Міністерство освіти і науки України
Чорноморський державний університет
імені Петра Могили

ПРАКТИКУМ
з об'єктно-орієнтованих методологій створення комп'ютерних систем

**Методичні вказівки
до виконання практичних робіт
для студентів напряму «Комп'ютерні науки»**

Випуск № 134

Видавництво ЧДУ ім. Петра Могили
Миколаїв – 2009

УДК 004.45(076.5)
ББК 32.973-01
П 69

Рекомендовано до друку вченою радою ЧДУ ім. Петра Могили (протокол № 2 від 08.10.2009 р.).

Рецензент:
Гожий О.П., к.т.н., доцент, зав. кафедрою інформаційних технологій та програмних систем ЧДУ імені Петра Могили.

П 69

Практикум з об'єктно-орієнтованих методологій створення комп'ютерних систем: Методичні вказівки до виконання практичних робіт для студентів напряму «Комп'ютерні науки» // Укладач В.О. Дегтярьов. – Миколаїв: Видавництво ЧДУ ім. Петра Могили, 2009. – Вип. 134. – 68 с. (Методична серія)

Методичні вказівки вміщують теоретичний і практичний матеріал до виконання практичних робіт із дисципліни «Об'єктно-орієнтовані методології створення комп'ютерних систем». Розглянуто 13 практичних робіт, які пов'язані з вивченням особливостей аналізу, проектування та розробки програмного забезпечення складних комп'ютерних систем. Роботи містять вивчення концепцій уніфікованої мови моделювання UML та проектування програмного забезпечення з використанням CASE засобів на прикладі Rational Rose. Наведений перелік завдань із темами практичних робіт, довідкові матеріали, необхідні для виконання практичних робіт та список рекомендованої літератури. Призначені для студентів спеціальності «Інтелектуальні системи прийняття рішень».

УДК 004.45(076.5)
ББК 32.973-01

ЗМІСТ

Вступ 4

Практична робота № 1. Аналіз предметної галузі 5

Практична робота № 2. Написання usecases (варіантів використання системи) 6

Практична робота № 3. Побудова діаграм варіантів використання (usecase diagrams) 13

Практична робота № 4. Побудова діаграм взаємодії (interaction diagrams) 16

Практична робота № 5. Побудова діаграм класів 21

Практична робота № 6. Діаграми станів та переходів 28

Практична робота № 7. Побудова діаграм діяльності 32

Практична робота № 8. Побудова діаграм компонентів 36

Практична робота № 9. Побудова діаграм впровадження, кодогенерація та зворотне проектування. 40

Практична робота № 10. Розробка інтерфейсу користувача 45

Практична робота № 11. Взаємодія програмної системи з базою даних 50

Практична робота № 12. Генерація вихідних даних 54

Список літератури 64

Додаток А. Перелік варіантів завдання 65

ВСТУП

Цикл практичних робіт з дисципліни «Об'єктно-орієнтовані методології створення комп'ютерних систем» призначений для вивчення методів та засобів аналізу, проектування та розробки програмного забезпечення для складних комп'ютерних систем.

Метою курсу є набуття студентами практичних навичок у галузі проектування комп'ютерних систем із використанням CASE засобів; вивчення уніфікованої мови моделювання UML та удосконалення практичних навичок розробки програмного забезпечення.

Під час виконання практичних робіт із курсу ООМ студенти виконують роботу з аналізу, проектування та створення програмного забезпечення відповідно до обраного варіанта. Курс ООМСКС тісно пов'язаний із курсом систем керування базами даних, оскільки робота на кожному етапі проводиться відповідно до створеного формату бази даних. Кінцевою метою практичного курсу є створення кожним студентом працюючого варіанта програмного забезпечення, яке може використовуватися для практичних потреб.

Перелік практичних робіт курсу умовно розподілено на три частини. Практичні роботи № 1-2 присвячені аналізу прикладної галузі майбутньої програмної системи та створенню варіантів використання – сукупності сценаріїв роботи системи, описаних з точки зору користувача.

Практичні роботи № 3-9 пов'язані з проектуванням комп'ютерної системи за допомогою мови UML та використанням CASE засобу Rational Rose. В роботах показаний процес створення архітектури системи, починаючи від діаграми варіантів використання (Usecase Diagram) і закінчуючи діаграмою впровадження (Deployment Diagram). Практична робота № 9 також включає відомості щодо проведення автоматичної кодогенерації та зворотного проектування.

Практичні роботи № 10-12 визначають перелік етапів для безпосереднього створення програмного забезпечення шляхом створення та вдосконалення програмного коду. Дані роботи імітують ітеративний процес розробки від етапу створення базового прототипу інтерфейсу користувача, через кодування бізнес логіки програмного комплексу до розробки можливостей генерації вихідної документації.

Практична робота № 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Мета роботи: відповідно до варіанта завдання, узгодженого з викладачем, знайти приклади 3-4 аналогічних програмних систем та проаналізувати принципи їх функціонування. На основі отриманих даних створити проект технічного завдання для розробки власної системи.

Звіт повинен складатися з двох частин.

Перша частина містить аналіз аналогічних систем. Для кожного аналогу ПЗ визначити наступні характеристики:

- назва;
- розробник (дистриб'ютор);
- архітектура (desktop application, client-server, 3tier web application);
- мова реалізації;
- перелік функцій, характеристик (не менше 5);
- аналіз переваг та недоліків даного ПЗ;
- джерело інформації (веб-сайт);

Друга частина визначає перелік характеристик системи, що проектується та розробляється студентом. Для програмної системи, що проектується визначити:

- призначення ПЗ;
- основні функції (не менше 5);
- користувачі системи (2-3 користувачі);
- архітектура системи;
- опис структури (схема);
- сценарії роботи (не менше 3-х сценаріїв);
- основні логічні сутності (не менше 5);
- основні таблиці БД, що використовуються (не менше 5 таблиць);
- засоби апаратної та програмної реалізації (платформа, ОС та мова програмування);
- output (вихідні дані): таблиці, звіти, графіки (не менше 3).

Практична робота № 2
НАПИСАННЯ USECASES (ВАРІАНТІВ ВИКОРИСТАННЯ СИСТЕМИ)

Теоретичні відомості

Usecase – це текстовий опис сукупності сценаріїв, що виконуються користувачем при роботі з системою для досягнення певної мети.

Сценарій – послідовність дій при взаємодії користувача із системою для виконання певної операції.

Наприклад, купівля товарів у супермаркеті з використанням кредитної картки і невдала спроба купівлі через перевищення кредитного ліміту – це сценарії, а купівля товарів у супермаркеті – usecase.

Приклад 1. Зняття готівки в банкоматі.

Користувач підходить до банкомату та вставляє свою картку. Система перевіряє картку та просить ввести пін-код. Після перевірки пін-коду система виводить головне меню, де користувач обирає опцію видачі готівки. Він визначає суму грошей та підтверджує виконання операції. Банкомат видає готівку та чек. Система видає запит про здійснення наступної операції. Користувач обирає опцію «не виконувати», забирає картку та йде.

Приклад 2. Замовлення літератури в бібліотеці.

Головний сценарій (успішний):

Користувач підходить до бібліотекаря і надає йому замовлення (перелік видань). Бібліотекар заносить номер читацького квитка до системи і перевіряє облікову картку користувача. Бібліотекар шукає кожне видання в базі і визначає кількість екземплярів вільних у даний момент часу. Система видає шифри (коди), за якими бібліотекар знаходить видання у сховищі, заносить номери книг до картки користувача і видає йому літературу. При цьому читацький квиток лишається у бібліотекаря.

Альтернативні сценарії:

1. Користувач бібліотеки є боржником, бібліотекар не може видати йому літературу.
2. Бібліотекар не може видати користувачу літературу, оскільки в даний момент часу немає вільних екземплярів у сховищі.

3. Читацький квиток користувача є недійсним. Бібліотекар вилучає його.
4. Технічний збій роботи системи. Видається повідомлення «немає зв'язку з сервером баз даних». Бібліотекар викликає адміністратора системи.

