МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ФАХОВИЙ КОЛЕДЖ ПРОМИСЛОВОЇ АВТОМАТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

ЗАТВЕРДЖУЮ
Заступник директора з
навчально-методичної роботи
_____ Вікторія ОКСАНІЧЕНКО
___.___.2022 року

Методичні вказівки до виконання практичних робіт

з дисципліни Конструювання програмного забезпечення
(Назва дисципліни)

галузі знань 12 «Інформаційні технології»
(шифр і назва галузі знань)

спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

освітня програма Інженерія програмного забезпечення
(назва освітньо-професійної програми)

Одеса 2022

Методичні вказівки для виконання практичних робіт з дисципліни «Конструювання програмного забезпечення» для підготовки фахових молодших бакалаврів за спеціальністю 121 «Інженерія програмного забезпечення».

Укладач викладач вищої кваліфікаційної категорії ФКПАІТ ОНТУ Тетяна КОСТИРЕНКО

Методичні вказівки для виконання практичних робіт затверджені на засіданні циклової комісії комп'ютерних наук та інженерії програмного забезпечення ФКПАІТ ОНТУ

Протокол від ___._____.20___ року, № ____

Голова циклової комісії "Комп'ютерних наук та інженерії програмного забезпечення"

_____Тетяна КОСТИРЕНКО

(підпис) (власне ім'я та прізвище)

___._____.20___ року

Методичні вказівки до виконання практичних робіт з дисципліни «конструювання програмного забезпечення» розроблено згідно навчальної програми з предмету.

Предметом вивчення навчальної дисципліни є основи конструювання програмного забезпечення та уніфікована мова моделювання програмного забезпечення UML.

Метою вивчення навчальної дисципліни є оволодіння студентами теоретичних знань та набуття практичних навиків з дисципліни «конструювання програмного забезпечення», необхідних для спеціальної підготовки та майбутньої професійної діяльності.

Основними завданнями вивчення дисципліни є формування у студентів базових уявлень про конструювання програмного забезпечення та основи моделювання програмного забезпечення; інтелектуальний розвиток особистості, розвиток у студентів логічного мислення і просторової уяви, алгоритмічної, інформаційної та графічної культури, пам’яті, уваги, інтуїції; формування у студентів компетенцій за фахом.

Згідно з вимогами освітньо-професійної програми студенти повинні

Вміти:

- Виділяти об’єкти та класи по заданій предметній області;
- Встановлювати між об’єктами та класами відносини;
- Будувати статичні модель об’єктно-орієнтованих програмних систем;
- Будувати динамічні модель об’єктно-орієнтованих програмних систем;
- Будувати моделі реалізації об’єктно-орієнтованих програмних систем;
- Виконувати структурне тестування на базі потоку управління.

Практична робота №1

(4 години)

Тема: Опис предметної області.

Мета: Навчитися проводити аналіз предметної області та оформлювати результати.

Хід роботи:

І. Теоретичні відомості

Інформаційна система – система, що реалізує автоматизований збір, обробку та маніпулювання даними та включає технічні засоби обробки даних, програмне забезпечення і відповідний персонал.

Мета будь-якої інформаційної системи – обробка даних про об'єкти реального світу. Основою інформаційної системи є база даних. В широкому розумінні слова база даних – це сукупність відомостей про конкретні обєкти реального світу в будь-якій предметній області.

Під предметною областю прийнято розуміти частину реального світу, що підлягає вивченню для організіції керування і в кінцевому рахунку автоматизації.

Створюючи інформаційн систему, користувач прагне впорядкувати інформацію за різними ознаками і швидко робити вибірку з довільним поєднанням ознак. Велике значення при цьому набуває структурування даних.

Структурування даних - це введення угод про способи представлення даних. Неструктурованими називають дані, записані, наприклад, в текстовому файлі.

ІІ Завдання до практичної роботи №1

1. Обрати предметну область для виконання практичних робіт з дисципліни конструювання програмного забезпечення;
2. Дослідити предметну область. Проаналізувати актуальність створення ІС за обраною предметною областю.
3. Зробити опис обраної предметної області.

ІІІ Зробити висновки по виконанню практичної роботи №1

ІV Контрольні питання для підготовки к практичній роботі №1

1. Що таке предметна область?
2. Що таке інформаційна система?

Практична робота №2

(4 години)

Тема: Побудова діаграми Use Case.

Мета: Отримати практичні навики побудови діаграм Use Case.

Хід роботи:

І. Теоретичні відомості

Варіант використання

Конструкція або стандартний елемент мови UML *варіант використання* застосовується для специфікації загальних особливостей поведінки системи або будь-якої інший сутності предметної області без розгляду внутрішньої структури цієї сутності. Кожний варіант використання визначає послідовність дій, які повинні бути виконані проектованою системою при взаємодії її з відповідним актором. Діаграма варіантів може доповнюватися пояснювальним текстом, який розкриває зміст або семантику складових її компонентів. Такий пояснювальний текст одержав назву примітки або сценарію.

Окремий варіант використання позначається на діаграмі еліпсом, усередині якого втримується його коротка назва або ім'я у формі дієслова з пояснювальними словами (рис. 1).

Рисунок 1 - Графічне позначення варіанта використання

Ціль варіанта використання полягає в тому, щоб визначити закінчений аспект або фрагмент поведінки деякої сутності без розкриття внутрішньої структури цієї сутності. У якості такої сутності може виступати вихідна система або будь-який інший елемент моделі, який має власну поведінку, подібно підсистемі або класу в моделі системи.

Кожний варіант використання відповідає окремому сервісу, який надає моделируемую сутність або систему по запиту користувача (актора), тобто

визначає спосіб застосування цієї сутності. Сервіс, який инициализируется по запиту користувача, являє собою закінчену послідовність дій. Це означає, що після того як система закінчить обробку запиту користувача, вона повинна вернутися у вихідний стан, у якому готова до виконання наступних запитів.

Варіанти використання описують не тільки взаємодії між користувачами й сутністю, але також реакції сутності на одержання окремих повідомлень від користувачів і сприйняття цих повідомлень за межами сутності. Варіанти використання можуть містити в собі опис особливостей способів реалізації сервісу й різних виняткових ситуацій, таких як коректна обробка помилок системи. Безліч варіантів використання в цілому повинне визначати всі можливі сторони очікуваного поведінки системи. Для зручності безліч варіантів використання може розглядатися як окремий пакет.

Із системно-аналітичної точки зору варіанти використання можуть застосовуватися як для специфікації зовнішніх вимог до проектованої системи, так і для специфікації функціональної поведінки вже існуючої системи.

Актори

Актор являє собою будь-яку зовнішню стосовно моделируемой системі сутність, яка взаємодіє із системою й використовує її функціональні можливості для досягнення певних цілей або розв'язку приватних завдань. При цьому актори служать для позначення погодженого безлічі ролей, які можуть відіграти користувачі в процесі взаємодії із проектованою системою. Кожний актор може розглядатися як якась окрема роль щодо конкретного варіанта використання. Стандартним графічним позначенням актора на діаграмах є фігурка "чоловічка", під якою записується конкретне ім'я актора (рис. 2).

Рисунок 2 - Графічне позначення актора

У деяких випадках актор може позначатися у вигляді прямокутника класу із ключовим словом "актор" і звичайними складовими елементами класу.

Актори використовуються для моделювання зовнішніх стосовно проектованої системи сутностей, які взаємодіють із системою й використовують її

як окремі користувачів. У якості акторів можуть виступати інші системи, підсистеми проектованої системи або окремі класи. Важливо розуміти, що кожний актор визначає деяку погоджену безліч ролей, у яких можуть виступати користувачі даної системи в процесі взаємодії з нею. У кожний момент часу із системою взаємодіє цілком певний користувач, при цьому він відіграє або виступає в одній з таких ролей. Найбільш наочний приклад актора - конкретний користувач системи зі своїми власними параметрами аутентификации.

Будь-яка сутність, яка узгодиться з подібним неформальним визначенням актора, являє собою екземпляр або приклад актора. Для моделируемой системи акторами можуть бути як суб'єкти-користувачі, так і інші системи. Оскільки користувачі системи завжди є зовнішніми стосовно цієї системи, то вони завжди представляються у вигляді акторів.

Відносини на діаграмі варіантів використання

Між компонентами діаграми варіантів використання можуть існувати різні відносини, які описують взаємодія екземплярів одних акторів і варіантів використання з екземплярами інших акторів і варіантів. Один актор може взаємодіяти з декількома варіантами використання. У цьому випадку цей актор звертається до декільком сервісам даної системи. У свою чергу один варіант використання може взаємодіяти з декількома акторами, надаючи для всіх їх свій сервіс. Слід помітити, що два варіанти використання, певні для однієї й тієї ж сутності, не можуть взаємодіяти один з одним, оскільки кожний з них самостійно описує закінчений варіант використання цієї сутності. Більше того, варіанти використання завжди передбачають деякі сигнали або повідомлення, коли взаємодіють із акторами за межами системи. У той же час можуть бути визначені інші способи для взаємодії з елементами усередині системи.

У мові UML є кілька стандартних видів відносин між акторами й варіантами використання:

- Відношення асоціації (association relationship)
- Відношення розширення (extend relationship)
- Відношення узагальнення (generalization relationship)
- Відношення включення (include relationship)

При цьому загальні властивості варіантів використання можуть бути представлено трьома різними способами, а саме за допомогою відносин розширення, узагальнення й включення.

Відношення асоціації

Відношення асоціації є одним з фундаментальних понять у мові UML і тією чи іншою мірою використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Стосовно до діаграм варіантів використання воно служить для позначення специфічної ролі актора в окремому варіанті використання. Інакше кажучи, асоціація специфицирует семантичні особливості взаємодії акторів і варіантів використання в графічній моделі системи. Таким чином, це відношення встановлює, яку конкретну роль відіграє актор при взаємодії з екземпляром варіанта використання. На діаграмі варіантів використання, так само як і на інших діаграмах, відношення асоціації позначається суцільною лінією між актором і варіантом використання. Ця лінія може мати додаткові умовні позначки, такі, наприклад, як ім'я й кратність (рис. 3).

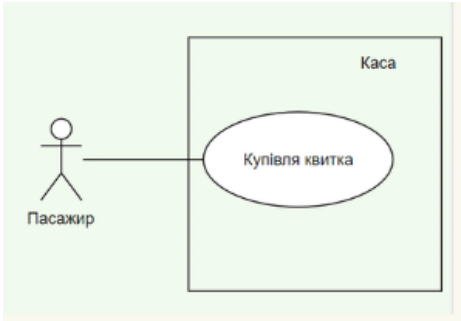


Рисунок 3 - Приклад графічної вистави відносини асоціації між актором і варіантом використання

Відношення розширення

Відношення розширення визначає взаємозв'язок екземплярів окремого варіанта використання з більш загальним варіантом, властивості якого визначаються на основі способу спільного об'єднання даних екземплярів. У метамодели відношення розширення є спрямованим і вказує, що стосовно до окремих прикладів деякого варіанта використання повинні бути виконані конкретні умови, певні для розширення даного варіанта використання. Так, якщо має місце

відношення розширення від варіанта використання А до варіанта використання В, те це означає, що властивості екземпляра варіанта використання В можуть бути доповнені завдяки наявності властивостей у розширеного варіанта використання А.

Відношення розширення між варіантами використання позначається пунктирною лінією зі стрілкою (варіант відносини залежності), спрямованої від того варіанта використання, який є розширенням для вихідного варіанта використання. Дана лінія зі стрілкою позначається ключовим словом "extend" ("розширює"), як показано на рис. 4.

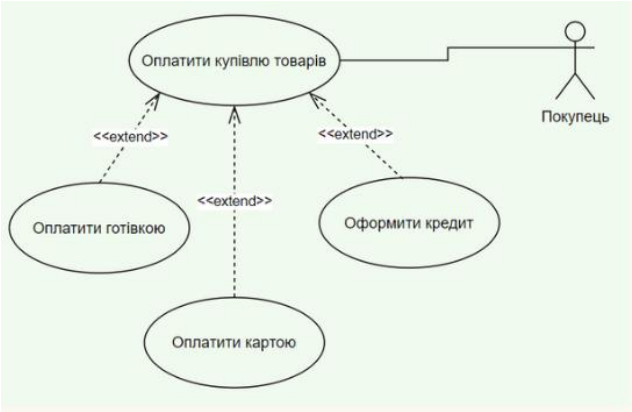


Рисунок 4 - Приклад графічного зображення відносини розширення між варіантами використання

Відношення розширення відзначає той факт, що один з варіантів використання може приєднувати до своєї поведінки деяка додаткова поведінка, певне для іншого варіанта використання. Дане відношення містить у собі деяка умова й посилання на крапки розширення в базовому варіанті використання. Щоб розширення мало місце, повинне бути виконана певна умова даного відношення. Посилання на крапки розширення визначають ті місця в базовому варіанті використання, у які повинне бути поміщене відповідне розширення при виконанні умови.

Відношення узагальнення

Відношення узагальнення служить для вказівки того факту, що деякий варіант використання А може бути узагальнений до варіанта використання В. У цьому випадку варіант А буде спеціалізацією варіанта В. При цьому В називається предком або батьком по відношенню А, а варіант А - нащадком стосовно варіанта використання В. Слід підкреслити, що нащадок успадковує всі властивості й

поведінка свого батька, а також може бути доповнений новими властивостями й особливостями поведінки. Графічно дане відношення позначається суцільною лінією зі стрілкою у формі незафарбованого трикутника, яка вказує на батьківський варіант використання (рис. 5). Ця лінія зі стрілкою має спеціальна назва - стрілка "узагальнення".

Рисунок 3.5 - Приклад графічного зображення відносини узагальнення між варіантами використання

Відношення узагальнення між варіантами використання застосовується в тому випадку, коли необхідно відзначити, що дочірні варіанти використання мають усі атрибути й особливостями поведінки батьківських варіантів. При цьому дочірні варіанти використання беруть участь у всіх відносинах батьківських варіантів. У свою чергу, дочірні варіанти можуть наділятися новими властивостями поведінки, які відсутні в батьківських варіантів використання, а також уточнювати або модифікувати наслідувані від них властивості поведінки.

Відношення включення

Відношення включення між двома варіантами використання вказує, що деяка задана поведінка для одного варіанта використання включається як складеного компонента в послідовність поведінки іншого варіанта використання. Дане відношення є спрямованим бінарним відношенням у тому розумінні, що пари екземплярів варіантів використання завжди впорядкована відносно включення.

Семантика цього відношення визначається в такий спосіб. Коли екземпляр першого варіанта використання в процесі свого виконання досягає крапки включення в послідовність поведінки екземпляра другого варіанта використання,

екземпляр першого варіанта використання виконує послідовність дій, що визначає поведінка екземпляра другого варіанта використання, після чого продовжує виконання дій своєї поведінки. При цьому передбачається, що навіть якщо екземпляр першого варіанта використання може мати екземплярів, що трохи включаються в себе, інших варіантів, виконувані ними дії повинні закінчитися до деякого моменту, після чого повинне бути продовжене виконання перерваних дій екземпляра першого варіанта використання відповідно до заданого для нього поведінкою.

