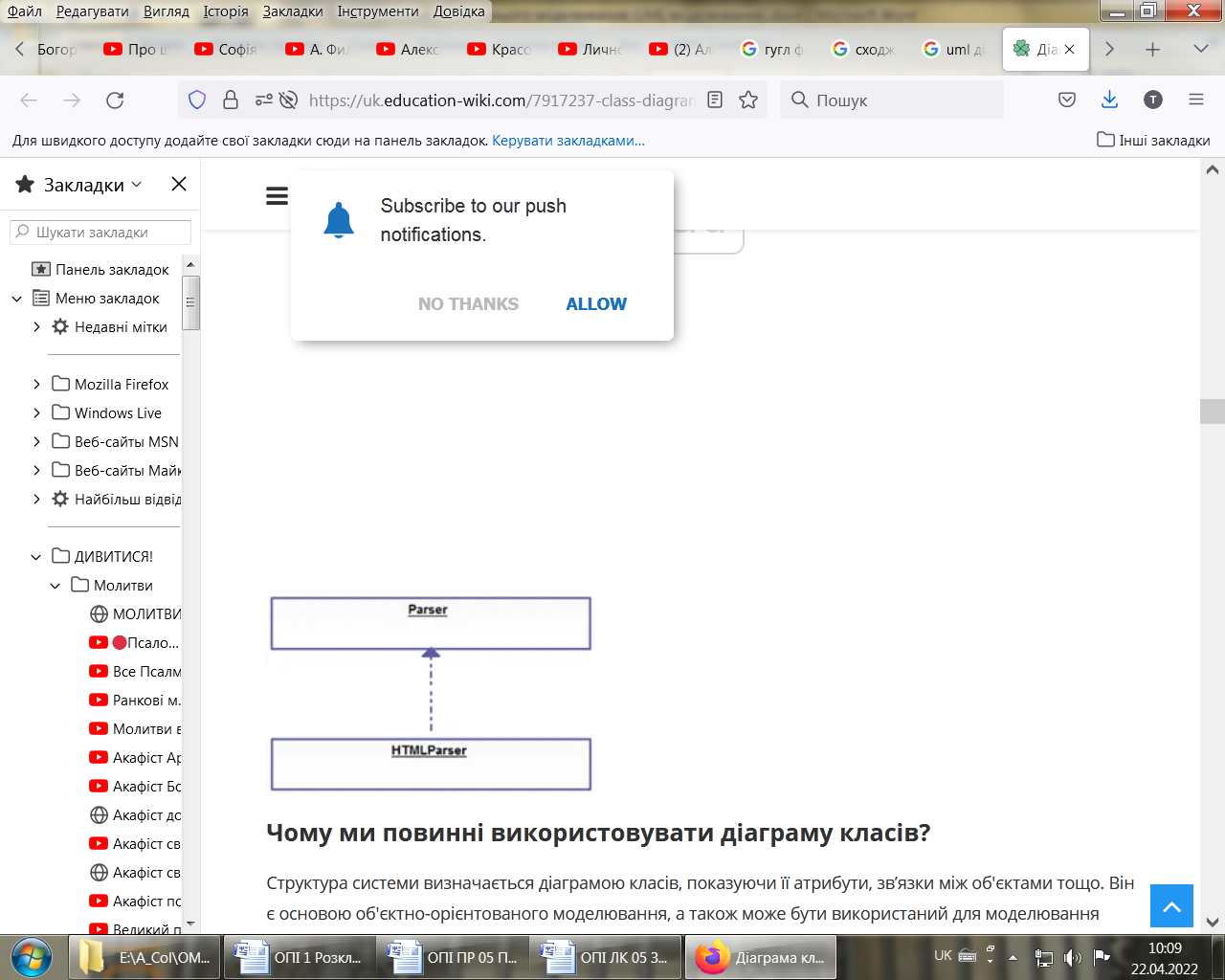
*Запитання для письмової відповіді.*

1. Які основні типи моделей, на Вашу думку, потрібно використовувати під час проектування? (див. Лекцію 5)? Чи вистачає Вам вивченої інформації по діаграмах для розроблення проекту? Відповідь поясніть.
2. Подумайте, чи доцільніше зробити окремий один клас для зв’язку викладача з групами. Відповідь поясніть.