Написання usecases дозволяє чітко визначити хто є користувачем системи, які її сценарії роботи, що є метою використання системи. Usecases – це функціональні та поведінкові вимоги до системи, які показують, що саме вона має робити.

Існують три форми написання usecases:

1. Коротка – короткий опис в один абзац одного зі сценаріїв (зазвичай успішного) роботи системи (приклад 1). Виконуються під час початкового аналізу вимог до системи.
2. Поверхнева – поверхневий опис у вільній формі усіх сценаріїв (головного і альтернативних) одного з usecases (приклад 2). Виконуються під час початкового аналізу вимог до системи.
3. Повна – всі кроки і дії детально описані, включаючи перед- та постумови виконання usecase. Виконуються на стадії відбору з повного переліку usecases у короткій та поверхневій формах невеликої частини важливих (критичних для роботи системи) usecases.

Повна форма опису usecase має перелік розділів, коротко описаних у таблиці 1. Нижче наведено приклад складання usecase у повній формі для системи продажу товарів у супермаркеті.

Таблиця 1

Параметри опису usecase

Usecase section	Comment
Use Case Name	Назва usecase (починається з дієслова)
Scope	System or Business
Level	User-goal or Sub-function
Primary Actor	Головний актор
Stakeholders and interests	Перелік осіб, які зацікавлені в виконанні даного usecase та мета, яку вони при цьому переслідують
Preconditions	Список передумов, які повинні виконуватись для початку виконання usecase
Success guarantee	Список умов, при виконанні яких можна говорити про успішне закінчення виконання usecase
Main Success Scenario	Головний успішний сценарій usecase. Найчастіше є безумовним

Закінчення таблиці 1

Extensions	Альтернативні сценарії успішного чи неуспішного закінчення usecase
Special Requirements	Спеціальні (нефункціональні) вимоги
Technology and Data Variations List	Поради для реалізації певних кроків usecase
Frequency of Occurrence	Частота виконання usecase при користуванні системою (у відсотках)
Miscellaneous	Додаткові вимоги чи відкриті питання

Приклад 3. Купівля товарів у супермаркеті.
Scope: система продажу товарів в супермаркеті (System)
Level: User-goal
Primary Actor: касир
Stakeholders and interests:

1. Касир: зацікавлений у точному швидкому вводі інформації про товари та відсутності помилок вводу, які призведуть до штрафу (зменшення зарплатні касира)
2. Покупець: зацікавлений у швидкому придбанні товарів (отриманні послуг) та зручному відображенні розрахунків суми покупки.
3. Компанія (продавець): зацікавлена в точній обробці транзакцій при покупці товарів та задоволенні інтересів користувача.
4. Менеджер: зацікавлений у швидкому розв'язанні проблем під час повернення товарів та легкій перевірці операцій, що здійснюють касири.
5. Державна податкова адміністрація: зацікавлена в отриманні податків від кожного продажу товарів.

Preconditions: касир виконав вхід до системи (авторизація)
Main Success Scenario:

1. Покупець підходить до каси з сукупністю товарів (послуг), які він хоче придбати.
2. Касир стартує нову продаж.
3. Касир вводить послідовно всі товари.
4. Система опрацьовує код кожного товару та виводить його назву, кількість та суму покупки для кожного товару.
5. Система розраховує загальну суму покупки та суму нарахованих податків.
6. Касир озвучує суму покупцеві та просить розрахуватися.
7. Покупець розраховується і система опрацьовує оплату.
8. Система запам'ятовує суму покупки до бази даних.

9. Система друкує чек.
10. Користувач відходить від каси з чеком і товарами.

Extensions:

a) у будь-який час менеджер виконує специфічну операцію:
   1) менеджер вводить свій код авторизації;
   2) менеджер виконує специфічну операцію (наприклад, перевіряє баланс даної каси);
   3) менеджер виходить з системи, і вона повертається до режиму роботи з касиром.

b) у будь-який час виникає фатальна помилка в системі:
   1) касир перевантажує систему, входить до неї та вводить запит про повернення до попереднього стану;
   2) система відновлює попередній стан.
      2a) система не може відновити попередній стан:
         – система видає помилку на екран та зберігає її у лог;
         – касир починає новий продаж.

1a) покупець чи менеджер хоче повернутися до відкладеного продажу:
   1) касир виконує операцію повернення та вводить код продажу;
   2) система відображає відтворений продаж.
      2a) продаж не було знайдено:
         – система виводить помилку на екран;
         – касир розпочинає новий продаж і вводить дані про всі товари з початку.

2-4a) покупець говорить касиру, що у нього є картка для отримання знижки:
   1) касир вводить номер картки до системи;
   2) система запам'ятовує код картки та вид знижки для розрахунку загальної суми продажу.

3a) код товару не знайдено в базі даних:
   1) система виводить помилку на екран;
   2) касир опрацьовує помилку:
      2a) додатковий код товару знаходиться на ньому і касир може його прочитати:
         – касир вводить код вручну;
         – система опрацьовує код та виводить назву товару та ціну.
      2b) касир виконує usecase «Знайти код невідомого товару» для ідентифікації товару.

3b) у покупця є декілька однакових товарів:
   1) касир може ввести код товару та його кількість вручну.

3-6a) користувач просить видалити один з товарів із поточного продажу:
1) касир вводить номер товару для видалення;
2) система видаляє строку товару з продажу і перераховує загальну суму.

3-6b) покупець просить касира відмінити продаж у цілому:
1) касир відміняє продаж.

3-6c) касир відкладає продаж:
1) система запам'ятовує продаж, яка може бути відновлена в будь-який час;
2) система видає спеціальний чек із переліком товарів та кодом продажу, який дозволить продовжити продаж.

4b) Покупець вважає, що товар є зіпсованим і пропонує купити його за нижчою ціною:
1) касир робить запит у менеджера;
2) менеджер дозволяє виконати продаж за нижчою ціною;
3) касир вводить вручну ціну, що є нижчою за попередню;
4) система відображає нову ціну для даного товару в поточному продажі.

5a) користувач говорить, що він повинен отримати знижку:
1) касир робить запит щодо знижки;
2) касир вводить ідентифікаційний код покупця;
3) система розраховує сумарну знижку на продаж.

6a) покупець говорить касиру, що він хотів розплатитися готівкою, але не має достатньо грошей:
1) касир просить обрати інший метод сплати за товари:
1a) покупець просить відмінити продаж.

7a) оплата готівкою:
1) касир вводить суму грошей, яку дав покупець;
2) система розраховує суму решти і відкриває касу;
3) касир видає решту покупцю;
4) система вносить випадок оплати до бази даних.

7b) оплата кредитною карткою:
1) покупець вводить інформацію про кредитну картку;
2) система показує суму оплати;
3) касир підтверджує суму оплати;
4) система відсилає авторизаційний запит до зовнішньої Системи Оплати Товарів та послуг та робить запит на підтвердження оплати;
5) система отримує підтвердження оплати, виводить інформацію для касира та відкриває касу;
6) система зберігає даний вид оплати за товари у базі даних;

7) касир просить покупця зробити підпис на чеку для підтвердження оплати. Покупець ставить підпис;
8) касир складає чек до каси та замикає її.

7c) касир робить відміну оплати:
1) система повертається до режиму вводу товарів.

9a) покупець просить видати подарунковий чек (без суми продажу):
1) касир робить запит про подарунковий чек. Система друкує подарунковий чек.

9b) в принтері закінчився папір:
1) система дає сигнал про закінчення паперу;
2) касир замінює папір;
3) касир робить запит про повторення друку чеку.

Special Requirements:

1) великий плоский монітор, що має функції сенсорного екрану. Текст повинно бути гарно видно з відстані в 1 м;
2) система авторизації запитів про оплату кредитною карткою повинна видавати результат за 30 секунд у 90 відсотків випадків;
3) система перекладу тексту на декілька мов;
4) додаткові бізнес-правила можуть бути додані до пунктів 3-7.