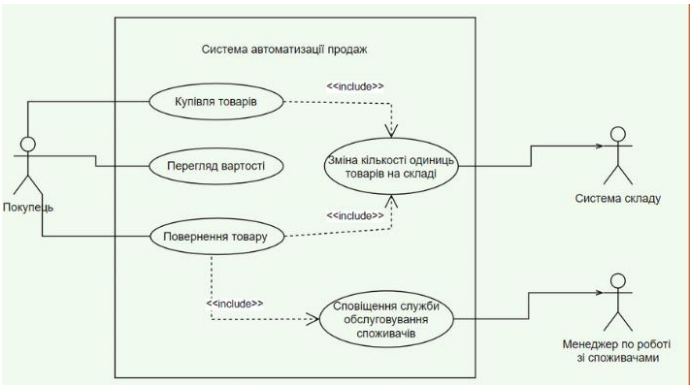


Рисунок 6 - Приклад графічного зображення відносини включення між варіантами використання

Приклад діаграми прецедентів.

Секретар розміщує на сервері меню обідніх страв на тиждень. Співробітники повинні мати можливість ознайомитися з меню і зробити замовлення, вибравши страви на кожен день наступного тижня. Офіс-менеджер повинен мати можливість сформувати рахунок і оплатити його. Система повинна бути написана на ASP.NET. Таке ось нехитрий інтернет-додаток для автоматизації замовлень обідів в офіс.

Прецедент	Дієва особа
Розташувати меню	Секретар
Ознайомитись з меню	Співробітник, секретар, офіс-менеджер
Зробити замовлення	Співробітник, секретар, офіс-менеджер
Сформувати рахунок	Офіс-менеджер
Оплатити рахунок	Офіс-менеджер

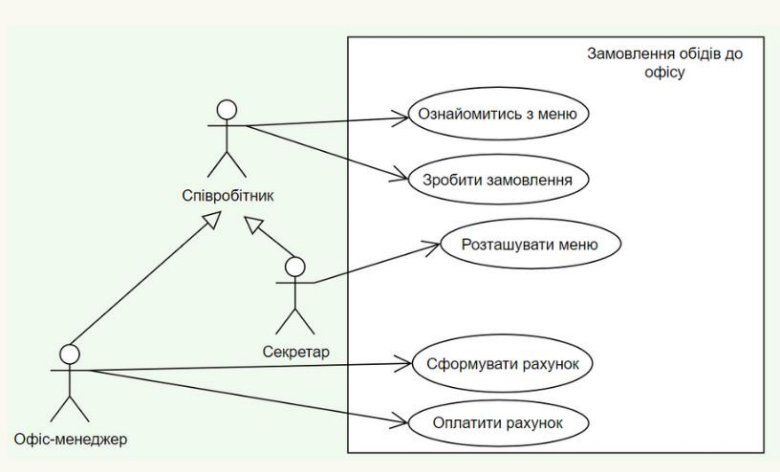


Рисунок 7 – Приклад діаграми прецедентів

II Завдання до практичної роботи №2

1. По власній предметній області побудуйте діаграму прецедентів (мінімум з точок зору 2-х акторів). При побудові діаграми намагайтеся використовувати всі види відносин.
2. Для отриманих прецедентів опишіть сценарії.

III Зробити висновки по виконанню практичної роботи №2

IV Контрольні питання для підготовки к практичній роботі №2

1. Які предмети UML зустрічаються на діаграмі прецедентів?
2. Які відносини дозволені між предметами на діаграмі Use Case?
3. Для чого застосовують діаграми Use Case?

Практична робота №3

(2 години)

Тема: Виділення класів.

Мета: Навчитися виділяти базові класи предметної області ґрунтуючись на принципах ООП.

Хід роботи:

I. Теоретичні відомості

Клас - опис безлічі об'єктів, які розділяють однакові властивості, операції, відносини й семантику (зміст). Будь-який об'єкт - просто екземпляр класу.

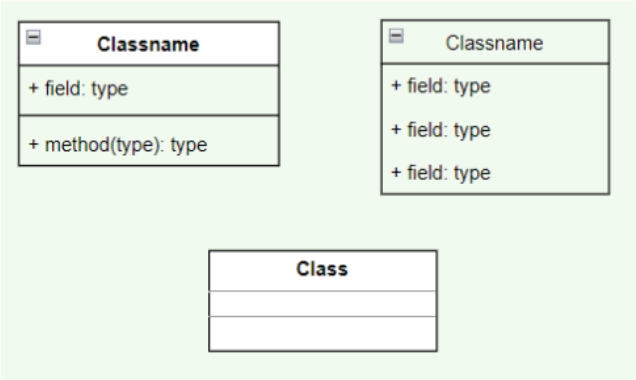


Рисунок 1 – Графічне представлення класу на діаграмі

Обов'язковим елементів позначення класу є його ім'я. На початкових етапах розробки діаграми окремі класи можуть позначатися простим прямокутником із вказівкою тільки імені відповідного класу. У міру проробляння окремих компонентів діаграми опису класів доповнюються атрибутами і операціями.

Навіть якщо секція атрибутів і операцій є порожній, у позначенні класу вона виділяється горизонтальною лінією, щоб відразу відрізнити клас від інших елементів мови UML.

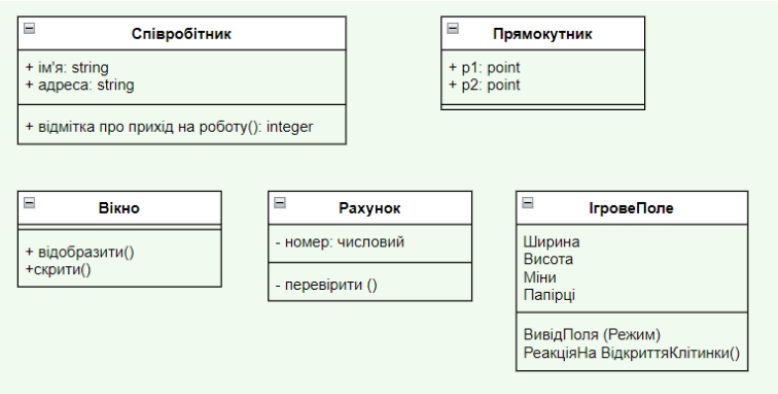


Рисунок 2 – Приклади графічного представлення класів на діаграмі

У другій зверху секції прямокутника класу записуються його *атрибути* (attributes) або *властивості*. У мові UML прийнята певна стандартизація запису атрибутів класу, яка підкоряється деяким синтаксичним правилам. Кожному атрибуту класу відповідає окремий рядок тексту, який складається із квантора видимості атрибута, імені атрибута, його кратності, типу значень атрибута й, можливо, його вихідного значення:

<квантор видимості><ім'я атрибута>[кратність]:

<тип атрибута> = <вихідне значення>{ рядок-властивість}

Квантор видимості може ухвалювати одне із трьох можливих значень і, відповідно, відображається за допомогою спеціальних символів:

• Символ "+" позначає атрибут з областю видимості типу *загальнодоступний* (public). Атрибут із цією областю видимості доступний або видний з будь-якого іншого класу пакета, у якому визначена діаграма.

• Символ "#" позначає атрибут з областю видимості типу *захищений* (protected). Атрибут із цією областю видимості недоступний або невидний для всіх класів, за винятком підкласів даного класу.

• Символ "-" позначає атрибут з областю видимості типу *закритий* (private). Атрибут із цією областю видимості недоступний або невидний для всіх класів без винятку.

Квантор видимості може бути опущений. У цьому випадку його відсутність проста означає, що видимість атрибута не вказується. Ця ситуація відрізняється від прийнятих за замовчуванням угод у традиційних мовах програмування, коли відсутність квантора видимості трактується як public або private. Однак замість умовних графічних позначень можна записувати відповідне ключове слово: public, protected, private.

Ім'я атрибута являє собою рядок тексту, який використовується в якості ідентифікатора відповідного атрибута й тому повинна бути унікальної в межах даного класу. Ім'я атрибута є єдиним обов'язковим елементом синтаксичного позначення атрибута.

Кратність атрибута характеризує загальна кількість конкретних атрибутів даного типу, що входять до складу окремого класу. У загальному випадку кратність записується у формі рядка тексту у квадратних дужках після імені відповідного атрибута:

[нижня_границя1 .. верхня_границя1, нижня_границя2.. верхня_грашица2, ..., нижняя_границаk .. верхня_границаk],

де нижня_границя й верхня_границя є позитивними цілими числами, кожна пара яких служить для позначення окремого замкненого інтервалу цілих чисел, у якого нижня (верхня) границя дорівнює значенню нижня_границя (верхня_границя). У цілому дана умовна позначка кратності відповідає теоретико-

множинному об'єднанню відповідних інтервалів. У якості верхньої_границі може використовуватися спеціальний символ "*", який означає довільне позитивне ціле число. Інакше кажучи, це означає необмежене зверху значення кратності відповідного атрибута.

Значення кратності з інтервалу випливають у монотонно зростаючому порядку без пропуску окремих чисел, що лежать між нижньої й верхньої границями. При цьому дотримуються наступного правила: відповідні нижні й верхні границі інтервалів включаються в значення кратності. Якщо в якості кратності вказується однина , то кратність атрибута ухвалюється рівної даному числу. Якщо ж вказується єдиний знак "*", то це означає, що кратність атрибута може бути довільним позитивним цілим числом або нулем.

Якщо кратність атрибута не зазначена, то за замовчуванням ухвалюється її значення рівне 1..1, тобто в точності 1.

Тип атрибута являє собою вираження, семантика якого визначається мовою специфікації відповідної до моделі. У нотації UML тип атрибута іноді визначається залежно від мови програмування, яка передбачається використовувати для реалізації даної моделі. У найпростішому випадку тип атрибута вказується рядком тексту, що має осмислене значення в межах пакета або моделі, до яких ставиться розглянутий клас.

Вихідне значення служить для завдання деякого початкового значення для відповідного атрибута в момент створення окремого екземпляра класу. Тут необхідно дотримуватися правила приналежності значення типу конкретного атрибута. Якщо вихідне значення не зазначене, то значення відповідного атрибута не визначене на момент створення нового екземпляра класу. З іншого боку, конструктор відповідного об'єкта може перевизначати вихідне значення в процесі виконання програми, якщо в цьому виникає необхідність.

У третій зверху секції прямокутника записуються *операції або методи класу*. Операція (operation) являє собою деякий сервіс, що надає кожний екземпляр класу на певному вимогу. Сукупність операцій характеризує функціональний аспект поведінки класу. Запис операцій класу в мові UML також стандартизована й підкоряється певним синтаксичним правилам. При цьому кожної операції класу

відповідає окремий рядок, який складається із квантора видимості операції, імені операції, вираження типу, що вертається операцією значення й, можливо, рядок-властивість даної операції:

<квантор видимості><ім'я операції>(список параметрів):

<вираження типу значення, що вертається, >{ рядок-властивість}

Квантор видимості, як і у випадку атрибутів класу, може ухвалювати одне із трьох можливих значень і, відповідно, відображається за допомогою спеціального символу. Символ "+" позначає операцію з областю видимості типу загальнодоступний (public). Символ "#" позначає операцію з областю видимості типу захищений (protected). І, нарешті, символ "-" використовується для позначення операції з областю видимості типу закритий (private).

Квантор видимості для операції може бути опущений.

Ім'я операції являє собою рядок тексту, який використовується в якості ідентифікатора відповідної до операції й тому повинна бути унікальної в межах даного класу. Ім'я атрибута є єдиним обов'язковим елементом синтаксичного позначення операції.

Список параметрів є переліком розділених комами формальних параметрів, кожний з яких може бути представлений у наступному виді:

<вид параметра><ім'я параметра>:<вираження типу>=<значення параметра за замовчуванням>.

Тут *вид параметра* - є одне із ключових слів in, out або inout зі значенням in за замовчуванням, у випадку якщо вид параметра не вказується. *Ім'я параметра* є ідентифікатор відповідного формального параметра. *Вираження типу* є залежною від конкретної мови програмування специфікацією типу значення, що вертається, для відповідного формального параметра. Нарешті, *значення за замовчуванням* у загальному випадку являє собою вираження для значення формального параметра, синтаксис якого залежить від конкретної мови програмування й підкоряється прийнятим у ньому обмеженням.

Вираження типу значення, що вертається, також є залежною від мови реалізації специфікацією типу або типів значень параметрів, які вертаються об'єктом після виконання відповідної операції.

Рядок-Властивість служить для вказівки значень властивостей, які можуть бути застосовані до даного елемента. Рядок-Властивість не є обов'язкової, вона може отсутствовать, якщо ніякі властивості не специфіковані.

II Завдання до практичної роботи №3

1. По вибраній предметній області виділити базові класи (не менше 10).
2. Для кожного класу виділити атрибути і записати в повній формі.
3. Для кожного класу виділити операції і представити в загальній формі.

III Зробити висновки по виконанню практичної роботи №3

IV Контрольні питання для підготовки к практичній роботі №3

1. Що таке клас?
2. Що таке атрибут класу?
3. Що таке квантор видимості?
4. Які значення може приймати кратність атрибута?
5. Що таке тип атрибуту?
6. Що таке вихідне значення?
7. Що таке операція класу?
8. Як записуються операції?
9. Як записується параметр операції?

Практична робота №4

(2 години)

Тема: Асоціація між класами. Агрегація. Композиція.

Мета: отримати практичні навики встановлення відношення типу асоціація між класами системи ґрунтуючись на принципах ООП.

Хід роботи:

I. Теоретичні відомості

Відношення асоціації відповідає наявності деякого відношення між класами. Дане відношення позначається суцільною лінією з додатковими спеціальними символами, які характеризують окремі властивості конкретної асоціації. У якості додаткових спеціальних символів можуть використовуватися ім'я асоціації, а також

імена й кратність класів-ролей асоціації. Ім'я асоціації є необов'язковим елементом її позначення.

Найбільш простий випадок даного відношення - *бінарна асоціація*. Вона зв'язує в точності два класи й, як виключення, може зв'язувати клас із самим собою. Для бінарної асоціації на діаграмі може бути зазначений порядок проходження класів з використанням трикутника у формі стрілки поруч із іменем даної асоціації. Напрямок цієї стрілки вказує на порядок класів, один з яких є першим (з боку трикутника), а іншої - другим (з боку вершини трикутника). Відсутність даної стрілки поруч із іменем асоціації означає, що порядок проходження класів у розглянутому відношенні не визначений.

У якості простого прикладу відносини бінарної асоціації розглянемо відношення між двома класами - класом "Компанія" і класом "Співробітник" (рис. 1). Вони зв'язані між собою бінарною асоціацією Робота, ім'я якої зазначено на малюнку поруч із лінією асоціації. Для даного відношення визначений порядок проходження класів, першим з яких є клас "Співробітник", а другим - клас "Компанія".