<https://www.omg.org/spec/UML/2.5.1/> About the Unified Modeling Language Specification Version 2.5.1

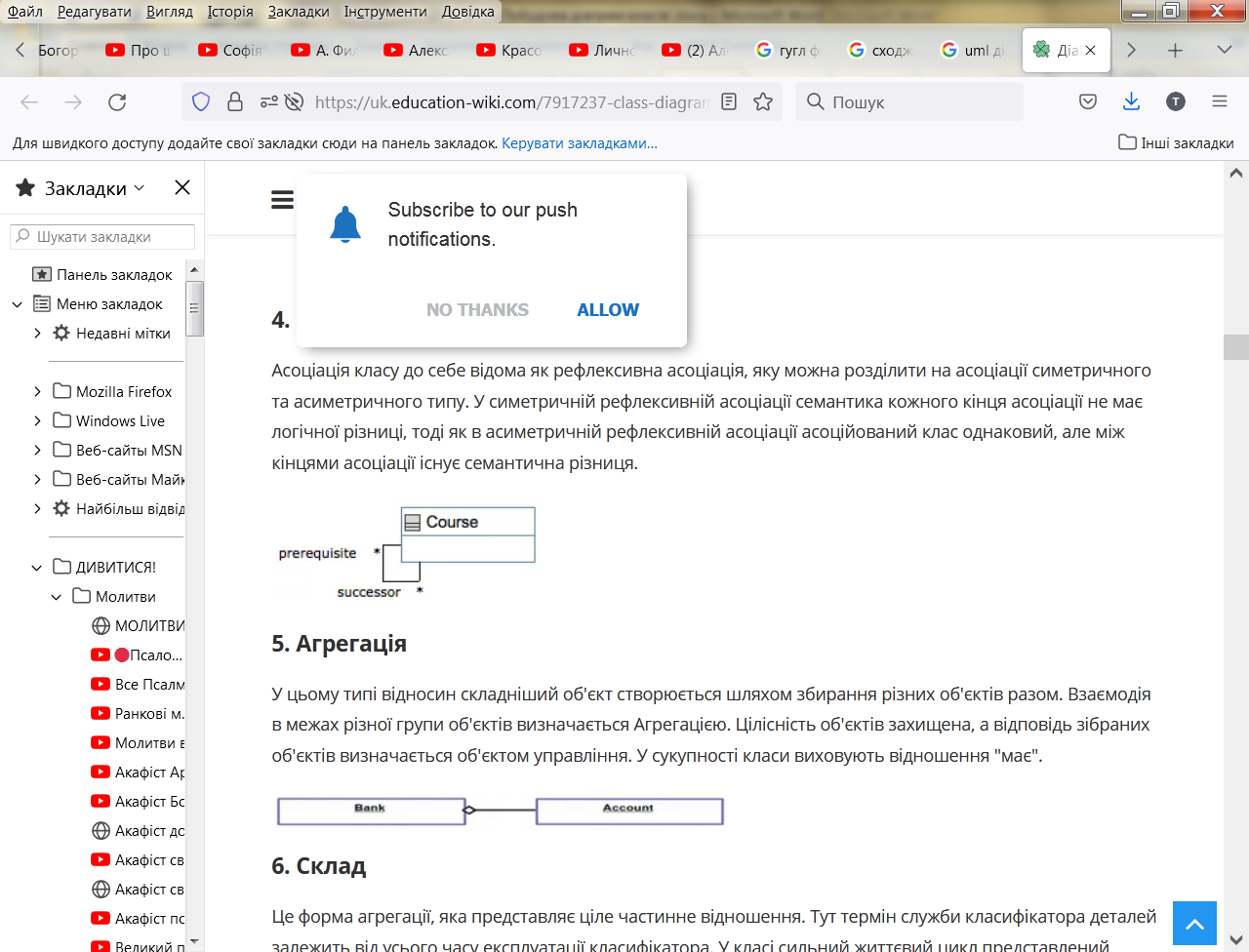
#### Реалізація

Поведінка одного елемента моделі реалізується заданою поведінкою іншого елемента моделі. Цей тип відносин не має імен.



#### Рефлексивна асоціація

Асоціація класу до себе відома як рефлексивна асоціація, яку можна розділити на асоціації симетричного та асиметричного типу. У симетричній рефлексивній асоціації семантика кожного кінця асоціації не має логічної різниці, тоді як в асиметричній рефлексивній асоціації асоційований клас однаковий, але між кінцями асоціації існує семантична різниця.

prerequisite – передумова: successor-спадкоємець

### Чому ми повинні використовувати діаграму класів?

Структура системи визначається діаграмою класів, показуючи її атрибути, зв’язки між об'єктами тощо. Він є основою об'єктно-орієнтованого моделювання, а також може бути використаний для моделювання даних. Діаграми класів допомагають складати попередні плани, які полегшують процес програмування. Більше того, ви завжди можете змінити діаграму класів, оскільки це дратує кодування різної функціональності після фактів. Це проектний план, на основі якого будується система. Це легко зрозуміти, не вимагаючи особливих технічних знань.

Діаграма класів забезпечує статичний вигляд програми, а його здатність до відображення за допомогою об'єктно-орієнтованої мови робить його готовим до використання в побудові. На відміну від діаграми послідовностей, діаграми діяльності тощо, діаграма класів є найпопулярнішою діаграмою UML. Нижче наведено призначення діаграми класу.