Technology and Data Variations List:

1) менеджер виконує авторизацію під час скасування певної операції за допомогою зчитування номера картки менеджера кард-рідером чи введенням коду менеджера з клавіатури;
2) код товару вводиться сканером штрих кодів товарів чи з клавіатури;
3) система може використовувати декілька схем кодування товарів (UTC, EAN, JAN, SKU);
4) інформація про кредитну картку вводиться за допомогою кард-рідера чи з клавіатури;
5) підпис покупця для підтвердження кредитної операції робиться на товарному чеці.

Frequency of Occurrence: 95 %.

Miscellaneous (Open Issues):

1) провести аналіз різних варіантів сплати податків;
2) вивчити можливість відновлення роботи системи після збою;
3) яка додаткова функціональність потрібна для різних прикладних галузей?
4) чи повинен касир забирати гроші з каси, коли він виходить із системи (закінчує зміну)?
5) чи може покупець використовувати сканер для ідентифікації товарів під час покупки, чи це повинен робити касир?

Даний приклад є більш ніж детальним прикладом опису usecase в повній формі, але є цілком реальним (розроблений під час ОО аналізу вимог для системи продажу товарів NextGen POS, яка реалізована мовою Java).

Завдання

Відповідно до обраної теми (варіанта) на основі проведеного аналізу прикладної галузі (практична робота № 1) розробити три різні usecases (по одному в короткій, поверхневій та повній формах відповідно) для своєї системи. Повна форма опису має містити всі пункти наведені в таблиці 1. Головний успішний сценарій повинен мати не менше 10 кроків. Передбачити не менше 5 альтернативних сценаріїв.

Контрольні запитання

1. Дайте визначення варіанта використання (usecase).
2. Чим відрізняються варіант використання та сценарій (scenario)?
3. Які форми опису варіантів використання ви знаєте?
4. Дайте визначення та наведіть приклад короткої форми опису usecase.
5. Дайте визначення та наведіть приклад поверхневої форми опису usecase.
6. Що таке головний успішний сценарій?
7. Дайте визначення альтернативних сценаріїв? Чи можуть вони бути успішними/неуспішними?
8. Дайте визначення повної форми опису usecase.
9. Хто може виступати в ролі актора для варіанта використання?
10. Хто може виступати в ролі зацікавлених осіб (stakeholders) для варіанта використання?
11. Яким чином визначаються альтернативні сценарії і як вони пов'язані з головним успішним?
12. Яким чином визначається параметр frequency of occurrence для повної форми опису варіанта використання?

Практична робота № 3
ПОБУДОВА ДІАГРАМ ВАРІАНТІВ ВИКОРИСТАННЯ (USECASE DIAGRAMS)

Теоретичні відомості

Діаграми варіантів використання (***usecase diagrams***) використовуються для відображення сценаріїв використання системи (***usecases***) та користувачів системи (***actors***), які використовують її функції.

Актори на діаграмі варіантів використання позначаються символом людини, а варіанти використання – еліпсом. Актори та варіанти використання поєднуються напрямленою асоціацією (unidirectional ***association***) – стрілкою, що спрямована від актора до варіанта використання. Також актори можуть поєднуватися з використанням зв'язків узагальнення. На рис. 1 показано фрагмент діаграми варіантів використання для інтернет-магазину. Актор «Покупець» при цьому може виконати сценарій «Замовити товар».

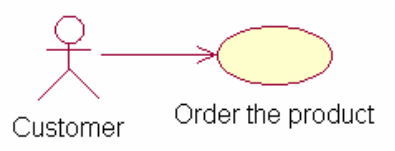


Рис. 1. Фрагмент діаграми варіантів використання в Rational Rose

Варіанти використання можуть бути пов'язані між собою трьома видами зв'язків: узагальненням (***generalization***), розширенням (***extend*** relationship) та включенням (***include*** relationship).

Відношення узагальнення (generalization) показують відношення між загальним і частковим. Наприклад на рис. 2 варіанти використання «Search by category» та «Search by producer» є частковими випадками загального варіанта «Search product», тому вони поєднані даним відношенням. Також дане відношення може використовуватися для поєднання акторів. Актор «Administrator» може виконувати всі функції актора «Customer», тобто виступає частковим випадком покупця, але може виконувати і специфічні операції (варіант використання «Check db info»).

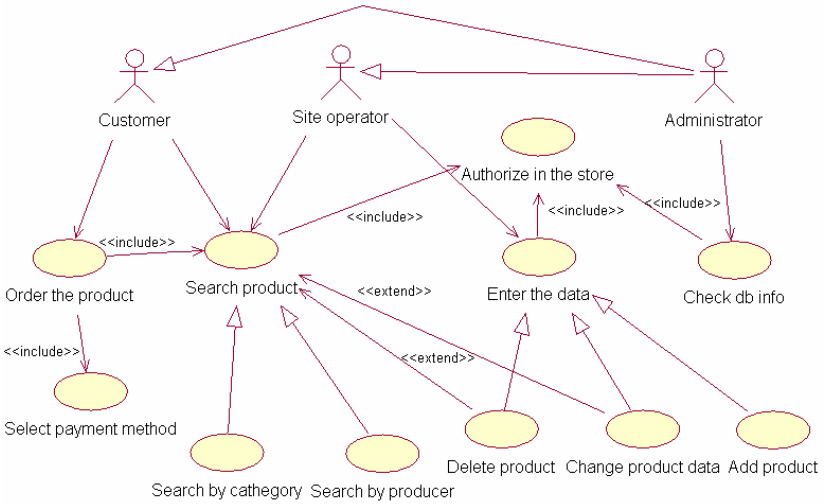


Рис. 2. Повна діаграма варіантів використання для інтернет-магазину

Відношення включення (include) відображає зв'язок «ціле – частина», тобто один варіант завжди в певний момент виконання повністю включає інший. Для прикладу частиною варіанта використання «Order the product» є сценарії «Search product» та «Select payment method», оскільки для того, щоб замовити товар покупець завжди має відшукати його в каталозі та обрати метод оплати.

Відношення розширення (extend) визначає такий тип відношення, коли один варіант за певних умов повністю використовує інший (розширює його). Так, наприклад, оператор Інтернет магазину може видалити товар («Delete product»), знаючи його ідентифікатор, або провівши попередньо пошук товару («Search product»).

Завдання

У середовищі Rational Rose створити діаграму варіантів використання для обраного варіанта комп'ютерної системи. Діаграма повинна містити усіх акторів (користувачів системи) та по три варіанти використання для кожного актора. Пов'язати варіанти використання та акторів, при цьому використати усі види зв'язків (unidirectional association, generalization, extend relationship, include relationship).

Контрольні запитання

1. Для чого призначена діаграма варіантів використання (usecase diagram)?
2. Дайте визначення актора (actor) програмної системи. Хто може виступати в ролі акторів?
3. Дайте визначення варіанта використання (usecase). Яким чином варіант використання визначається на діаграмі?
4. Перелічіть відношення, що можуть використовуватися на діаграмі.
5. Дайте визначення відношення узагальнення (generalization). Наведіть приклад.
6. У чому відмінність між відношенням узагальнення між акторами та між варіантами використання?
7. Дайте визначення відношення асоціації (association). Наведіть приклад.
8. Дайте визначення відношення включення (include). Наведіть приклад.
9. Дайте визначення відношення розширення (extend). Наведіть приклад.
10. Які відношення можуть використовуватися для поєднання тільки акторів, акторів та варіантів використання, тільки варіантів використання?

Практична робота № 4
ПОБУДОВА ДІАГРАМ ВЗАЄМОДІЇ (INTERACTION DIAGRAMS)

Теоретичні відомості

У Rational Rose діаграми взаємодії (***Interaction Diagrams***) відображають взаємодію логічних елементів системи між собою у процесі виконання актором певного варіанта використання. Розрізняють два типи діаграм взаємодії: діаграми послідовності (***Sequence Diagrams***) та кооперації (***Collaboration Diagrams***), які є різними представленнями одного і того ж процесу.