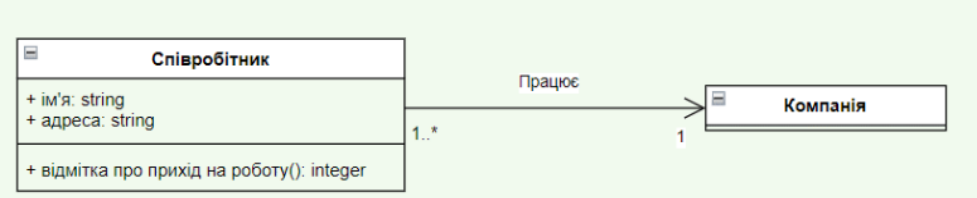


Рисунок 1 - Графічне зображення відносини бінарної асоціації між класами

Тернарна асоціація й асоціації більш високої арности в загальному випадку називаються *N- Арной асоціацією* (читається - "эн арная асоціація"). Така асоціація зв'язує деяким відношенням 3 і більш класів, при цьому один клас може брати участь в асоціації більш ніж один раз. Клас асоціації має певну роль у відповідному відношенні, що може бути явно зазначене на діаграмі. Кожний екземпляр N- Арной асоціації являє собою N- Арний кортеж значень об'єктів з відповідних класів. Бінарна асоціація є часним випадком N- Арной асоціації, коли значення N=2, і має своє власне позначення.

N-Арная асоціація графічно позначається ромбом, від якого ведуть лінії до символів класів даної асоціації. У цьому випадку ромб з'єднується із символами

відповідних класів суцільними лініями. Звичайно лінії проводяться від вершин ромба або від середини його сторін. Ім'я N- Арної асоціації записується поруч із ромбом відповідної асоціації.

Порядок класів в N- Арнії асоціації на діаграмі не фіксується. Деякий клас може бути приєднаний до ромба пунктирною лінією. Це означає, що даний клас забезпечує підтримку властивостей відповідної N-Арної асоціації, а сама N-Арна асоціація має атрибути, операції й/або асоціації. Інакше кажучи, така асоціація, у свою чергу, є класом з відповідним позначенням у вигляді прямокутника і є самостійним елементом мови UML - асоціацією-класом (Association Class). N-Арная асоціація не може містити символ агрегації ні для якої зі своїх ролей.

Як приклад конкретної тернарной асоціації розглянемо відношення між трьома класами: "Футбольна команда", "Рік" і "Гра". Дана асоціація вказує на наявність відносини між цими трьома класами, яке може представляти інформацію про ігри футбольних команд у національному чемпіонаті протягом декількох останніх років (рис. 2).

Як уже згадувалося, окремий клас асоціації має власну роль у відношенні. Ця роль може бути зображена графічно на діаграмі класів.

Рисунок 2 - Графічне зображення тернарной асоціації між трьома класами

Одним з додаткових позначень є *ім'я ролі* окремого класу, що входить в асоціацію. Ім'я ролі являє собою рядок тексту поруч із кінцем асоціації для відповідного класу. Вона вказує специфічну роль, яку відіграє клас. Ім'я ролі не є обов'язковим елементом позначень і може отсутствовать на діаграмі.

Кратність класу позначається у вигляді інтервалу цілих чисел, аналогічно кратності атрибутів і операцій класів. Інтервал записується поруч із кінцем асоціації й для N-Арной асоціації означає потенційне число окремих екземплярів.

Відношення агрегації має місце між декількома класами в тому випадку, якщо один із класів являє собою деяку сутність, що включає в себе як складені частини інші сутності.

Дане відношення має фундаментальне значення для опису структури складних систем, оскільки застосовується для вистави системних взаємозв'язків типу "частина-ціле". Розкриваючи внутрішню структуру системи, відношення агрегації показує, з яких компонентів складається система і як вони зв'язані між собою. З погляду моделі окремі частини системи можуть виступати як у вигляді елементів, так і у вигляді підсистем, які, у свою чергу, теж можуть утворювати складені компоненти або підсистеми. Це відношення по своїй суті описує декомпозицію або розбивку складної системи на більш прості складові частини, які також можуть бути піддані декомпозиції, якщо в цьому виникне необхідність надалі .

Очевидно, що розглянуте в такому аспекті розподіл системи на складові частини являє собою деяку ієрархію її компонентів, однак дана ієрархія принципово відрізняється від ієрархії, породжуваної відношенням узагальнення. Відмінність полягає в тому, що частини системи ніяк не зобов'язано успадковувати її властивості й поведінка, оскільки є цілком самостійними сутностями. Більше того, частини цілого мають свої власні атрибути й операціями, які суттєво відрізняються від атрибутів і операцій цілого.

Як приклад відносини агрегації розглянемо взаємозв'язок типу " частина-ціле", яка має місце між сутністю "Вантажний автомобіль" і такими компонентами, як "Двигун", "Шасі", "Кабіна", "Кузов". Не претендуючи на точну відповідність термінології даної предметної області, неважко уявити собі, що вантажний автомобіль складається із двигуна, шасі, кабіни й кузова. Саме це відношення між класом "Вантажний_автомобіль" і класами "Двигун", "Шасі", "Кабіна", "Кузов" описує відношення агрегації.

Графічно відношення агрегації зображується суцільною лінією, один з кінців якої являє собою не зафарбований усередині ромб. Цей ромб указує на той із класів, який являє собою "ціле". Інші класи є його "частинами".

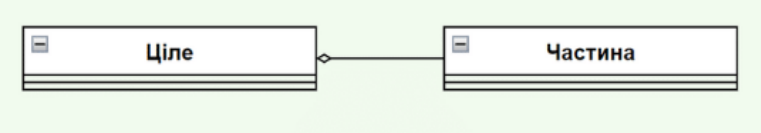


Рисунок 3 – Графічне представлення та приклад агрегації (за посиланням)

Відношення композиції, як уже згадувалося раніше, є частним випадком відносини агрегації. Це відношення служить для виділення спеціальної форми відносини "частина-ціле", при якій складові частини в деякому змісті перебувають усередині цілого. Специфіка взаємозв'язку між ними полягає в тому, що частини не можуть виступати у відриві від цілого, тобто зі знищенням цілого знищуються й усі його складові частини.

Приклад - вікно інтерфейсу програми, яке може складатися з рядка заголовка, кнопок керування розміром, смуг прокручування, головного меню, робочої області й рядка стану. Неважко зрозуміти, що подібне вікно являє собою клас, а його компоненти є як класами, так і атрибутами або властивостями вікна. Остання обставина досить характерна для відношення композиції, оскільки відбиває різні способи вистави даного відношення.

Графічно відношення композиції зображується суцільною лінією, один з кінців якої являє собою зафарбований усередині ромб. Цей ромб указує на той із класів, який являє собою клас-композицію або "ціле". Інші класи є його "частинами".

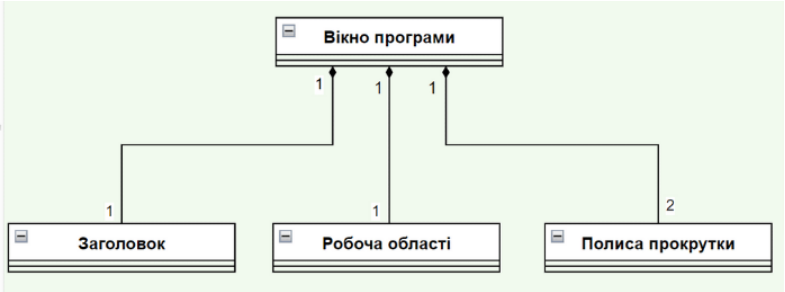


Рисунок 4 - Графічне представлення та приклад композиції (за величеною)

II Завдання до практичної роботи №4

1. Використовуючи відношення асоціації створити пари класів (класи з практичної роботи №3). Для кожної асоціації вказати символ порядку класів в асоціації, кратність асоціації та ім’я асоціації.
1. Зв’язати класи використовуючи N-арну асоціацію. Для кожного класу вказати кратність.
2. Використовуючи класи з практичної роботи №3 побудуйте для них агрегацію по посиланню та по величині.

III Зробити висновки по виконанню практичної роботи №4

IV Контрольні питання для підготовки к практичній роботі №4

1. Що таке асоціація?
2. Як вказати навігацію в асоціації?
3. Для чого використовується кратність?
4. Що таке N-арна асоціація?
5. Що таке агрегація?
6. В чому різниця між агрегацією та композицією?
7. Як графічно можна представити агрегацію по величині?

Практична робота №5

(2 години)

Тема: Побудова ієрархії простого спадкування.

Мета: отримати практичні навики встановлення відношення типу узагальнення між класами системи ґрунтуючись на принципах ООП.

Хід роботи:

І. Теоретичні відомості

Відношення узагальнення є звичайним таксономическим відношенням між більш загальним елементом (батьком або предком) і більш приватним або спеціальним елементом (дочірнім або нащадком). Дане відношення може використовуватися для вистави взаємозв'язків між пакетами, класами, варіантами використання й іншими елементами мови UML.

Стосовно до діаграми класів дане відношення описує ієрархічна будова класів і спадкування їх властивостей і поведінки. При цьому передбачається, що клас- нащадок має всі властивості й поведінкою класу-предка, а також має свої власні властивості й поведінка, які відсутні в класу-предка. На діаграмах відношення узагальнення позначається суцільною лінією із трикутною стрілкою на одному з кінців (мал. 1). Стрілка вказує на більш загальний клас (клас-предок або суперклас), а її відсутність - на більш спеціальний клас (клас-нащадок або підклас).

Рисунок 1 - Графічне зображення відносини узагальнення в мові UML

Як правило, на діаграмі може вказуватися кілька ліній для одного відношення узагальнення, що відбиває його таксономический характер. У цьому випадку більш загальний клас розбивається на підкласи одним відношенням Узагальнення. Наприклад, клас Геометрична_фігура_на_площині (курсив позначає абстрактний клас) може виступати в якості суперкласу для підкласів, що відповідають конкретним геометричним фігурам, таким як,Прямокутник, Окружність, Еліпс і ін. Даний факт може бути представлений графічно у формі діаграми класів наступного виду.

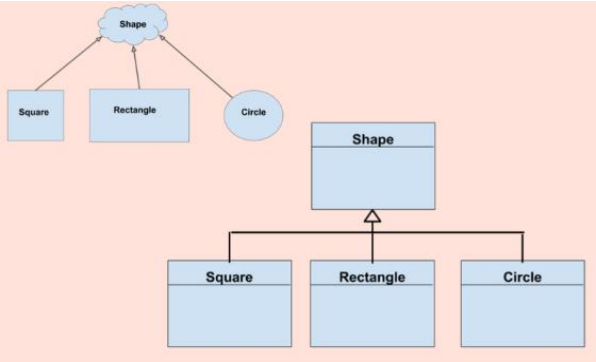


Рисунок 2 - Варіант графічного зображення відносини узагальнення класів для випадку об'єднання окремих ліній

Це позначення за формою відповідає графові спеціального виду, а саме - ієрархічному дереву. У цьому випадку клас-предок є коренем цього дерева, а класи-нащадки - його листами. Відмінність полягає в можливості вказівки на діаграмі класів потенційної можливості наявності інших класів-нащадків, які не включені в позначення представлених на діаграмі класів (три крапки замість прямокутника).

Поруч зі стрілкою узагальнення може розміщатися рядок тексту, що вказує на деякі додаткові властивості цього відношення. Даний текст буде ставитися до всіх ліній узагальнення, які йдуть до класів-нащадкам. Інакше кажучи, відзначена властивість стосується всіх підкласів даного відношення. При цьому текст слід розглядати як обмеження, і тоді він записується у фігурних дужках.

У якості обмежень можуть бути використані наступні ключові слова мови UML:

1. {complete} - означає, що в даному відношенні узагальнення специфіковані всі класи- нащадки, і інших класів- нащадків у даного класу- предка бути не може. Приклад - клас Клієнт_банку є предком для двох класів: Фізичне_особа й Компанія, і інших класів-нащадків він не має. На відповідній діаграмі класів це можна вказати явно, записавши поруч із лінією узагальнення даний рядок-обмеження;
2. {disjoint} - означає, що класи-нащадки не можуть містити об'єктів, що одночасно є екземплярами двох або більш класів. У наведеному вище прикладі ця умова також виконується, оскільки передбачається, що ніяка конкретна фізична особа не може бути одночасно й конкретною компанією. У

цьому випадку поруч із лінією узагальнення можна записати даний рядок-обмеження;

3. {incomplete} - означає випадок, протилежний першому. А саме, передбачається, що на діаграмі зазначені не всі нащадки. Надалі можливо заповнити їхній перелік не змінюючи вже побудовану діаграму. Приклад - діаграма класу "Автомобіль", для якої вказівка всіх без винятку моделей автомобілів порівнянне зі створенням відповідного каталогу. З іншого боку, для окремого завдання, такий як розробка системи продажу автомобілів конкретних моделей, у цьому немає необхідності. Але вказати неповноту структури класів- нащадків все-таки випливає;
4. {overlapping} - означає, що окремі екземпляри класів- нащадків можуть належати одночасно декільком класам. Приклад - клас "Багатокутник" є класом- предком для класу "Прямокутник" і класу "Ромб". Однак існує окремий клас "Квадрат", екземпляри якого одночасно є об'єктами перших двох класів. Цілком природно таку ситуацію вказати явно за допомогою даною рядка-обмеження.

З урахуванням можливості використання рядків-обмежень діаграма класів (мал. 2) може бути зображена без три крапки й без втрати інформації (мал. 3).

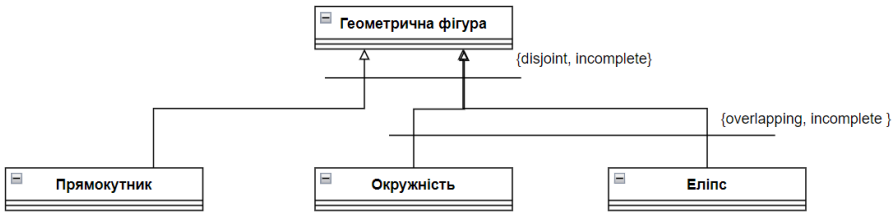


Рисунок 3 - Варіант графічного зображення відносини узагальнення класів з використанням рядка-обмеження

II Завдання до практичної роботи №5

1. Використовуючи відношення узагальнення побудуйте ієрархію спадкування для класів з практичної роботи №3.
2. Для даних відношень створіть рядки-обмеження.
3. Побудуйте ієрархію спадкування використовуючи множинне спадкування.

III Зробити висновки по виконанню практичної роботи №5

IV Контрольні питання для підготовки к практичній роботі №5

1. Що таке спадкування?
2. Що таке множинне спадкування?
3. Для чого використовують рядки-обмеження?
4. Які ключові слова можна використовувати в якості рядків-обмежень? Що вони означають?

Практична робота №6

(2 години)

Тема: Залежність між класами.

Мета: отримати практичні навики встановлення відношення типу залежність між класами системи ґрунтуючись на принципах ООП.

Хід роботи:

І. Теоретичні відомості

Відношення залежності в загальному випадку вказує деяке семантичне відношення між двома елементами моделі або двома множинами таких елементів, яке не є відношенням асоціації, узагальнення або реалізації. Воно стосується тільки самих елементів моделі й не вимагає безлічі окремих прикладів для пояснення свого змісту. Відношення залежності використовується в такій ситуації, коли деяка зміна одного елемента моделі може зажадати зміни іншого залежного від нього елемента моделі.