* Статичний вигляд програми розробляється та аналізується.
* Обов'язки системи описані нею.
* Компоненти та основа діаграми розгортання є діаграма класів.
* На діаграму класів впливає пряма та зворотна інженерія.

### Переваги діаграми класів

Діаграма класів може бути реалізована на різних етапах проекту і є серцем UML. Представлення реальності створюється діаграмою класів, з'являючись на доменній моделі під час аналізу. Моделювання програмного забезпечення проводиться на етапі проектування, тоді як код формується на етапі впровадження. Основою програмних продуктів є діаграми класів, які є невід'ємною частиною будь-якого проекту.

Почуття орієнтації задають діаграми класів. Структура системи детально аналізується на класовій діаграмі, а також синергія між різними елементами переглядається ними разом із їх властивостями. Це швидко і легко читати, і його можна легко створити, якщо потрібне програмне забезпечення є на місці. Будь-яка система, яку потрібно створити, діаграми класів є основою для цього.

### Переваги

* Будь-яку просту або складну модель даних можна проілюструвати за допомогою діаграми класів для отримання максимальної інформації.
* Схематику програми можна зрозуміти за допомогою неї.
* Будь-яка потреба в системі може бути візуалізована та передана по всьому бізнесу для конкретних дій.
* Будь-яка вимога щодо реалізації конкретного коду може бути виділена за допомогою діаграм і запрограмована до описаної структури.
* Опис, незалежний від реалізації, може бути наданий та переданий компонентам.

### Недоліки діаграми класів

Хоча діаграма класів - це перше, що слід врахувати у виробничих умовах, щоб створити бездоганну систему, проте, безумовно, є і її неабияка частка мінусів.

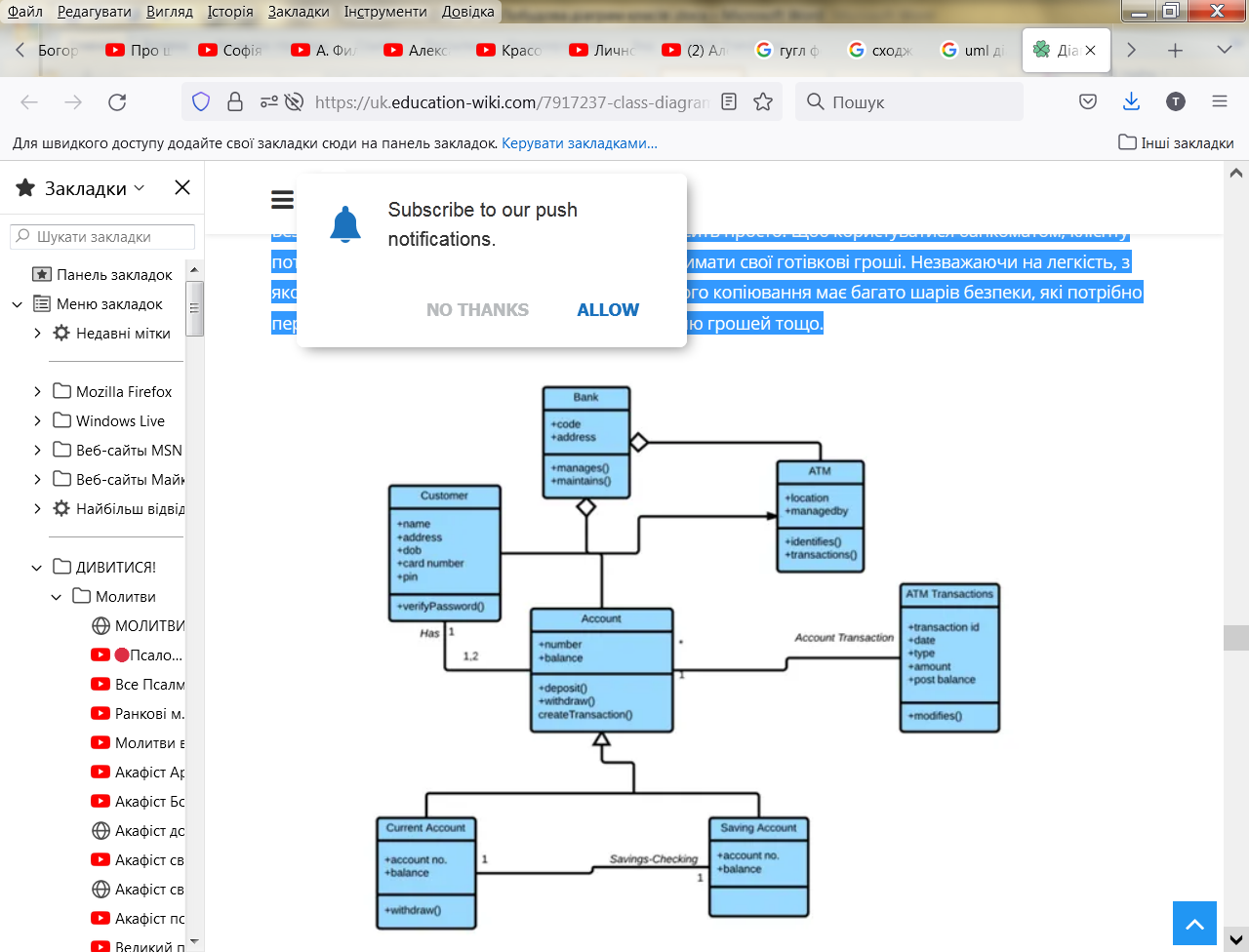
Діаграми класів часто можуть зайняти більше часу та підтримувати, що інколи дратує розробника. Для його налаштування та обслуговування потрібен час для синхронізації з програмним кодом. Часто розробникам або невеликим компаніям важко синхронізувати код, оскільки це вимагає додаткової кількості роботи.