Головними елементами ***діаграм послідовності*** є *об'єкти*, які є логічними сутностями, що представляють окремі елементи системи та *повідомлення*, якими вони обмінюються. В якості об'єктів можуть виступати також актори. Повідомлення можуть бути не тільки абстрактними діями, що виконуються, але і методи класів, створених на діаграмі класів. Повідомлення на діаграмі послідовності пронумеровані, тобто мають чітку послідовність.

Для прикладу розглянемо систему електронного документообігу на підприємстві. Для побудови діаграми послідовності оберемо варіант використання «Створити документ», який виконує актор «Користувач» (рис. 3).

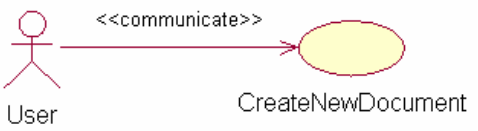


Рис. 3. Фрагмент діаграми варіантів використання для системи документообігу

Наведемо опис головного успішного сценарію даного варіанта використання у короткій формі: *Користувач має на екрані форму із запитом на створення документа, він обирає шаблон документа зі списку шаблонів, задає шлях для збереження та назву документа; система зберігає введені дані в якості параметрів документа, запускає прикладне програмне забезпечення, яке відповідає формату документа (наприклад MS Word, MS Excel і т. д.) та створює документ із відповідною назвою.*

На рис. 4 наведено діаграму послідовності, створену в середовищі Rational Rose для даного прикладу. У верхній частині діаграми наведено перелік об'єктів (логічні сутності), які взаємодіють між собою у процесі виконання сценарію. Часова шкала на даній діаграмі направлена згори донизу, крім того повідомлення пронумеровані відповідно до черги їх пересилання між об'єктами. Нижче наведено короткий опис подій, що відбуваються при виконанні даного варіанта використання.

При ініціалізації діалогу для створення нового документа (1) завантажується (2) та відображається (3) список шаблонів документів. При виборі користувачем одного з шаблонів (4), в діалозі відображається його початковий вигляд (5). Після того, як користувач підтверджує свій вибір остаточно, наприклад, скориставшись методом DoubleClick (6), система ініціалізує діалог (7), де просить ввести шлях до директорії та ім'я файла, у якому документ буде зберігатися. При цьому відображаються default значення для шляху та імені файла (8). Користувач вводить шлях (9) та ім'я файла (10), система перевіряє введені дані (11). Система зчитує інформацію шаблона (12) і записує всі параметри до об'єкта, який представляє документ (13). Потім документ додається до списку активних документів (14) і відповідно до параметрів документа запускається прикладна програма, яка відповідає формату шаблона документа (15). При цьому в прикладній програмі створюється новий документ з ім'ям, яке ввів користувач (16) (рис. 4).

Додатковими елементами діаграми послідовності є лінії життя об'єктів (довгі вертикальні пунктирні лінії), які відображають час життя об'єкта від його створення до знищення; фокус керування (прямокутники на лініях життя об'єктів) відображає, який об'єкт виконує операції в певний момент часу; та символи знищення об'єктів (хрест на лінії життя об'єкта), що відображає процес знищення об'єкта (рис. 5). Наприклад, об'єкт «Authorization Record» на діаграмі створюється на 2-му кроці виконання сценарію, використовується протягом 4-6 кроків та знищується після шостого кроку.

Основні типи повідомлень на діаграмі послідовності:

- пряме – повідомлення, яке об'єкт-ініціатор надсилає об'єкту-приймачу (1-4);
- рефлексивне, яке об'єкт надсилає сам собі (5);
- зворотнє, при якому управління повертається об'єкту ініціатору (6), часто повідомлення даного типу не мають назви.

Діаграма кооперації має те ж саме призначення, що і діаграма послідовності, але відображає процес виконання варіанта використання

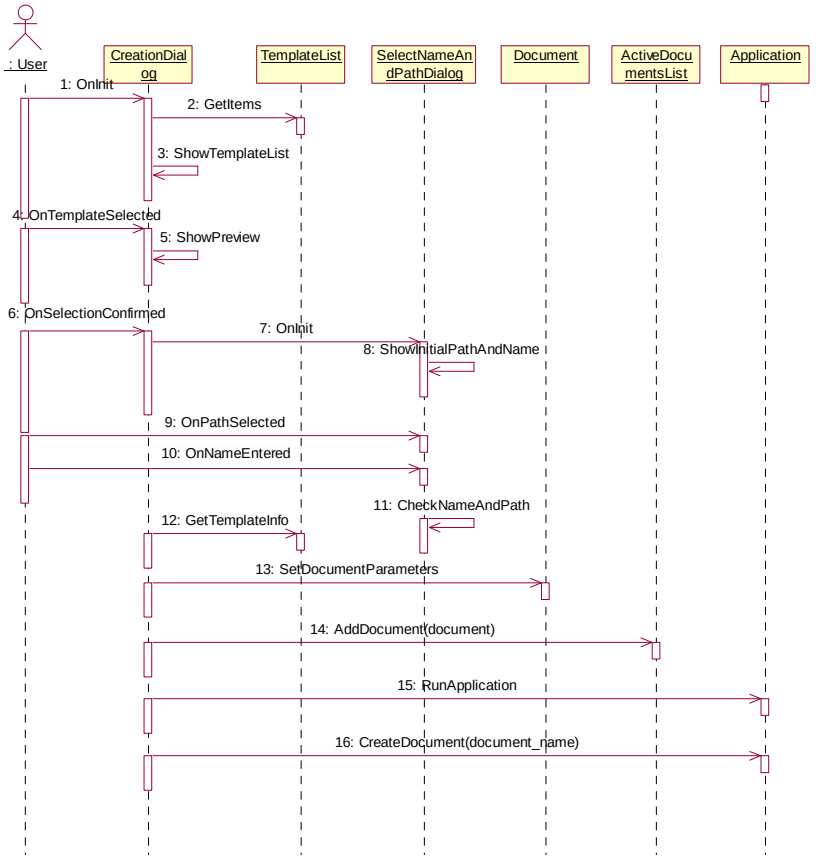


Рис. 4. Діаграма послідовності для варіанта використання «Створити новий документ»

в іншій нотації (рис. 6). Її головними компонентами є об’єкти, назви повідомлень та їх напрям. Характеристика даних компонентів була надана у процесі опису діаграми послідовності.

Примітка: діаграму послідовності можна автоматично перетворити в діаграму кооперації (або навпаки), натиснувши клавішу F5.

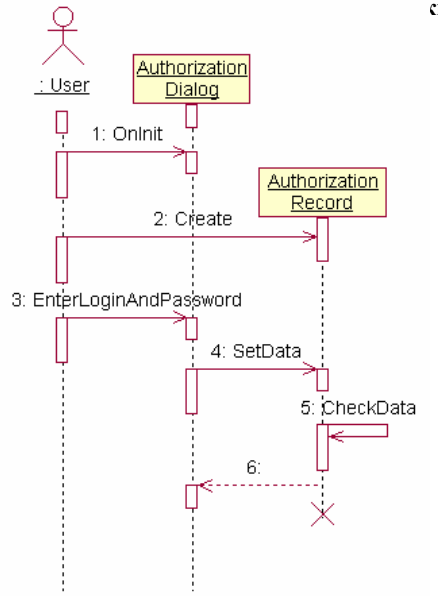


Рис. 5. Фрагмент діаграми послідовності для процесу авторизації користувача

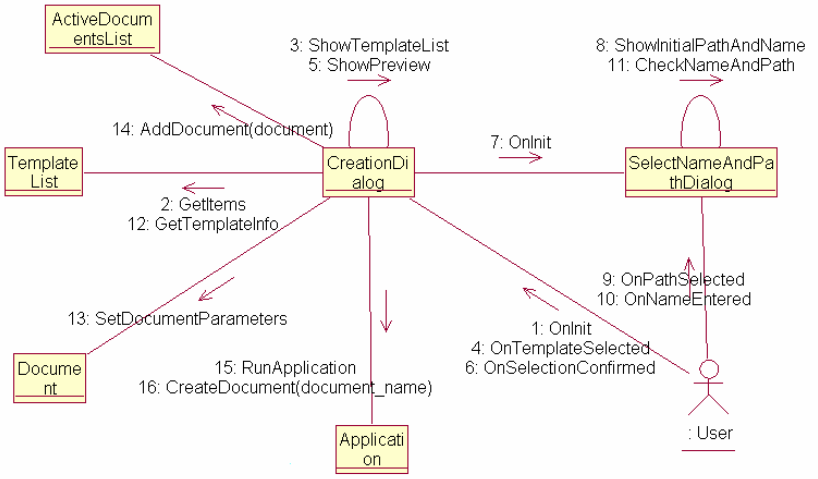


Рис. 6. Діаграма кооперації для варіанта використання «Створити новий документ»