Відношення залежності графічно зображується пунктирною лінією між відповідними елементами зі стрілкою на одному з її кінців ("->" або "<-"). На діаграмі класів дане відношення зв'язує окремі класи між собою, при цьому стрілка спрямована від класу-клієнта залежності до незалежного класу або класу-джерелу (рис. 1). Наприклад, якщо функціонування Класу_А залежить від особливостей реалізації Класу_Б, то дана залежність може бути зображена в такий спосіб

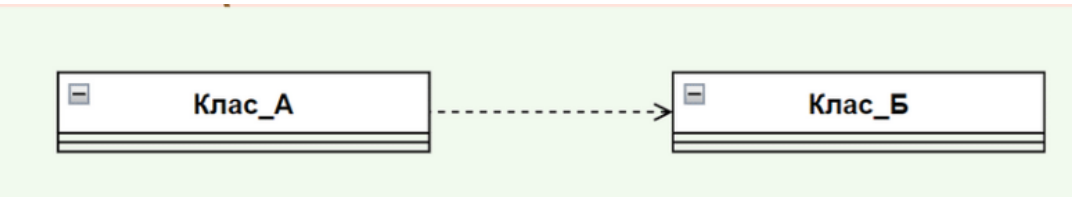


Рисунок 1 – Графічне зображення відношення залежності на діаграмі

У якості класу-клієнта й класу-джерела залежності можуть виступати цілі безлічі елементів моделі. У цьому випадку одна лінія зі стрілкою, що виходить від джерела залежності, розщеплюється в деякій крапці на кілька окремих ліній, кожна з яких має окрему стрілку для класу-клієнта. Наприклад, якщо функціонування Класу_С залежить від особливостей реалізації Класу_А и Класу_Б, то дана залежність може бути зображена в такий спосіб (рис. 2).

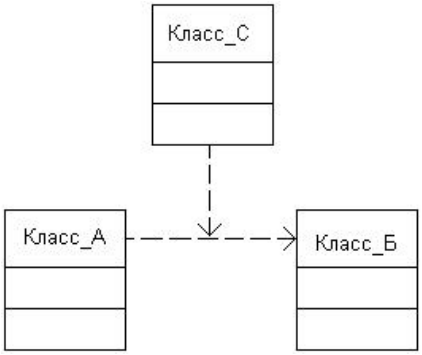


Рисунок 2 - Графічне представлення залежності між класом-клієнтом (Клас_С) і класами-джерелами (Клас_А и Клас_Б)

Стрілка може позначатися необов'язковим, але стандартним ключовим словом у лапках і необов'язковим індивідуальним іменем. Для відношення залежності визначені ключові слова, які позначають деякі спеціальні види залежностей. Ці ключові слова *(стереотипи)* записуються в лапках поруч зі стрілкою, яка відповідає даній залежності. Приклади стереотипів для відношення залежності представлені нижче:

• "access" - служить для позначення доступності відкритих атрибутів і операцій класу-джерела для класів-клієнтів;

• "bind" - клас-клієнт може використовувати деякий шаблон для своєї наступної параметризації;

• "derive" - атрибути класу-клієнта можуть бути обчислені по атрибутах класу-джерела;

• "import" - відкриті атрибути й операції класу-джерела стають частиною класу-клієнта, як якби вони були оголошені безпосередньо в ньому;

• та інші

II Завдання до практичної роботи №6

1. Використовуючи класи з практичної роботи №3 побудуйте бінарні пари зв'язані відношенням залежності. При необхідності вкажіть стереотип залежності.

III Зробити висновки по виконанню практичної роботи №6

IV Контрольні питання для підготовки к практичній роботі №6

1. Що таке залежність?
2. Що таке стереотип залежності?
3. Які стереотипи залежностей Вам відомі?

Практична робота №7

(2 години)

Тема: Побудова діаграми класів.

Мета: Отримати практичні навики побудови діаграми класів.

Хід роботи:

I. Теоретичні відомості

Діаграми класів (Class diagrams) – головний тип діаграм UML, які відображають логічну структуру програмної системи та суттєво впливають на процес генерації програмного коду. Основними елементами діаграми класів у Rational Rose є безпосередньо класи (classes) та відношення між ними (relations).

Клас – сукупність логічних об'єктів, які мають схожі характеристики і відрізняються однаковою поведінкою. В ООП характеристики об'єктів певного класу представлені сукупністю атрибутів, а поведінка – сукупністю операцій.

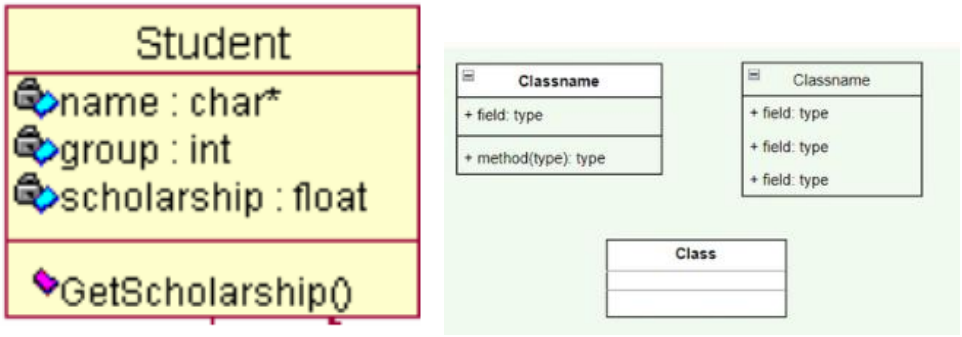


Рисунок 1 - Графічне представлення класу на діаграмі класів

Кожен клас графічно представлений прямокутником, що має три секції: ім'я класу, перелік атрибутів та перелік операцій (рис. 1). Для кожного атрибуту задається тип даних, для кожної операції тип даних для значення, що повертається, та перелік параметрів. Атрибути та операції можуть бути визначені для кожного об'єкта класу, чи для класу в цілому (static attributes and operations). Також для всіх атрибутів та операцій можна визначити тип видимості (public, protected, private).

Розрізняють декілька типів класів:

1. Конкретний клас – для даного класу можна створити об'єкт.

2. Абстрактний клас – клас, для якого не можна створити об'єкт. Даний клас виступає чистою абстракцією, від нього можна наслідувати конкретні класи.

3. Параметризований клас – клас, для визначення якого використовується список формальних параметрів, які впливають на атрибути та операції (аналог шаблона в C++). Найчастіше такі класи використовуються для створення абстрактних типів даних (списки, вектори, стеки, черги і т. п.).

4. Інтерфейс – клас, що містить тільки визначення набору операцій (без реалізації). У мові C++ аналогом даного класу є клас, всі операції якого є чистими віртуальними функціями. В мові Java класи можуть реалізовувати декілька інтерфейсів (інтерфейси використовуються для реалізації концепції множинного наслідування).

Відомо, що піктограма класу включає три секції (для імені, для властивостей і для операцій). Пустота секції не означає, що в класі відсутні властивості або операції, просто в цей момент вони не показуються. Можна явно визначити наявність у класі більшої кількості властивостей або атрибутів. Для цього наприкінці показаного списку проставляються три крапки. Як показано на рис. 2, у

довгих списках властивостей і операцій дозволяється угруповання - кожна група починається зі свого стереотипу.

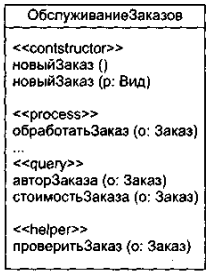


Рисунок 2 – Стереотипи для характеристик класу

Множинність

Іноді буває необхідно обмежити кількість екземплярів класу:

- задати нуль екземплярів (у цьому випадку клас перетворюється в утиліту, яка пропонує свої властивості й операції);
- задати один екземпляр (клас-singleton);
- задати конкретну кількість екземплярів;
- не обмежувати кількість екземплярів (це випадок, передбачуваний за замовчуванням).

Кількість екземплярів класу називається його множинністю. Вираження множинності записується в правому верхньому куті значка класу. Наприклад, як показано на рис. 3, *Контроллер углов* - це клас-singleton, а для класу *Датчик угла* дозволено три екземпляри.

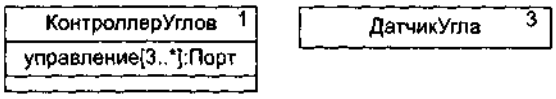


Рисунок 3 – Множинність

Відносини в діаграмах класів

Відносини, використовувані в діаграмах класів, показані на рис. 4.

Рисунок 4 – Відношення на діаграмах класів

Асоціації відображають структурні відносини між екземплярами класів, тобто з'єднання між об'єктами.

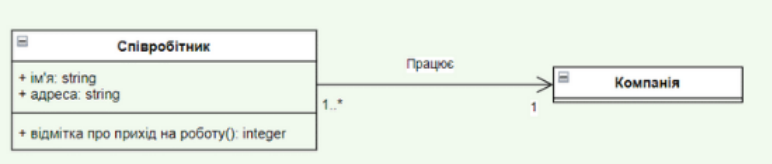


Рисунок 5 – Приклади асоціації

Агрегація показує відношення по посиланню (в агрегат включені тільки покажчики на частині), композиція - відношення фізичного включення (в агрегат включені самі частини).

Залежність є відношенням використання між клієнтом (залежним елементом) і постачальником (незалежним елементом). Звичайно операції клієнта:

- викликають операції постачальника;
- мають сигнатури, у яких значення, що вертається, або аргументи належать класу постачальника.

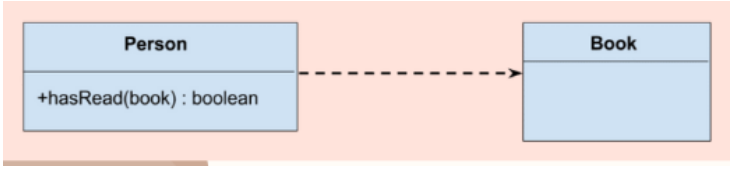


Рисунок 6 – Приклад залежності

Узагальнення - відношення між загальним предметом (суперкласом) і спеціалізованим різновидом цього предмета (підкласом). Підклас може мати одного батька (один суперклас) або кілька батьків (кілька суперкласів). У другому випадку говорять про множинне спадкування.

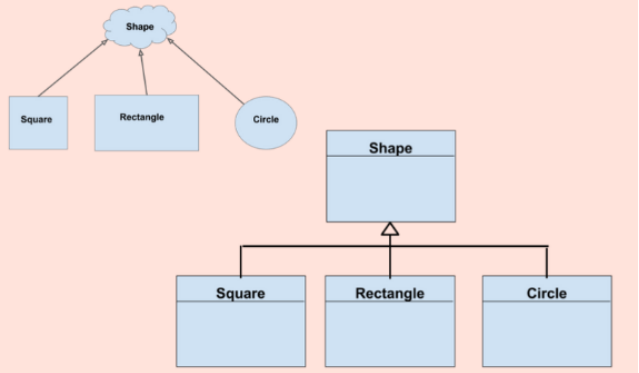


Рисунок 7 – Приклад множинного спадкування та ромбовидної решітки спадкування

Реалізація - семантичне відношення між класами, у якому клас-приймач виконує реалізацію операцій інтерфейсу класу-джерела. Наприклад, на мал. 8.6 показане, що клас Каталог повинен реалізувати інтерфейс Оброблювач каталогу, тобто Оброблювач каталогу розглядається як джерело, а Каталог - як приймач.

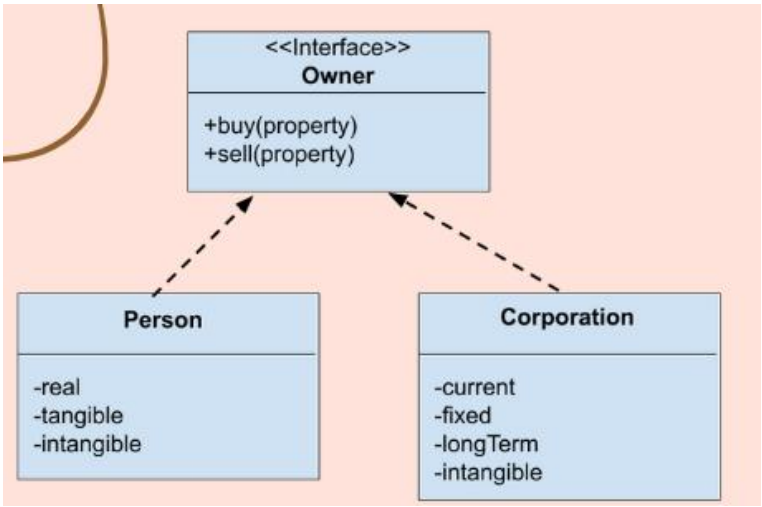


Рисунок 8 – Реалізація інтерфейсу

Інтерфейс Оброблювач каталогу дозволяє клієнтам взаємодіяти з об'єктами класу Каталог без знання тієї дисципліни доступу, яка тут реалізована.

Приклади діаграм класів:

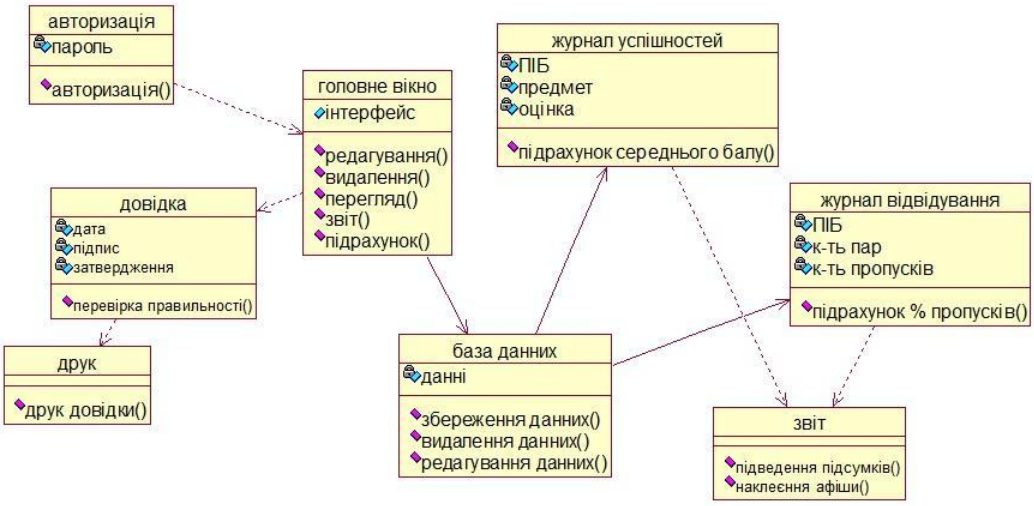


Рисунок 9 - Діаграма класів системи керування польотом

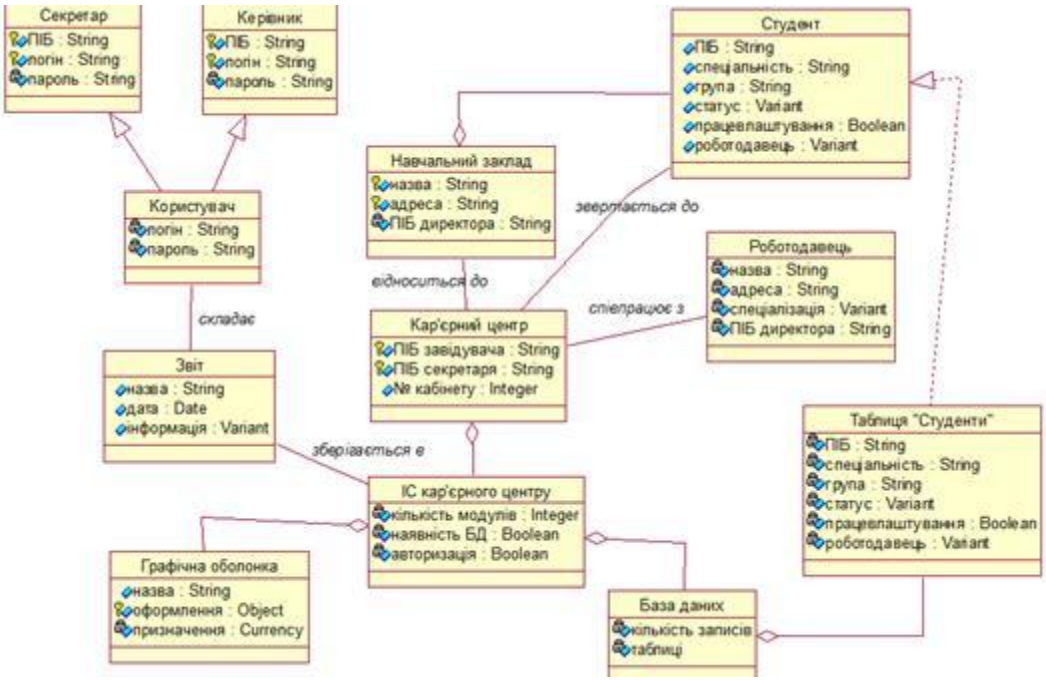


Рисунок 10 – Діаграма класів кар’єрного центру ЗВО

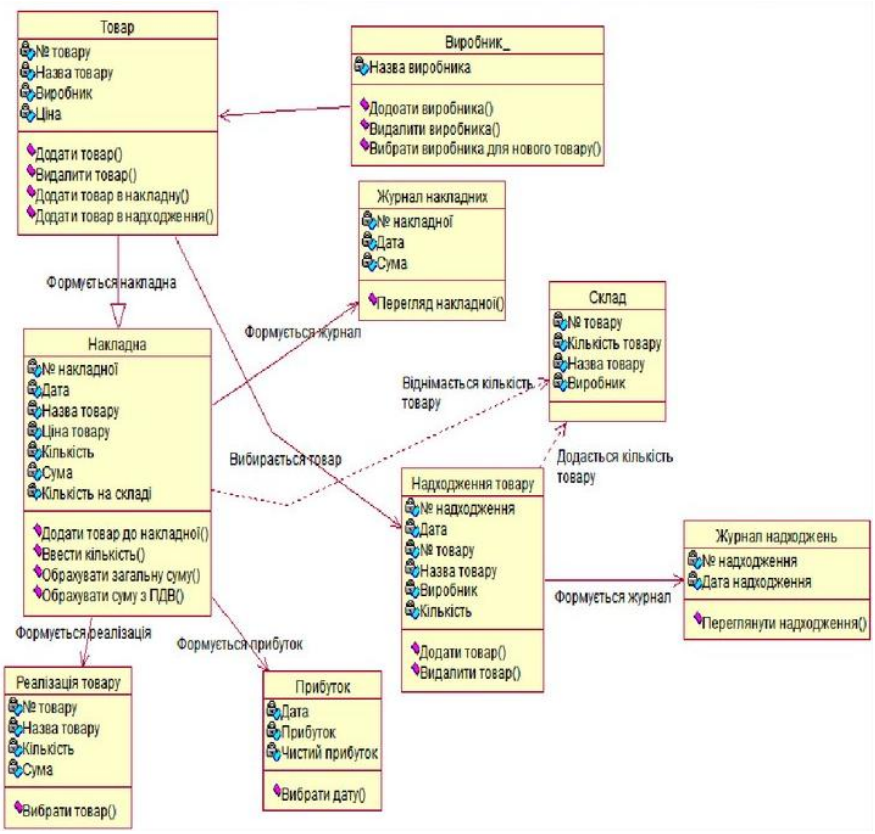


Рисунок 11 – Діаграма класів продажі товарів

II Завдання до практичної роботи №7

1. По своїй предметній області побудувати діаграму класів використовуючи різні види відносин з вказівкою ролей та кратності. При виконання практичної роботи можна опиратися на матеріал практичних робіт № 3-6.

III Зробити висновки по виконанню практичної роботи №7

IV Контрольні питання для підготовки к практичній роботі №7

1. Що таке клас?
2. Що є основним засобом для вистави статичних моделей?
3. Як використовуються статичні моделі?

Практична робота №8

(4 години)

Тема: Побудова діаграми співпраці.

Мета: Отримати практичні навики побудови діаграм співпраці.

Хід роботи:

І. Теоретичні відомості

Діаграми співробітництва

Діаграми співробітництва відображають взаємодія об'єктів у процесі функціонування системи. Такі діаграми моделюють сценарії поведінки системи. У російській літературі діаграми співробітництва часто називають діаграмами кооперації.

Об'єкти взаємодіють один з одним за допомогою *зв'язків* - каналів для передачі повідомлень. Зв'язок між парою об'єктів розглядається як екземпляр асоціації між їхніми класами. Іншими словами, зв'язок між двома об'єктами існує тільки тоді, коли є асоціація між їхніми класами. Неявно всі класи мають асоціацію самі із собою, отже, об'єкт може послати повідомлення самому собі.

Отже, зв'язок - це шлях для пересилання повідомлення. Шлях може бути постачений характеристикою видимості. Характеристика видимості проставляється як стандартний стереотип над далеким кінцем зв'язку. У мові передбачені наступні стандартні стереотипи видимості:

«global»	Об'єкт-Постачальник перебуває в глобальній області визначення
«local»	Об'єкт-Постачальник перебуває в локальній області визначення об'єкта-клієнта
«parameter»	Об'єкт-Постачальник є параметром операції об'єкта-клієнта
«self»	Той самий об'єкт є й клієнтом, і постачальником

Повідомлення - це специфікація передачі інформації між об'єктами чекаючи того, що буде забезпечена необхідна діяльність. Приймання повідомлення розглядається як подія.

Результатом обробки повідомлення звичайно є дія. У мові UML моделюються наступні різновиди дій:

Виклик	В об'єкті запускається операція
Повернення	Повернення значення в ухвалий об'єкт
Посилка(Send)	В об'єкт посилається сигнал
Створення	Створення об'єкта, виконується по стандартному повідомленню "create"
Знищення	Знищення об'єкта, виконується по стандартному повідомленню "destroy"

Для запису повідомлень у мові UML прийнятий наступний синтаксис:

ПовертаємаВеличина := Ім'яПовідомлення (Аргументи),

де ПовертаємаВеличина задає величину, що вертається як результат обробки повідомлення.

Приклади запису повідомлень:

Коорд:=ПоточнеПоложення(самолетТ1)	Виклик операції, повернення значення
Сповіщення( )	Відправка сигналу
ВстановитиМаршрут(х)	Виклик операції з дійсним параметром
«create»	Стандартне повідомлення для створення об'єкту

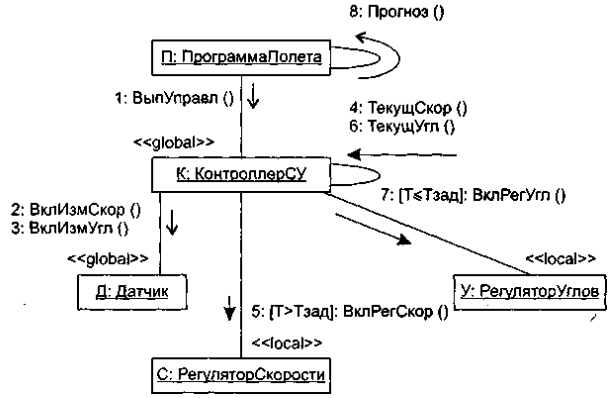


Рисунок 1 - Діаграма співробітництва системи керування польотом

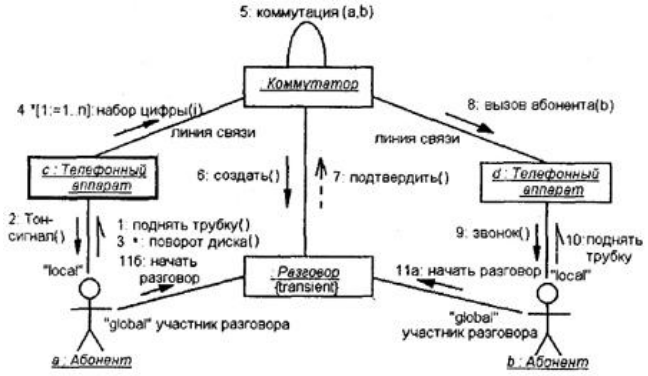


Рисунок 2 - Діаграма співробітництва процесу розмови по телефону

II Завдання до практичної роботи №8

1. По власній предметній області побудуйте діаграму співпраці.

При створені діаграм намагайтеся використовувати різні види повідомлень та розгалуження.

III Зробити висновки по виконанню практичної роботи №8

IV Контрольні питання для підготовки к практичній роботі №8

1. Чим відрізняється процедурний потік від асинхронного потоку повідомлень?
2. Як вказується повторення повідомлень?
3. Як показати розгалуження повідомлень?

Практична робота №9

(4 години)

Тема: Побудова діаграми послідовності.

Мета: Отримати практичні навики побудови діаграм послідовності.

I Теоретичні відомості

Діаграма послідовності - різновид діаграм взаємодії. Відбиваючи сценарій поведінки в системі, ця діаграма забезпечує більш наочна вистава порядку передачі повідомлень. Правда, вона не дозволяє показати такі деталі, які видні на діаграмі співробітництва (структурні характеристики об'єктів і зв'язків).

Графічно діаграма послідовності - різновид таблиці, який показує об'єкти, розміщені уздовж осі X, і повідомлення, упорядковані за часом уздовж осі Y.

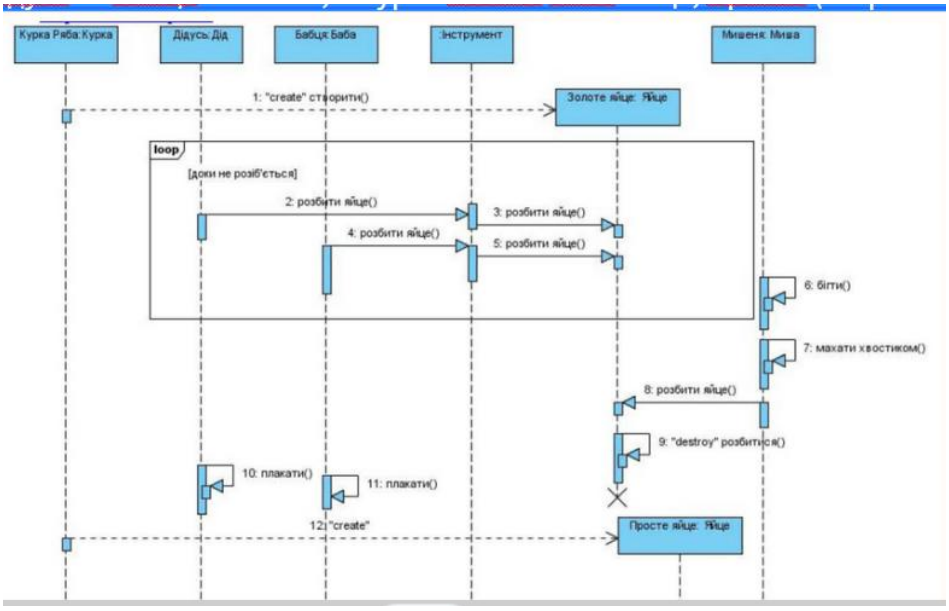


Рисунок 1 - Діаграма послідовності системи керування польотом

Як показано на мал. 1, об'єкти, що брати участь у взаємодії, містяться на вершині діаграми, уздовж осі X. Звичайно ліворуч розміщається об'єкт, що ініціює взаємодія, а праворуч - об'єкти по зростанню підпорядкованості. Повідомлення, що

посилають і прийняті об'єктами, містяться уздовж осі Y у порядку зростання часу від вершини до підстави діаграми. Використовуються ті ж синтаксис і позначення синхронізації, що й у діаграмах співробітництва. Таким чином, забезпечується проста візуальна вистава потоку керування в часі.

Від діаграм співробітництва діаграми послідовності відрізняють дві важливі характеристики.

Перша характеристика - *лінія життя об'єкта*.

Лінія життя об'єкта - це вертикальна пунктирна лінія, яка позначає період існування об'єкта. Більшість об'єктів існують протягом усього взаємодії, їх лінії життя тягнуться від вершини дощенту діаграми. Втім, об'єкти можуть створюватися в ході взаємодії. Їхні лінії життя починаються з моменту приймання повідомлення "create". Крім того, об'єкти можуть знищуватися в ході взаємодії. Їхні лінії життя закінчуються з моменту приймання повідомлення "destroy". Як презентовано на мал. 2, знищення лінії життя відзначається позначкою X наприкінці лінії:

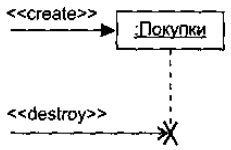


Рисунок 2 - Створення й знищення об'єкта

Друга характеристика - *фокус керування*.

Фокус керування - це високий тонкий прямокутник, що відображає період часу, протягом якого об'єкт виконує дія (свою або підлеглу процедуру). Вершина прямокутника відзначає початок дії, а підстава - його завершення. Момент завершення може маркіруватися повідомленням повернення, яке показується пунктирною стрілкою. Можна показати вкладення фокуса керування (наприклад, рекурсивний виклик власної операції). Для цього другий фокус керування рисується небагато правее першого (мал. 3).

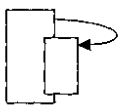


Рисунок 3 - Вкладення фокусів керування

Для відображення "умовності" лінія життя може бути розділена на кілька паралельних ліній життя. Кожна окрема лінія відповідає умовному розгалуженню у взаємодії. Далі в деякій крапці лінії життя можуть бути знову злиті (мал. 4).

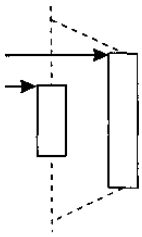


Рисунок 4 - Паралельні лінії життя

Розгалуження показується безліччю стрілок, що йдуть із однієї крапки. Кожна стрілка відзначається сторожовою умовою (мал. 5).

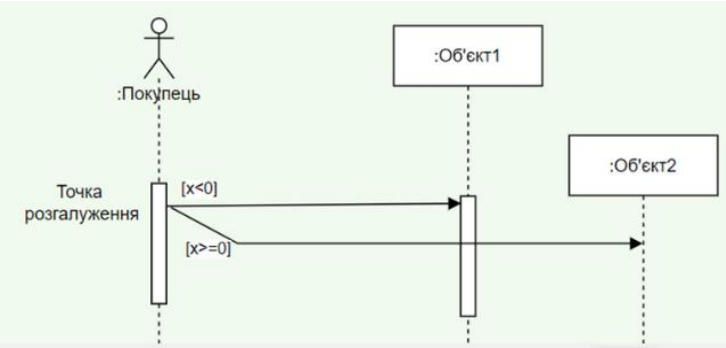


Рисунок 5 – Розгалуження

Приклад діаграми:

Курка Ряба знесла (стереотип <<create>>) Золоте яйце. Дідусь та Бабця намагались розбити яйце (циклічний фрагмент), проте їхні спроби не були вдалими. Бішло Мишеня (рекурсивне повідомлення Бігти()) , махало хвостиком (рекусивне повідомлення Михати хвостиком()) та розбило яйце (стериотип <<destroy>>). Дідусь та Бабця плакали, а Курка знесла нове Яйце, Просте (стереотип <<create>>).