* Недолік розуміння бенефіціара діаграми також є недоліком. Оскільки розробники програмного забезпечення працюють з кодом, іноді діаграми класів не дуже допомагають. Однак керівники проектів можуть отримати користь від діаграм, оскільки це дає огляд робочого процесу певного інструменту. Отже, часто є аргумент, щоб не витрачати час на діаграми класів, а зосередитись на використанні дошки або паперу для нанесення діаграми.
* Завищена або занижена діаграма не допомагає розробникам програмного забезпечення в їх роботі. Можуть виникнути ситуації, коли розробники засмучуються через структуру діаграм класів. Картографування кожного окремого сценарію може зробити діаграму безладним і важким для роботи. Використання інформації високого рівня може якось допомогти боротися з такими проблемами.
* Розміщення конструкції може перешкоджати розробникам та компаніям. Зацікавлені сторони могли б легко проаналізувати проблеми, переглянувши схему класів, і прикладання занадто великих зусиль до функцій програмного забезпечення може призвести до втрати уваги. Людям потрібно зайнятися фактичною роботою, а не витрачати час на перегляд діаграми та вирішення питань.

Як бачите, незважаючи на важливість діаграми класів у життєвому циклі розробки програмного забезпечення, вона, звичайно, не позбавлена недоліків і може ускладнити життя розробникам та компаніям, якщо не використовувати їх розумно.

### Приклад діаграми класів

Без суєти технічних обмежень схему створити досить просто. Щоб користуватися банкоматом, клієнту потрібно лише натиснути кілька кнопок, щоб отримати свої готівкові гроші. Незважаючи на легкість, з якою витікають грошові кошти, система резервного копіювання має багато шарів безпеки, які потрібно передати для запобігання шахрайству, відмиванню грошей тощо.



Як бачимо тут, існує декілька сутностей, які слідують властивостям різних відносин, як описано раніше. Ці відносини описують структуру, в якій побудована система банкоматів, та рівні безпеки, які вона повинна пройти, щоб забезпечити прозорість та цілісність транзакції.

Є три перспективи, за якими можна розділити діаграму класів -

1. По-перше, це концептуальна перспектива, яку об'єкти реального світу описують за допомогою концептуальних діаграм. Домен, що вивчається, представлений діаграмою. Вона не залежить від мови та пов'язана з класом.
2. Програмні компоненти описуються в перспективі Специфікації з інтерфейсами та специфікаціями. Однак у випадку конкретного впровадження жодних зобов'язань не надається.
3. Конкретну мовну реалізацію можна здійснити за допомогою діаграм перспективних класів впровадження.

### Робота з діаграмою класів

Для розробки програмного забезпечення найважливішою діаграмою UML є Діаграма класів. Щоб скласти діаграму класів, що представляє різні аспекти програми, кілька властивостей, які потрібно врахувати, -

* Змістовна назва повинна бути дана діаграмі класів, що описує реальний аспект системи.
* Потрібно, щоб заздалегідь зрозуміти взаємозв'язок між кожним елементом.
* Щоб розробити кращий продукт, слід визнати відповідальність серед класів.
* Щоб уникнути ускладнення діаграми, слід вказати конкретні властивості класу.
* Документація є хорошою практикою в будь-якому проекті з розробки програмного забезпечення. Таким чином, для визначення будь-якого аспекту на діаграмі потрібна відповідна документація або примітки, щоб їх зрозуміли інші. Команда з розробки програмного забезпечення в кінці повинна розуміти, що налаштовано на діаграмі.
* Малювання на дошці або звичайному папері потрібно до створення остаточної версії. Однак потрібно переконатися, що подається лише готова діаграма, яка може включати в себе кілька змін.

За матеріалами https://uk.education-wiki.com/7917237-class-diagram