Завдання для студентів

Для кожного варіанта використання на Usecase Diagram створити Sequence або Collaboration Diagram (тобто у проекті повинно бути не менше шести діаграм кооперації та послідовності). На кожній діаграмі взаємодії повинен бути головний актор (при наявності) та не менше 5 об'єктів. Кожна діаграма взаємодії повинна містити не менше 10 повідомлень, якими обмінюються об'єкти в процесі виконання сценарію. Загальна сума різних об'єктів у проекті повинна налічувати 12-15 об'єктів. Об'єкти та повідомлення на діаграмах повинні мати зрозумілі назви. При побудові діаграм використовувати прямі, рефлексивні та зворотні типи повідомлень, а також символи знищення об'єктів.

Контрольні запитання

1. Які види діаграм взаємодії (interaction diagram) ви знаєте?
2. Для чого призначена діаграма послідовності (sequence diagram)? Наведіть її основні елементи.
3. Дайте визначення об'єктів діаграми послідовності, наведіть їх основні характеристики.
4. Дайте визначення повідомлень діаграми послідовності, перелічіть основні типи повідомлень.
5. Для чого використовуються прямі, зворотні та рефлексивні повідомлення?
6. Яким чином на діаграмі послідовності використовуються актори?
7. Що таке лінія життя об'єкта? Чим визначається початковий і кінцевий момент періоду життя кожного об'єкта?
8. Для чого призначена діаграма кооперації (collaboration diagram)? Наведіть її основні елементи.
9. В яких випадках зручніше використовувати діаграму послідовності?
10. В яких випадках зручніше використовувати діаграму кооперації?
11. Як діаграму послідовності можна перетворити в діаграму кооперації?

Практична робота № 5
ПОБУДОВА ДІАГРАМ КЛАСІВ

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.

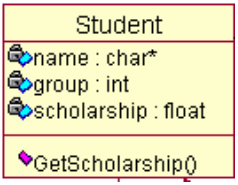


Рис. 7. Графічне представлення класу на діаграмі класів у Rational Rose

У Rational Rose кожен клас графічно представлений прямокутником, що має три секції: ім'я класу, перелік атрибутів та перелік операцій (рис. 7). 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).

Створення нових класів та операцій класів:

У Rational Rose для кожного об'єкта на діаграмах взаємодії існує можливість призначити певний клас (чи створити новий). При цьому в прямокутнику об'єкта дані відображаються в наступному форматі <ім'я об'єкта>:<ім'я класу>. Також кожне повідомлення на діаграмі взаємодії представляє певну операцію класу-приймача. Для кожного повідомлення можна обрати існуючий метод класу-приймача чи створити новий. На рис. 8 наведено фрагмент модифікованої діаграми послідовності для варіанта використання «Переглянути список студентів».

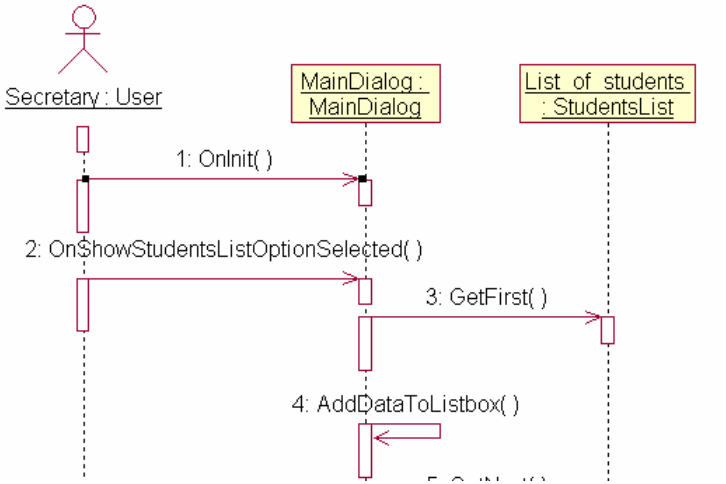


Рис. 8. Фрагмент діаграми послідовності для варіанта використання «Переглянути список студентів»

Після створення набору класів з операціями можна відобразити їх на діаграмі класів. При цьому кожен клас буде мати ім'я та перелік операцій, визначених користувачем. На рис. 9 показано відображення класу MainDialog з операціями, створеними для повідомлень 2 та 4 попередньої діаграми (повідомлення 1 буде показано пізніше на загальному вигляді діаграми класів, оскільки воно визначено для класу Dialog, який є базовим для даного класу).

Рис. 9. Відображення класу MainDialog на діаграмі класів

Також нові класи та операції класів можна створювати безпосередньо з діаграми класів.

Додавання атрибутів класів:

Після створення переліку класів із набором операцій необхідно для кожного класу створити набір атрибутів – даних, якими оперує програма у процесі виконання операцій класу. Наприклад, виходячи з діаграми послідовності, зображеної на рис. 10 можна визначити для класу MainDialog атрибути для кнопки (Button), яку натискає користувач для перегляду списку студентів та списку, до якого безпосередньо додається інформація про кожного студента.

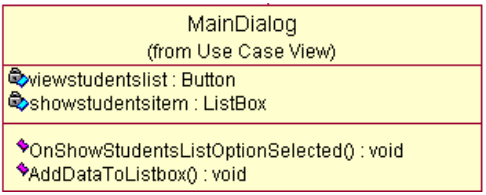


Рис. 10. Клас MainDialog з повним набором атрибутів та операцій

Повний формат відображення атрибутів на діаграмі класів:
<attribute_name>:<attribute_type>
Повний формат відображення операцій на діаграмі класів:
<operation_name>(<list_of_arguments>):<type_of_return_value>
Відношення між класами:

Rational Rose для нотації Unified можна задати 5 основних типів *відношень* між класами:

1. Відношення асоціації (***association***) – визначає абстрактний зв'язок між класами, може представляти аналог відношення «m до n».
   Приклад: відношення між класами Student та Course (кожен студент відвідує декілька курсів (дисциплін), для кожної дисципліни визначений набір студентів, які її відвідують). ***Представлення в програмному коді:***
   class Student
   { .....
   Course* theCourses;
   .....};

```
class Course
{.....................
    Student* theStudents;
.....................};
```

При цьому Course* в класі Student може вказувати на масив дисциплін, а Student* у класі Course – на перелік студентів.

2. Відношення агрегації (***aggregation***) – відношення типу «частина – ціле», при якому час життя класа частини не співпадає з часом життя класа цілого.

   Приклад: відношення між класами Student та Ticket (у час переоформлення студентського квитка студент не має його).

   Представлення у програмному коді:

```
class Student
{ .............
    Ticket* ticket;
.............};
```

   Атрибут ticket при цьому може створюватися та видалятися (за допомогою операторів new та delete) в коді будь-яких методів класу Student.

3. Відношення композиції (***composition***) – відношення типу «частина – ціле», при якому час життя класа частини співпадає з часом життя класа цілого.

   Приклад: відношення між класами Student та Date_of_Birth (кожен студент у будь-який момент часу характеризується датою народження).

   Представлення у програмному коді:

```
class Student
{ .............
    Date_of_Birth theDate_of_Birth;
.................}
```

   Атрибут theDate_of_Birth існує протягом усього часу життя об'єкта класу Student.

4. Відношення наслідування (***generalization***) – відношення типу «загальне – часткове».

   Приклад: відношення між класами Student та Astudent (студент відмінник є частковим випадком студента).