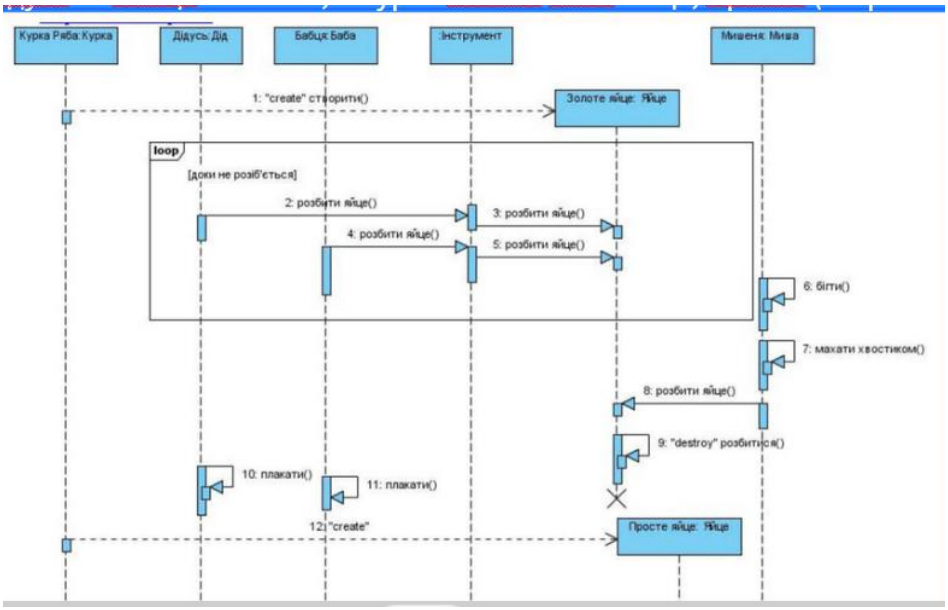


Рисунок 6 – приклад діаграми

II Завдання до практичної роботи №9

1. На основі побудованої діаграми співпраці створіть діаграму послідовностей.

При створені діаграм намагайтеся використовувати різні види повідомлень та розгалуження.

III Зробити висновки по виконанню практичної роботи №9

IV Контрольні питання для підготовки к практичній роботі №9

1. Чим відрізняється процедурний потік від асинхронного потоку повідомлень?
2. Як вказується повторення повідомлень?
3. Як показати розгалуження повідомлень?
4. Що загального в діаграмі послідовності й діаграмі співробітництва? Чим вони відрізняються друг від друга?

Практична робота №10

(4 години)

Тема: Побудова діаграми схем станів.

Мета: Отримати практичні навики побудови діаграми станів.

Хід роботи:

І. Теоретичні відомості

Для моделювання поведінки системи використовують:

- автомати;
- взаємодії.

Діаграми схем станів

Діаграма схем станів - одна з п'яти діаграм UML, що моделюють динаміку систем. Діаграма схем станів відображає кінцевий автомат, виділяючи потік керування, що випливає від стану до стану. *Кінцевий автомат* - поведінка, яка визначає послідовність станів у ході існування об'єкта. Ця послідовність розглядається як відповідь на події й включає реакції на ці події.

Діаграма схем станів показує:

1) набір станів системи;

2) події, які викликають перехід з одного стану в інше;

3) дії, які відбуваються в результаті зміни стану.

Головне призначення цієї діаграми - описати можливі послідовності станів і переходів, які в сукупності характеризують поведінка елемента моделі протягом його життєвого циклу. Діаграма станів представляє динамічну поведінку сутностей, на основі специфікації їх реакції на сприйняття деяких конкретних подій. Системи, які реагують на зовнішні дії від інших систем або від користувачів, іноді називають реактивними. Якщо такі дії ініціюються в довільні випадкові моменти часу, то говорять про асинхронну поведінку моделі.

Хоча діаграми станів найчастіше використовуються для опису поведінки окремих екземплярів класів (об'єктів), але вони також можуть бути застосовані для специфікації функціональності інших компонентів моделей, таких як варіанти використання, актори, підсистеми, операції й методи.

Діаграма станів по суті є графом спеціального виду, який представляє деякий автомат. Поняття автомата в контексті UML має досить специфічну семантику, засновану на теорії автоматів. Вершинами цього графа є стани й деякі інші типи елементів автомата (псевдо-стани), які зображуються відповідними графічними символами. Дуги графа служать для позначення переходів зі стану в стан. Діаграми станів можуть бути вкладені друг у друга, створюючи вкладені діаграми більш детального представлення окремих елементів моделі.

Стан

У мові UML під *станом* розуміється абстрактний мета клас, використовуваний для моделювання окремої ситуації, протягом якої має місце виконання деякої умови. Стан може бути заданий у вигляді набору конкретних значень атрибутів класу або об'єкта, при цьому зміна їх окремих значень буде відбивати зміну стану моделюємого класу або об'єкта.

Не кожний атрибут класу може характеризувати його стан. Як правило, мають значення тільки такі властивості елементів системи, які відбивають динамічний або функціональний аспект її поведінки. У цьому випадку стан буде характеризуватися деякою інваріантною умовою, що включають у себе тільки значимі для поведінки класу атрибути і їх значення.

Елемент переходить у стан в момент початку відповідної діяльності й залишає даний стан у момент її завершення.

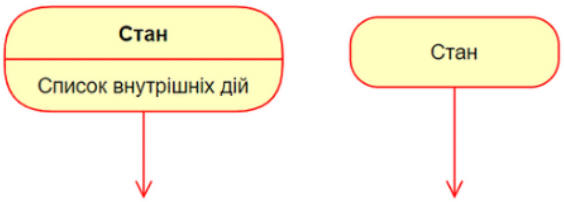


Рисунок 1 - Графічне зображення станів на діаграмі станів

Стан на діаграмі зображується прямокутником з округленими вершинами (рис. 1). Цей прямокутник, у свою чергу, може бути розділений на дві секції горизонтальною лінією. Якщо зазначена лише одна секція, то в ній записується тільки ім'я стану. А якщо ні, то в першій з них записується ім'я стану, а в другий - список деяких *внутрішніх дій* або *переходів* у даному стані. При цьому під *дією* в

мові UML розуміють деяку атомарну операцію, виконання якої приводить до зміни стану або поверненню деякого значення (наприклад, "істина" або "неправда").

Список внутрішніх дій

Ця секція містить перелік внутрішніх дій або *діяльностей*, які виконуються в процесі знаходження елемента в даному стані. Кожне з дій записується у вигляді окремого рядка й має наступний формат:

< мітка-дії '/' вираження - дії>

Мітка дії вказує на обставини або умови, при яких буде виконуватися діяльність, певна вираженням дії. При цьому *вираження дії* може використовувати будь-які атрибути й зв'язки, які належать області імен або контексту об'єкта.

Перелік міток дії має фіксовані значення в мові UML, які не можуть бути використані в якості імен подій. Ці значення наступні:

- entry - ця мітка вказує на дію, специфікована наступним за нею вираженням дії, яка виконується в момент входу в даний стан (вхідна дія);
- exit - ця мітка вказує на дію, специфікована наступним за нею вираженням дії, яка виконується в момент виходу з даного стану (вихідна дія);
- do - ця мітка специфицирует діяльність, що виконується ("do activity"), яка виконується протягом усього часу, поки об'єкт перебуває в даному стані, або доти , поки не закінчиться обчислення, специфіковане наступним за нею вираженням дії.

В останньому випадку при завершенні події генерується відповідний результат;

- include - ця мітка використовується для звертання до під-автомату, при цьому наступне за нею вираження дії містить ім'я цього під-автомату.

Як приклад стану розглянемо ситуацію введення пароля користувача при аутентифікації входу в деяку програмну систему. У цьому випадку список внутрішніх дій у даному стані не порожній і включає 4 окремих дії, перші два з яких стандартні й описані вище, а два останні визначаються своєю специфікацією.

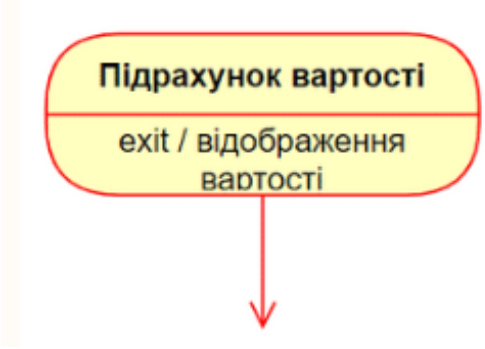


Рисунок 2 - Приклад стану з непустою секцією внутрішніх дій

Початковий стан

Початковий стан являє собою окремий випадок стану, який не містить ніяких внутрішніх дій (псевдостан). У цьому стані перебуває об'єкт за замовчуванням у початковий момент часу. Воно служить для вказівки на діаграмі станів графічної області, від якої починається процес зміни станів.

Рисунок 3 – Початковий та кінцевий стани

Кінцевий стан

Кінцевий (фінальне) стан являє собою окремий випадок стану, який також не містить ніяких внутрішніх дій (псевдостан). У цьому стані буде перебувати об'єкт за замовчуванням після завершення роботи автомата в кінцевий момент часу. Воно служить для вказівки на діаграмі станів графічної області, у якій завершується процес зміни станів або життєвий цикл даного об'єкта.

Перехід

Простий перехід (simple transition) являє собою відношення між двома послідовними станами, яке вказує на факт зміни одного стану іншим. Перебування об'єкта в першому стані може супроводжуватися виконанням деяких дій, а перехід у другий стан буде можливий після завершення цих дій, а також після задоволення деяких додаткових умов. У цьому випадку говорять, що перехід спрацьовує, Або відбувається спрацьовування переходу. До спрацьовування переходу об'єкт перебуває в попередньому від нього стані, називаним вихідним станом, або в джерелі, а після його спрацьовування об'єкт перебуває наступному від нього стані (цільовому стані).

Перехід здійснюється при настанні деякої події: закінчення виконання діяльності (do activity), одержанні об'єктом повідомлення або прийманням сигналу. Спрацьовування переходу може залежати не тільки від настання деякої події, але й від виконання певного умови, називаної *сторожовою умовою*. Об'єкт перейде з одного стану в інше в тому випадку, якщо відбулася зазначена подія й сторожова умова прийняла значення "істина".

На діаграмі станів перехід зображується суцільною лінією зі стрілкою, яка спрямована в цільовий стан. Кожний перехід може бути позначений рядком тексту, який має наступний загальний формат:

<сигнатура події>'['<сторожова умова>']' <вираження дії>.

При цьому сигнатура події описує деяка подія з необхідними аргументами:

<ім'я події>'('<список параметрів, розділених комами>')'.

Розглянемо приклад побудови схем станів на прикладі кофейного апарату:

Автомат Saeco Quarzo 500 призначений для установки в місцях з великою прохідністю: інститути, зали чекання, великі офісні й торгові центри й т.п..

Процес вибору напоїв гранично зрозумілий і простий. Кава готується під високим тиском.

Автомат дозволяє наготовлювати від 14 до 16 гарячих напоїв з натуральних кавових зерен, молока, шоколаду, сподіваючись і бульйону. Автомат готовить:

- кава эспрессо
- кава американо
- подвійний эспрессо
- капучино
- кава латте
- мокачино
- кава з молоком
- кава із шоколадом
- гарячий шоколад
- шоколадне молоко
- подвійний шоколад
- гаряче молоко

- молочний шоколад

- чай

- бульйон

Автомат Saeco Quarzo 500 автоматично видає склянку, цукор і ложечку. За допомогою панелі керування можна встановити режим готування напою з декофенированного кава (якщо в один з контейнерів засипаний розчинний кава). Є функції скасування подачі стаканчика, регулювання цукру на напій.

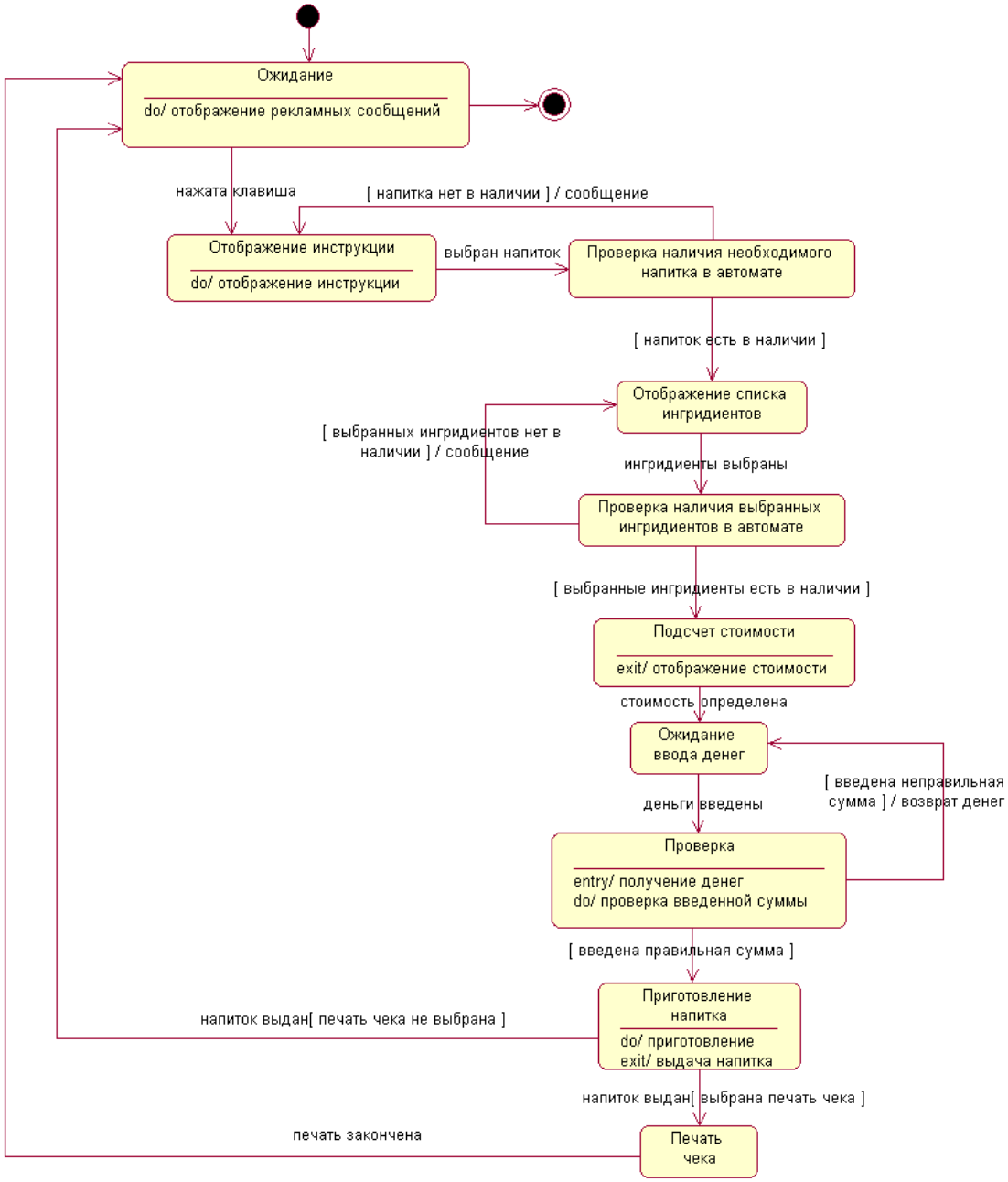
Меню автомата повністю русифіковане. Усі настроювання апарата можна задавати с зовнішньої панелі керування (установлюється витрата продуктів на порцію, температура обробки, режим самоочистки, знімаються показники лічильників приготовлених порцій, виторгу і т.д.).