   Представлення в програмному коді:

```
class Student ;
class Astudent : public Student{......
    float GetScholarBonus();
.........};
```

Клас Astudent має додатковий метод GetScholarBonus(), який розраховує надбавку до стипендії для студента відмінника.

5. Відношення інстанціювання (instantiation) – визначає інстанціювання нового класу з параметризованого класу шляхом підставлення фактичних параметрів у формальні параметри шаблона.

   Приклад: відношення між класами List та StudentsList (з абстрактного списку інстанціюється список студентів).

   Представлення в програмному коді:

   template <class T> class List {…….} // List – абстрактний список

   typedef List <Student> StudentsList; // List<Student> – список студентів

Для класу, що інстанціюється, генерація програмного коду відсутня, оскільки він генерується компілятором автоматично при зверненні до параметризованого класу (підставлення фактичних параметрів у формальні параметри шаблона).

На рис. 11 наведено приклад діаграми класів для системи обліку успішності студентів у деканаті, яка містить перелік класів, пов'язаних різними типами відношень.

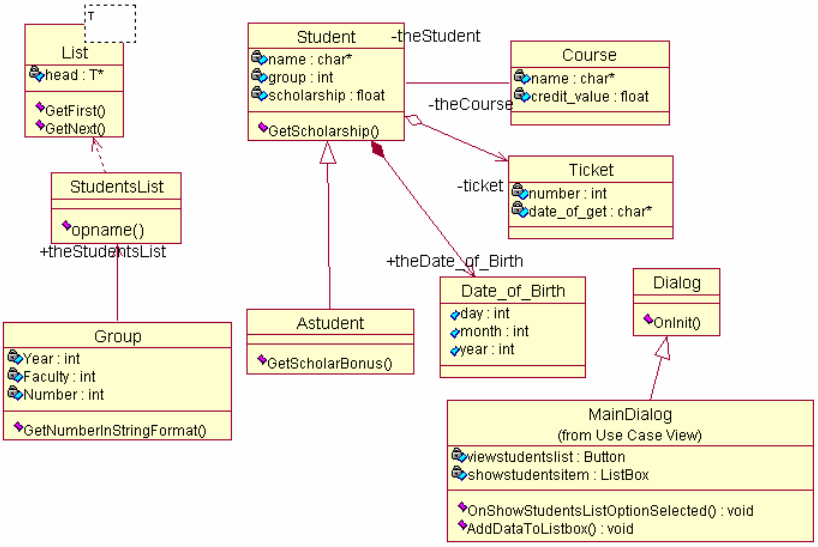


Рис. 11. Діаграма класів для системи обліку успішності навчання студентів у деканаті

Завдання

1. Для всіх об'єктів на діаграмах взаємодії призначити (створити) певний клас; для кожного повідомлення призначити (створити) відповідний метод (операцію) для класу об'єкта-приймача.
2. Розташувати створені класи з переліком операцій на діаграмі класів.
3. Для кожної операції визначити атрибути, які вона використовує та при необхідності додати їх до списку атрибутів класу.
4. Для кожного атрибуту задати логічний тип даних, для кожної операції логічний тип даних для return value та для переліку аргументів, якщо вони присутні.
5. Пов'язати класи на діаграмі класів, використовуючи різні типи відношень (асоціацію, агрегацію, композицію, наслідування, інстанціювання).

Вимоги

1. Діаграма класів повинна містити не менше 10 класів.
2. Для кожного класу визначити не менше 5 атрибутів та 5 операцій.
3. По можливості використати всі типи відношень між класами.

Контрольні запитання

1. Для чого призначена діаграма класів (class diagram)? Назвіть її основні елементи.
2. Дайте визначення класу. Які типи класів ви знаєте?
3. Чим відрізняється абстрактний клас від конкретного?
4. Чим відрізняється абстрактний клас від інтерфейсу?
5. Що таке параметризований клас? Дайте визначення процесу інстанціювання. Наведіть приклад.
6. Назвіть основні характеристики класу.
7. Що таке атрибути класу? Які типи атрибутів ви знаєте?
8. Що таке операції класу? Які типи операцій ви знаєте?
9. Дайте визначення статичних атрибутів та операцій класу.
10. Дайте визначення наслідування. Назвіть основні типи наслідування.
11. Яким чином реалізується інкапсуляція для атрибутів та операцій класу? Назвіть типи модифікаторів доступу до елементів класу.
12. Яким чином реалізується поліморфізм для класів, що знаходяться в єдиній ієрархії? Наведіть приклад.
13. Що таке віртуальні функції, для чого вони призначені?
14. Яким чином можна створити класи та їх операції з діаграм взаємодії?
15. В якому форматі відображають атрибути та операції на діаграмі класів?

16. Перелічіть можливі відношення між класами на діаграмі класів.
17. Які характеристики можна визначити для відношення асоціації?
18. Як відношення асоціації можна перетворити в агрегацію/композицію?
19. У чому відмінність між відношеннями агрегації та композиції?
20. Який клас є базовим, а який похідним при визначенні відношення наслідування?
21. Які типи класів поєднує відношення інстанціювання?
22. Яким чином кожний тип відношень між класами впливає на генерацію програмного коду?

Практична робота № 6
ДІАГРАМИ СТАНІВ ТА ПЕРЕХОДІВ

Теоретичні відомості

Діаграми станів та переходів (statechart diagrams) разом із діаграмами діяльності та взаємодії, відображають певний сценарій, що виконується у процесі функціонування системи в цілому, або певної її частини.

Діаграма станів відображає скінчений автомат у вигляді графу, вершинами якого є стани об'єкта, поведінка якого моделюється, а переходами – події, які переводять об'єкт, який розглядається, з одного стану в інший. При цьому вважається, що час перебування об'єкта в певному стані набагато більший за час, необхідний для переходу з одного стану в інший, тобто переходи між станами здійснюються миттєво.

Стан (state) – це логічна сутність, що використовується для моделювання певної ситуації, дії, процесу. Кожен стан має ім'я та список внутрішніх дій. В якості імені стану найчастіше використовуються іменники, наприклад: «Введення паролю», «Очікування», «Перевірка параметрів». Список внутрішніх дій містить перелік дій, які виконуються у процесі знаходження системи чи об'єкта в даному стані. Кожна дія відображається у форматі:

<період виконання>/<назва дії>,

де поле <період виконання> може набувати наступних значень:

OnEntry – дія виконується під час того, як система входить у даний стан;

OnExit – дія виконується при виході з даного стану;

Do – дія виконується під час знаходження в даному стані;

OnEvent – дія виконується при настанні певної (зовнішньої) події.

Графічне представлення стану «Введення паролю» з трьома внутрішніми діями наведено на рис. 12. Перехід у даний стан ініціюється при введенні користувачем символів логіну. Для кожного введеного символа система зчитує строку логіну, додає додатковий символ до неї і зберігає її.

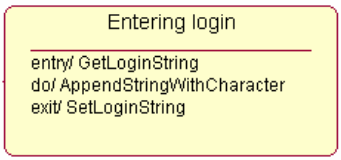


Рис. 12. Графічне представлення стану в середовищі Rational Rose

У Rational Rose існує можливість створити також один із специфічних станів:

1. Вхідний стан – стан, в якому знаходиться система (об'єкт) у початковий момент часу.
2. Вихідний стан – стан, в якому знаходиться система (об'єкт) в момент закінчення виконання певної послідовності дій.
3. Стан історії – стан, який запам'ятовує дані, що використовувалися при попередньому входженні системи в даний стан.

Рис. 13. Представлення початкового та кіцевого станів на statechart diagrams

Рис. 14 демонструє стан історії «Перевірка параметрів дзвінка». Після перевірки параметрів перед здійсненням виклику запам'ятовується номер абонента для можливості його повторного виклику в майбутньому.

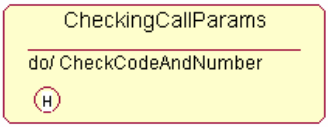


Рис. 14. Графічне представлення стану історії

Переходи (transitions) на statechart diagrams представлені стрілкою, що виходить з попереднього стану і входить у наступний. Кожен перехід має наступну специфікацію:

<тригер>(<параметри>)[<гранична умова>]/<дія>,

де <тригер> – подія, що ініціює можливість переходу;

(<параметри>) – параметри події;

[<гранична умова>] – умова, необхідна для здійснення переходу;

<дія> – дія, що виконується у процесі переходу.

На statechart diagram можна задати два типи переходів:

1. Звичайний – перехід з одного стану в інший.
2. Рефлексивний – перехід із даного стану в цей же стан (зображається у вигляді петлі на графі).

На рис. 15 представлений фрагмент діаграми станів і переходів із застосуванням різних типів переходів для системи «Міні-АТС». Перехід із початкового стану у стан «Очікування» відбувається при ввімкненні системи, за умови її успішної ініціалізації. При цьому виконується дія

Tone (подання тонового сигналу). Рефлексивний перехід для стану введення коду міста виникає при введенні нової цифри, параметр int d символізує код цифри.

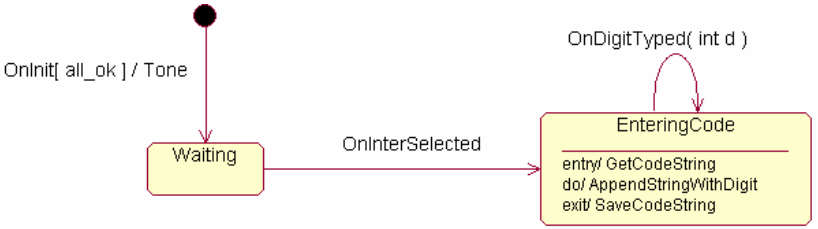


Рис. 15. Фрагмент діаграми станів та переходів для Міні-АТС

Приклади діаграм:

На рис. 16 зображено приклад діаграми станів та переходів для конкретного об'єкта (діалога авторизації певної системи). Рис. 17 представляє діаграму станів та переходів для Міні-АТС.

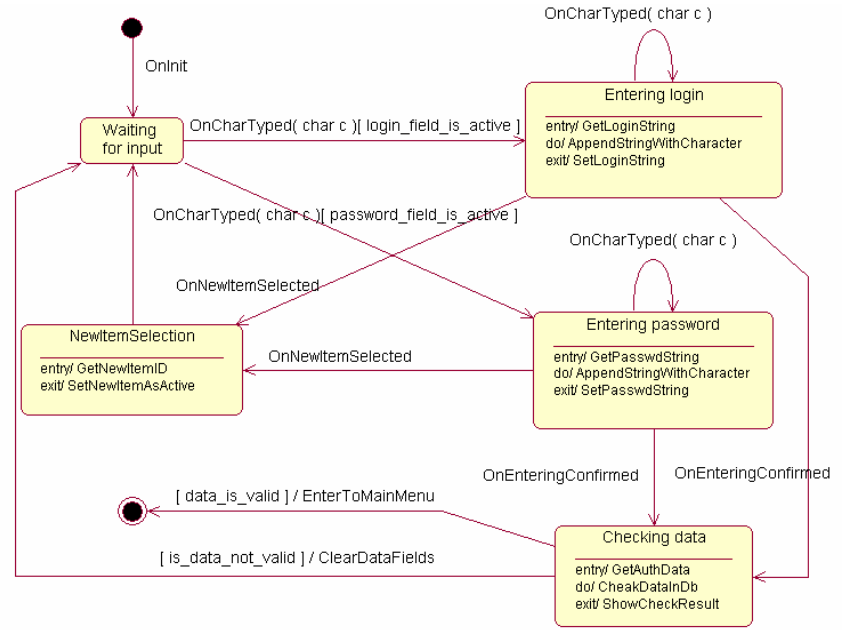


Рис. 16. Приклад діаграми станів та переходів для авторизації користувача в системі

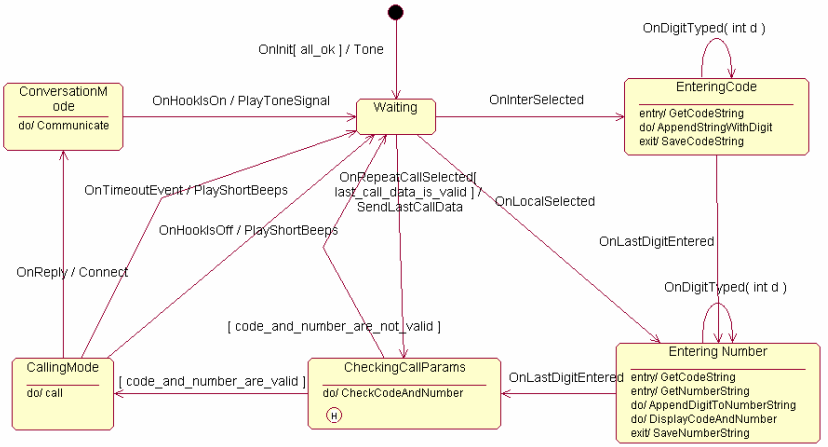


Рис. 17. Приклад діаграми станів та переходів для системи «Міні-АТС»

Завдання

Створити одну діаграму станів для опису процесу функціонування обраної системи в цілому і дві діаграми для конкретних елементів системи. Використовувати діаграму станів для авторизації користувачів забороняється.

Вимоги

1. Кожна діаграма повинна містити не менше 6 станів.
2. По можливості використати обидва типи переходів (звичайний і рефлексивний).
3. Для кожного переходу визначити хоча б одну з характеристик (тригер, гранична умова, дія).

Контрольні запитання

1. Для чого призначена діаграма станів та переходів (statechart diagram)?
2. Назвіть основні елементи statechart diagram.
3. Що таке стан? Перелічіть список тригерів для внутрішніх дій стану.
4. Перелічіть специфічні стани та наведіть їх призначення.
5. Для чого призначені переходи на statechart diagram?
6. Які характеристики властиві для кожного переходу? Чи всі вони обов’язкові?
7. Які типи переходів існують на діаграмі? Чим вони відрізняються?

Практична робота № 7
ПОБУДОВА ДІАГРАМ ДІЯЛЬНОСТІ

Теоретичні відомості

Діаграми діяльності (**activity diagrams**) відображають послідовність дій, що виконується в процесі реалізації певного варіанта використання або функціонування системи в цілому. Діаграми діяльності є аналогом блок-схеми будь-якого алгоритму. Вони, як і діаграми станів та переходів, відображаються у вигляді орієнтованого графу, вершинами якого є дії, а ребрами – переходи між діями.

Діяльність (activity) є частковим випадком стану (state) без назви, який має одну вхідну подію (OnEntry action). Тому для кожної діяльності назва складається з дієслова та декількох пояснюючих слів, наприклад «Розрахувати заробітну платню» чи «Перевірити результати запиту». Графічне зображення діяльності «Перекласти слово» подано на рис. 18.

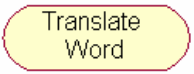


Рис. 18. Графічне представлення елементу Activity на діаграмі діяльності

Події (events) на переходах діаграми діяльності не задаються, оскільки вважається, що перехід від однієї дії до іншої здійснюється безумовно. Гранична умова (guard condition) використовується лише для визначення дії, до якої переходить керування у випадку неоднозначності (рис. 19). Тобто, якщо з даної вершини на діаграмі діяльності можна перейти до декількох інших вершин для всіх переходів необхідно визначити граничну умову. Характеристика дії (action) для переходу також не має сенсу, оскільки всі дії на цій діаграмі представлені вершинами графу.

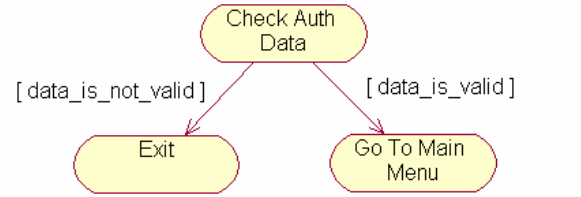


Рис. 19. Фрагмент діаграми діяльності для процесу авторизації

Для діаграми діяльності характерними є наступні спеціальні стани:

1. Початковий стан – аналогічний до діаграми станів та переходів.
2. Кінцевий стан – аналогічний до діаграми станів та переходів.
3. Стан прийняття рішення – стан, в якому здійснюється прийняття рішення про перенаправлення потоку управління до одного зі станів, пов'язаних із даним станом.
4. Стан синхронізації – стан, в якому здійснюється розділення загального потоку управління на декілька гілок (чи навпаки, декілька гілок поєднуються в єдиний поток).