Автомат обладнаний платіжними системами:

- монетоприемником з видачею здачі

- банкнотоприемником зі стекером

Робота з автоматом відбувається в такий спосіб. Автомат за замовчуванням перебуває в режимі очікування й відображає на екрані рекламну інформацію. Людей, що бажає одержати напій, натискає будь-яку клавішу. Автомат переходить в активний режим і відображає на екрані інструкцію з його використання. Людей вибирає бажаний напій, додаткові ингридиенты до нього. При відсутності чого-небудь автомат видає про це повідомлення й пропонує зробити інший вибір. Далі автомат на екрані відображає вартість обраного напою й переходить у режим очікування введення грошей. Людей уводить в автомат необхідну суму. Якщо введена сума вірна, автомат готовить і видає напій. Інакше - повертає гроші. При бажанні людей може роздрукувати чек. Потім автомат знову переходить у режим очікування.



Опис станів:

Состояние	Описание состояния
Ожидание	Автомат по приготовлению кофе находится в режиме ожидания входных воздействий. В это время автомат отображает рекламные сообщения.
Отображение инструкции	После нажатия любой клавиши автомат находится в режиме отображения инструкции.
Проверка наличия необходимого напитка в автомате	Автомат находится в режиме проверки наличия выбранного напитка. В случае его

	отсутствия автомат выдает соответствующее сообщение и переходит в состояние отображения инструкции.
Отображение списка ингридиентов	Автомат отображает список ингридиентов, которые можно добавить к выбранному напитку
Проверка наличия выбранных ингридиентов в автомате	Автомат проверяет наличия выбранных ингридиентов. В случае их отсутствия автомат выдает соответствующее сообщение и переходит в состояние отображения списка ингридиентов.
Подсчет стоимости	Автомат определяет стоимость напитка и отображает ее на экране
Ожидание ввода денег	Автомат ожидает ввода денег в приемное устройство
Проверка	Автомат сравнивает полученную сумму с требуемой. Если сумма неправильная, автомат возращает деньги и переходит в состояние ожидания ввода денег
Приготовление напитка	Если введена правильная сумма, автомат начинает готовит выбранный напиток и затем выдает его человеку
Печать чека	Автомат печатает чек, если человек была выбрана печать чека

II Завдання до практичної роботи №10

1. По своїй предметній області розробити діаграму схем станів.
2. На діаграмі мають бути присутні паралельні та послідовні підстани.
3. Кожний стан описати у вигляді таблиці як показано в прикладі.

III Зробити висновки по виконанню практичної роботи №10

IV Контрольні питання для підготовки к практичній роботі №10

1. Що відображає діаграма схем станів?
2. Що таке стан?
3. Дайте визначення поняттю список внутрішніх подій.
4. Які види псевдо-станів Вам відомі?
5. Що таке подія?
6. Для чого використовується сторожова умова?

Практична робота №11

(4 години)

Тема: Побудова діаграми діяльності.

Мета: Отримати практичні навики побудови діаграми діяльності.

Хід роботи:

І. Теоретичні відомості

Стан дії

Стан дії (action state) є спеціальним випадком стану з деякою вхідною дією й принаймні одним вихідним зі стану переходом. Цей перехід неявно припускає, що вхідна дія вже завершилася. Стан дії не може мати внутрішніх переходів, оскільки воно є елементарним. Звичайне використання стану дії полягає в моделюванні одного кроку виконання алгоритму (процедури) або потоку керування.

Графічно стан дії зображується фігурою, що нагадує прямокутник, бічні сторони якого замінені опуклими дугами (рис. 1). Усередині цієї фігури записується вираження дії (action-expression), яка повинна є унікальним у межах однієї діаграми діяльності.

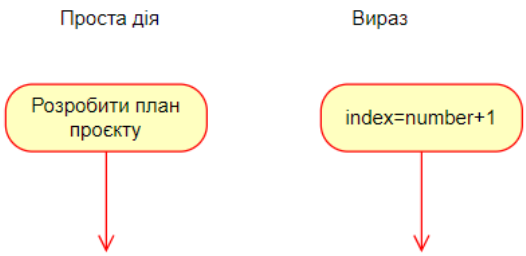


Рисунок 1 – Графічне зображення стану дії

Дія може бути записане природньою мовою, деякому псевдокоді або мові програмування. Ніяких додаткових або неявних обмежень при записі дій не накладається.

Іноді виникає необхідність представити на діаграмі діяльності деяка складна дія, яка, у свою чергу, складається з декількох більш простих дій. У цьому випадку можна використовувати спеціальне позначення так званого стану під-діяльності (subactivity state). Такий стан є графом діяльності й позначається спеціальною піктограмою в правому нижньому куті символу стану дії (рис. 2). Ця конструкція

може застосовуватися до будь-якого елемента мови UML, який підтримує "вкладеність" своєї структури. При цьому піктограма може бути додатково позначена типом вкладеної структури.

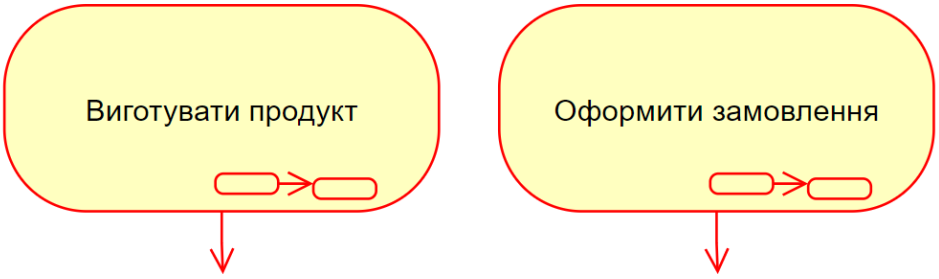


Рисунок 2 - Графічне зображення стану під-діяльності

Кожна діаграма діяльності повинна мати єдиний початковий і єдиний кінцевий стану. Вони мають такі ж позначення, як і на діаграмі станів. При цьому кожна діяльність починається в початковому стані й закінчується в кінцевому стані. Саму діаграму діяльності прийнято мати у своєму розпорядженні такий образ, щоб дії випливали зверху вниз. У цьому випадку початковий стан буде зображуватися у верхній частині діаграми, а кінцеве - у її нижній частині.

Переходи

При побудові діаграми діяльності використовуються тільки *не тригерні переходи,* тобто такі, які спрацьовують відразу після завершення діяльності або виконання відповідного дії. Цей перехід переводить діяльність у наступний стан відразу, як тільки закінчиться дія в попередньому стані. На діаграмі такий перехід зображується суцільною лінією зі стрілкою.

Якщо зі стану дії виходить єдиний перехід, то він може бути ніяк не позначений. Якщо ж таких переходів кілька, то спрацювати може тільки один з них. Саме в цьому випадку для кожного з таких переходів повинне бути явно записана *сторожова умова* в прямих дужках. При цьому для всіх вихідних з деякого стану переходів повинне виконуватися вимога істинності тільки одного з них. Подібний випадок зустрічається тоді, коли послідовно виконувана діяльність повинна розділитися на альтернативні галузі залежно від значення деякого проміжного результату. Така ситуація одержала назву *розгалуження*, а для її позначення застосовується спеціальний символ.

Графічно розгалуження на діаграмі діяльності позначається невеликим ромбом, усередині якого немає ніякого тексту.

Доріжки

Діаграми діяльності можуть бути використані не тільки для специфікації алгоритмів обчислень або потоків керування в програмних системах. Не менш важлива область їх застосування пов'язана з моделюванням бізнес-процесів. Дійсно, діяльність будь-якої компанії (фірми) також являє собою не що інше, як сукупність окремих дій, спрямованих на досягнення необхідного результату. Однак стосовно до бізнес-процесам бажане виконання кожної дії асоціювати з конкретним підрозділом компанії. У цьому випадку підрозділ відповідає за реалізацію окремих дій, а сам бізнес-процес представляється у вигляді переходів дій з одного підрозділу до іншого.

Для моделювання цих особливостей у мові UML використовується спеціальна конструкція назва, що одержала, доріжки (swimlanes). Мається на увазі візуальна аналогія із плавальними доріжками в басейні, якщо дивитися на відповідну діаграму. При цьому всі стани дії на діаграмі діяльності діляться на окремі групи, які відділяються друг від друга вертикальними лініями. Дві сусідні лінії й утворюють доріжку, а група станів між цими лініями виконується окремим підрозділом (відділом, групою, відділенням, філією) компанії.

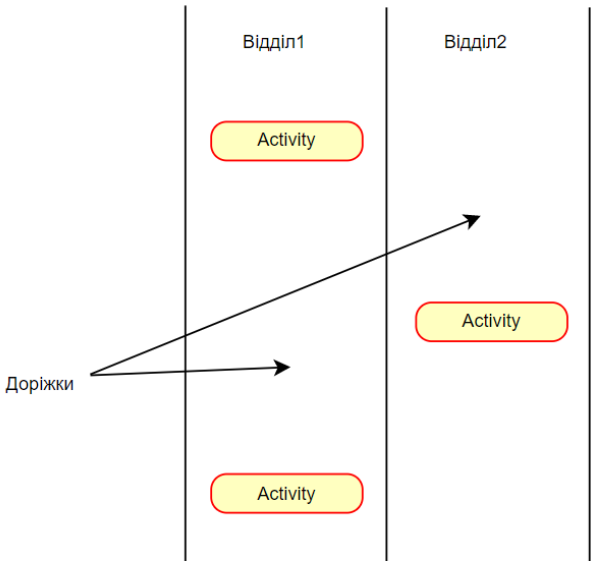


Рисунок 3 - Варіант діаграми діяльності з доріжками

Об'єкти

У загальному випадку дії на діаграмі діяльності виконуються над тими або іншими об'єктами. Ці об'єкти або ініціюють виконання дій, або визначають деякий результат цих дій. При цьому дії специфицируют виклики, які передаються від одного об'єкта графа діяльності до іншого. Оскільки в такому ракурсі об'єкти відіграють певну роль у розумінні процесу діяльності, іноді виникає необхідність явно вказати їх на діаграмі діяльності.

На діаграмі діяльності з доріжками розташування об'єкта може мати деякий додатковий зміст. А саме, якщо об'єкт розташовано на границі двох доріжок, те це може означати, що перехід до наступного стану дії в сусідній доріжці асоційований з готовністю деякого документа (об'єкт у деякому стані). Якщо ж об'єкт цілком розташований усередині доріжки, то й стан цього об'єкта цілком визначається діями даної доріжки.

Для приклада розглянемо ту ж саму область, що й в практичній роботі №10. (Рисунок 4)

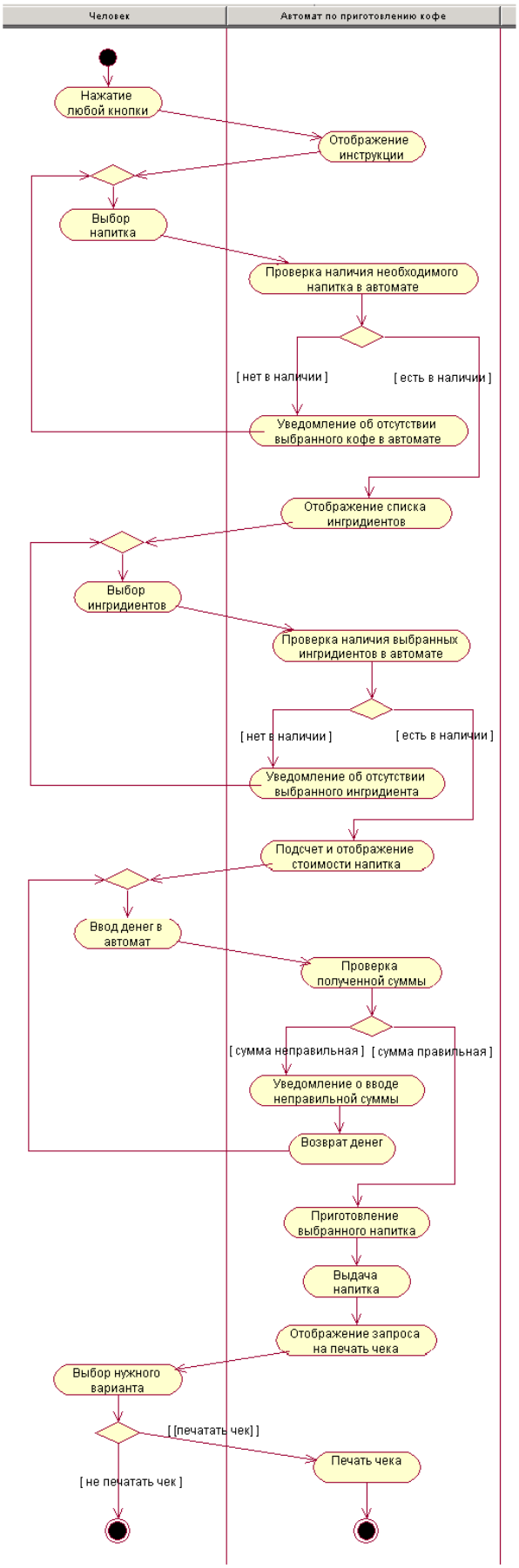


Рисунок 4 – Приклад діаграми діяльності

II Завдання до практичної роботи №11

1. По своїй предметній області розробити діаграму діяльності.
2. Діаграма має містити доріжки та об’єкти.
3. При побудові діаграми використовуйте: розподіл, злиття та розгалуження.

III Зробити висновки по виконанню практичної роботи №11

IV Контрольні питання для підготовки к практичній роботі №11

1. В чому різниця діаграми діяльності від діаграми станів?
2. Які елементи використовуються на діаграмі діяльності?
3. Що таке діяльність?
4. Що таке стан дії?
5. Які переходи використовуються на цих діаграмах?
6. Для чого необхідні доріжки?
7. Що відображають об’єкти?
8. Як відобразити паралельні дії?

Практична робота №12

(4 години)

Тема: Побудова компонентної діаграми.

Мета: Отримати практичні навики побудови компонентної діаграми.

Хід роботи:

І. Теоретичні відомості

Компонентні діаграми

Компонентна діаграма - перша із двох різновидів діаграм реалізації, що моделюють фізичні аспекти об'єктно-орієнтованих систем. Компонентна діаграма показує організацію набору компонентів і залежності між компонентами.

Елементами компонентних діаграм є компоненти й інтерфейси, а також відносини залежності й реалізації. Як і інші діаграми, компонентні діаграми можуть включати примітки й обмеження. Крім того, компонентні діаграми можуть

містити пакети або підсистеми, використовувані для угруповання елементів моделі у великі фрагменти.

Компоненти

По своїй суті компонентів є фізичним фрагментом реалізації системи, який містить у собі програмний код (вихідний, двійковий, що виконується), сценарні описи або набори команд операційної системи (маються на увазі командні файли). Мова UML дає наступне визначення.

Компонент - фізична й замінна частина системи, яка відповідає набору інтерфейсів і забезпечує реалізацію цього набору інтерфейсів.

Інтерфейс - дуже важлива частина поняття "компонент", його ми обговоримо в наступному підрозділі. Графічно компонент зображується як прямокутник із вкладками, що звичайно включає ім'я (рис. 1).

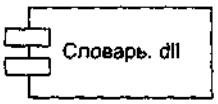


Рисунок 1 - Позначення компонента

Як показано на мал. 2, за допомогою відносини залежності можна явно відобразити відношення між компонентом і класами, які він реалізує.

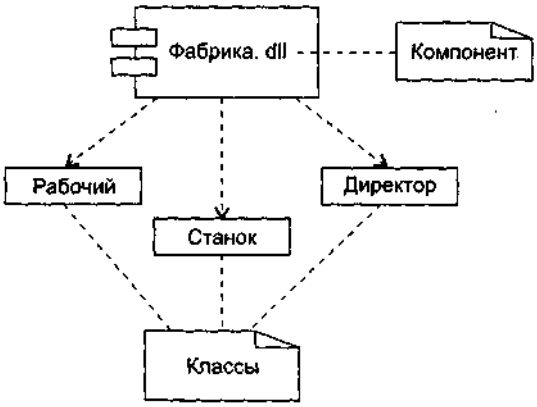


Рисунок 2 - Класи в компоненті

Інтерфейси

Інтерфейс - список операцій, які визначають послуги класу або компонента.

Компонування системи

За останні піввіку розроблювачі апаратури пройшли шлях від комп'ютерів розміром з кімнату до малюсіньких "ноутбуків", що забезпечили зрослі

функціональні можливості. За ті ж піввіку розроблювачі програмного забезпечення пройшли шлях від великих систем на Асемблері й Фортрані до ще більших систем на C++ і Java. На жаль, але програмний інструментарій розбудовується повільніше, чим апаратний інструментарій.

Цей секрет - компонента. Розроблювач апаратури створює систему з готових апаратних компонентів (мікросхем), що виконують певні функції, що й надають набір послуг через ясні інтерфейси. Завдання конструкторів спрощується за рахунок повторного використання результатів, отриманих іншими.

Повторне використання - магістральний шлях розвитку програмного інструментарію. Створення нового ПЗ з існуючих, працездатних програмних компонентів приводить до більш надійного й дешевого коду. При цьому строки розробки суттєво скорочуються.

Основна мета програмних компонентів - допускати складання системи із двійкових замінних частин. Вони повинні забезпечити початкове створення системи з компонентів, а потім і її розвиток - додавання нових компонентів і заміну деяких старих компонентів без перебудови системи в цілому. Ключ до втілення такої можливості - інтерфейси. Після того як інтерфейс визначений, до виконуваної системи можна підключити будь-який компонент, який задовольняє йому або забезпечує цей інтерфейс. Для розширення системи проводяться компоненти, які забезпечують додаткові послуги через нові інтерфейси. Такий підхід ґрунтується на особливостях компонента:

- Компонент фізичний. Він живе у світі бітів, а не логічних понять і не залежить від мови програмування
- Компонент - замінний елемент. Властивість заменяемості дозволяє замінити один компонент іншим компонентом, який задовольняє тим же інтерфейсам. Механізм заміни застережений сучасними компонентними моделями (COM, COM+, CORBA, Java Beans), що вимагають незначних перетворень або, що надають утиліти, які автоматизують механізм
- Компонент є частиною системи, він рідко автономний. Частіше компонент співробітничає з іншими компонентами й існує в

архітектурній або технологічному середовищі, призначеної для його використання. Компонент зв'язаний і фізично, і логічно, він позначає фрагмент великої системи

- Компонент відповідає набору інтерфейсів і забезпечує реалізацію цього набору інтерфейсів

Різновиди компонентів

Мир сучасних компонентів досить широкий і різноманітний. У мові UML для позначення нових різновидів компонентів використовують механізм стереотипів. Стандартні стереотипи, передбачені в UML для компонентів:

"executable" - Компонент, який може виконуватися у фізичному вузлі (має розширення .exe)

"library" - Статична або динамічна об'єктна бібліотека (має розширення .dll)

"file" - Компонент, який представляє файл, що містить вихідний код або дані (має розширення .ini)

"table" - Компонент, який представляє таблицю бази даних (має розширення .tbl)

"document" - Компонент, який представляє документ (має розширення .hlp)

У мові UML не визначені піктограми для перерахованих стереотипів, застосовувані на практиці піктограми компонентів показані на мал. 3-7.

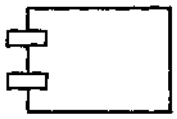


Рисунок 3 – Піктограма елемента що виконується.

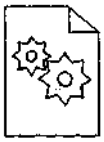


Рисунок 4 – Піктограма об’єктної Бібліотеки

Рисунок 5 – Піктограма документу з ісходним кодом

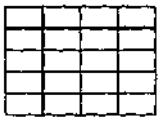


Рисунок 6 – Піктограма таблиці даних БД

Рисунок 7 – Піктограма документу

Приклад:

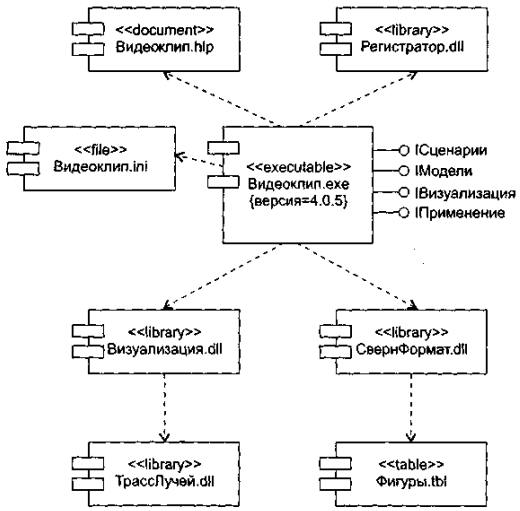


Рисунок 8 - Моделювання реалізації системи

Наприклад, на мал. 8 показана частина реалізації системи, группируемая навколо елемента, що виконується, Відеокліп.exe. Тут зображено чотири бібліотеки (Реєстратор.dll, Свернформат.dll, Візуалізація.dll, Трасслучей.dll), один документ (Відеокліп.hlp), один простий файл (Відеокліп.ini), а також таблиця бази даних (Фігури.tbl). У діаграмі зазначені відносини залежності, що існують між компонентами.

Для компонента, що виконується, Відеокліп.exe зазначений номер версії (за допомогою пгеговой величини), представлені його експортовані інтерфейси (Ісценарии, Івизуализация, Імодели, Іприменение). Ці інтерфейси утворюють API компонента "інтерфейс прикладного програмування).

На мал. 9 повторюється та ж діаграма, що моделює реалізацію, але тут для позначення компонентів використані спеціальні піктограми.

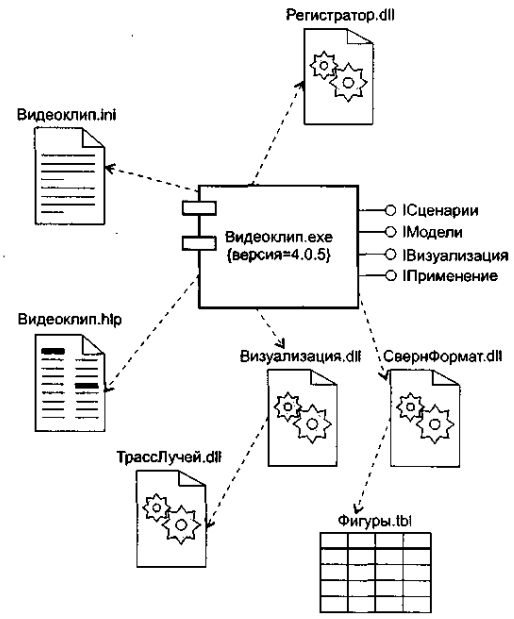


Рисунок 9 - Моделювання реалізації з використанням піктограм

II Завдання до практичної роботи №13

1. Зробити моделювання реалізації системи з використанням відомих видів компонентів. Для компонентів вказати інтерфейси.
2. Зробити моделювання реалізації системи з використанням спеціальних піктограм

III Зробити висновки по виконанню практичної роботи №12

IV Контрольні питання для підготовки к практичній роботі №12

1. Як задається ім'я компонента?
2. Що прийнято використовувати в якості простих імен?
3. Що таке компонент

Практична робота №13

(2 години)

Тема: Побудова діаграми розміщення.

Мета: Отримати практичні навики побудови діаграми розміщення.

Хід роботи:

I. Теоретичні відомості

Діаграма розміщення (розгортання) - друга із двох різновидів діаграм реалізації UML, що моделюють фізичні аспекти об'єктно-орієнтованих систем.

Діаграма розміщення показує конфігурацію обробних вузлів у період роботи системи, а також компоненти, "живучі" у них.

Елементами діаграм розміщення є вузли, а також відносини залежності й асоціації. Як і інші діаграми, діаграми розміщення можуть включати примітки й обмеження. Крім того, діаграми розміщення можуть включати компоненти, кожний з яких повинен жити в деякому вузлі, а також містити пакети або підсистеми, використовувані для угруповання елементів моделі у великі фрагменти. При необхідності візуалізації конкретного варіанта апаратної топології в діаграми розміщення можуть міститися об'єкти.

Вузли

Вузол - фізичний елемент, який існує в період роботи системи й представляє комп'ютерний ресурс, що має пам'ять, а можливо, і здатність обробки. Графічно вузол зображується як куб з іменем (рис. 1).

Рисунок 1 - Позначення вузла

Як і клас, вузол може мати додаткову секцію, що відображає розташовувані в ньому елементи (рис. 2).

Рисунок 12 - Розміщення компонентів у вузлі

Зрівняємо вузли з компонентами. Звичайно, у них є подібні характеристики:

- наявність імені;
- можливість бути вкладеним;
- наявність екземплярів.

Остання характеристика говорить про те, що на попередньому малюнку зображений тип Контролера. Зображення конкретного екземпляра, що належить цьому типу, презентовано на рис. 3.

Тепер обговоримо відмінності вузлів від компонентів. По-перше, вони належать до різних рівнів ієрархії у фізичній реалізації системи. Фізично система складається з вузлів, а вузли - з компонентів. По-друге, у кожного з них своє призначення. Компонент призначений для фізичного впакування й матеріалізації набору логічних елементів (класів і кооперацій). Вузол же є тем місцем, де фізично розміщаються компоненти, тобто відіграє роль "квартири" для компонентів.

Рисунок 3 - Екземпляр вузла

Відношення між вузлом і компонентами, які він розміщає, можна відобразити явно. Відношення залежності між вузлом Контролер і компонентами Введення.exe, Висновки.exe ілюструє рис. 4. Правда, найчастіше такі відносини не відображаються. Їх зручно представляти в окремій специфікації вузла.

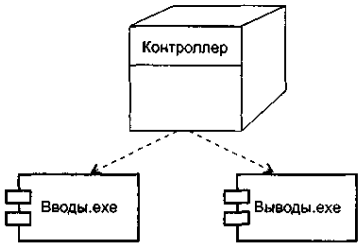


Рисунок 4 - Залежність вузла від компонентів

Угруповання набору об'єктів або компонентів, розташовуваних у вузлі, звичайно називають розповсюджуваним модулем.

Графічно діаграма розміщення - це граф з вузлів (або екземплярів вузлів), з'єднаних асоціаціями, які показують існуючі комунікації. Екземпляри вузлів можуть містити екземпляри компонентів, що живуть або запускаються у вузлах. Екземпляри компонентів можуть містити об'єкти. Як показано на мал. 5, компоненти з'єднуються один з одним пунктирними стрілками залежностей (прямо або через інтерфейси).

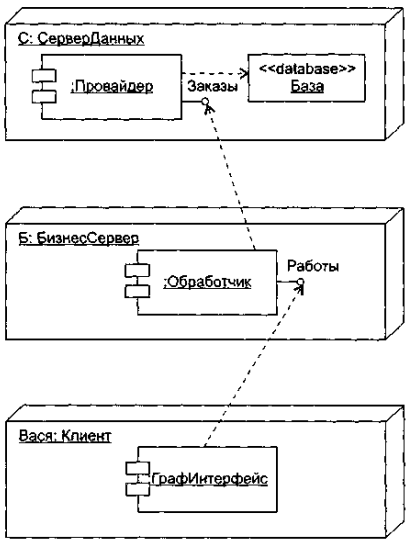


Рисунок 5 - Моделювання розміщення компонентів

На цій діаграмі зображена типова трехуровневая система:

- рівень бази даних реалізований екземпляром С вузла СервердДанных;
- рівень бізнес-логіки представлений екземпляром Б вузла БизнесСервер;
- рівень графічного інтерфейсу користувача утворений екземпляром Вася вузла Клієнт.

II Завдання до практичної роботи №13

1. Зробити побудову діаграми розгортання, базуючись на клієнт-серверному проєкті, створеному раніше на курсі дисципліни БД.

III Зробити висновки по виконанню практичної роботи №13

IV Контрольні питання для підготовки к практичній роботі №13

1. Які вершини й ребра утворюють діаграму розміщення?
2. Чим відрізняється вузол від компонента?
3. Де можна використовувати й де не можна використовувати екземпляри компонентів?

Практична робота №14

(10 години)

Тема: Конструювання програмного забезпечення.

Мета: отримати практичні навики побудови інформаційної системи за створеною моделлю UML.

Хід роботи:

1. Ґрунтуючись на діаграмі класів розробити БД, що міститиме всю необхідну для функціонування системи інформацію.
2. Ґрунтуючись на діаграмах прецедентів, кооперацій та компонентів розробити структуру інформаційної системи, передбачивши функції означені в діаграмі Use Case.
3. Ґрунтуючись на діаграмах взаємодії, станів та діяльності реалізувати роботу функції з п. 2.
4. Протестувати роботу системи.
5. Розробити інструкцію користувача.

Додаток А. Приклад оформлення титульного аркушу звіту

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ФАХОВИЙ КОЛЕДЖ ПРОМИСЛОВОЇ АВТОМАТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

ЗВІТ З ПРАКТИЧНОЇ РОБОТИ №__

з дисципліни_____Конструювання програмного забезпечення_____
(Назва дисципліни)

спеціальності_____121 «Інженерія програмного забезпечення»_____
(шифр і назва спеціальності)

тема _____
(Тема практичної роботи)

мета _____
_____
_____
(Мета практичної роботи)

Здобувача фахової передвищої освіти
____ курсу _____ групи
_____
(Власне ім'я та прізвище)

Одеса 20__ р.