Спеціальні стани прийняття рішення та синхронізації представлені на рис. 20а та 20б відповідно.

Рис. 20. Графічне представлення спеціальних станів діаграми діяльності

Розглянемо застосування спеціальних станів на конкретному прикладі (варіант використання «Перекласти слово» для електронного словника). На рис. 21 показано повний вигляд діаграми діяльності. На початковому етапі перекладу обирається словник, користувач вводить слово та система робить його переклад, далі з використанням стану прийняття рішення визначається чи обрані додаткові опції перекладу. У випадку, якщо опцій не обрано, система переходить до показу результату перекладу. В іншому випадку, поток керування розподіляється на дві гілки, кожна з яких виконує певну дію (отримати транскрипцію слова та список синонімів відповідно). Після закінчення виконання обох операцій дві гілки поєднуються в єдиний потік і здійснюється показ результату. Потім визначається чи потрібен друк для отриманої інформації, і у випадку необхідності вона друкується.

Також діаграми діяльності використовуються для відображення послідовності дій при моделюванні бізнес-процесів. При цьому використовується додатковий елемент діаграми, який має назву swimlane. Swimlane в дослівному перекладі означає дорожка плавального басейну (за аналогією з графічним відображенням). В якості swimlanes на діаграмі можуть виступати фізичні особи, групи осіб, відділи підприємства, чи навіть окремі організації. Розглянемо діаграму діяльності зі swimlanes для спрощеного варіанта бізнес-процесу «Розробка програмного забезпечення» (рис. 22).

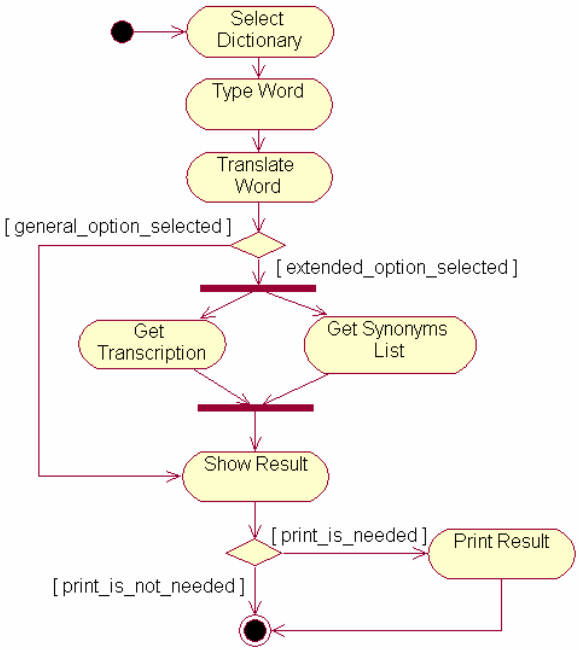


Рис. 21. Діаграма діяльності для перекладу слова електронним словником

В якості swimlanes в даній діаграмі виступають наступні особи (групи осіб):

1) аналітик, який розробляє вимоги до проекту;
2) керівник проекту, який складає план виконання робіт;
3) системний інженер, що проектує систему;
4) група розробників, які створюють програмний код;
5) група тестувальників, які формують варіанти тестування та тестують створену систему;
6) група впровадження, яка поставляє систему кінцевому користувачу та здійснює підтримку.

Завдання

Побудувати 3 діаграми діяльності для окремих варіантів використання системи.

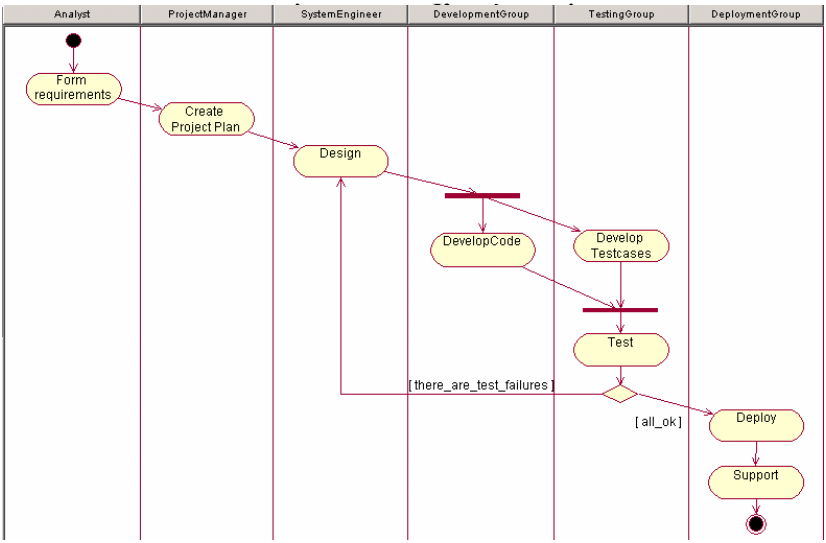


Рис. 22. Використання діаграми діяльності для відображення бізнес-процесів

Вимоги

1. Кожна діаграма повинна містити не менше 6 діяльностей.
2. При побудові кожної діаграми використовувати стани прийняття рішення та синхронізації.

Контрольні запитання

1. Для чого призначена діаграма діяльності (activity diagram)? Назвіть її основні елементи.
2. У чому відмінність між діаграмою діяльності і діаграмою станів та переходів?
3. Що таке діяльність? Чим діяльність відрізняється від стану?
4. Які характеристики властиві для переходу на діаграмі діяльності.
5. Що таке стан прийняття рішення?
6. Що таке стан синхронізації?
7. Для чого використовується елемент swimlane?

Практична робота № 8
ПОБУДОВА ДІАГРАМ КОМПОНЕНТІВ

Теоретичні відомості

Діаграми компонентів (component diagrams) відображають фізичне представлення програмної системи у вигляді сукупності елементів, які називають компонентами *(components).* Кожен компонент має ім'я, мову реалізації та перелік призначених класів. Фізично кожен компонент може бути представлений у вигляді окремого файлу, директорії чи програмного продукту. Графічний вигляд компоненту наведено на рис. 23.

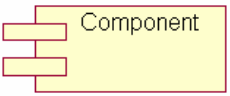


Рис. 23. Графічне представлення компоненту в Rational Rose

У залежності від обраного стереотипу компонент може належати до одного з наступних типів: Applet, Application, Database, DLL, EXE, MainProgram, SubProgram. Тип компонента є частиною його графічного представлення і не впливає на генерацію програмного коду.

Одним з елементів мови UML є також пакет (package). Головною відмінністю компонентів від пакетів є те, що пакет є логічним об'єднанням класів (аналог namespace в C++) за їх функціональністю, а компонент є окремим фізичним елементом системи, і може виступати в якості прикладної програми, підсистеми, бібліотеки тощо.

Компоненти поєднуються між собою за допомогою відношення залежності (dependency) відповідно до відношень між класами, що належать до цих компонентів.

Розглянемо побудову діаграми компонентів, виходячи з діаграми класів абстрактної системи. На рис. 24 відображено діаграму класів програмної системи, яка має так звану тришарову архітектуру.

Перший шар (позначений синім кольором у лівій частині діаграми) складається з сукупності класів, які формують інтерфейс користувача. До нього належить ієрархія діалогів системи та клас прав користувача, що використовується в діалозі авторизації.

Другий шар (сукупність класів у центральній частині діаграми, обмежена лінією зеленого кольору) відповідає за бізнес-логіку (перелік алгоритмів, що оперують даними для виконання функцій системи). Він сформований сукупністю класів алгоритмів та класом прав користувача на виконання тих чи інших операцій.

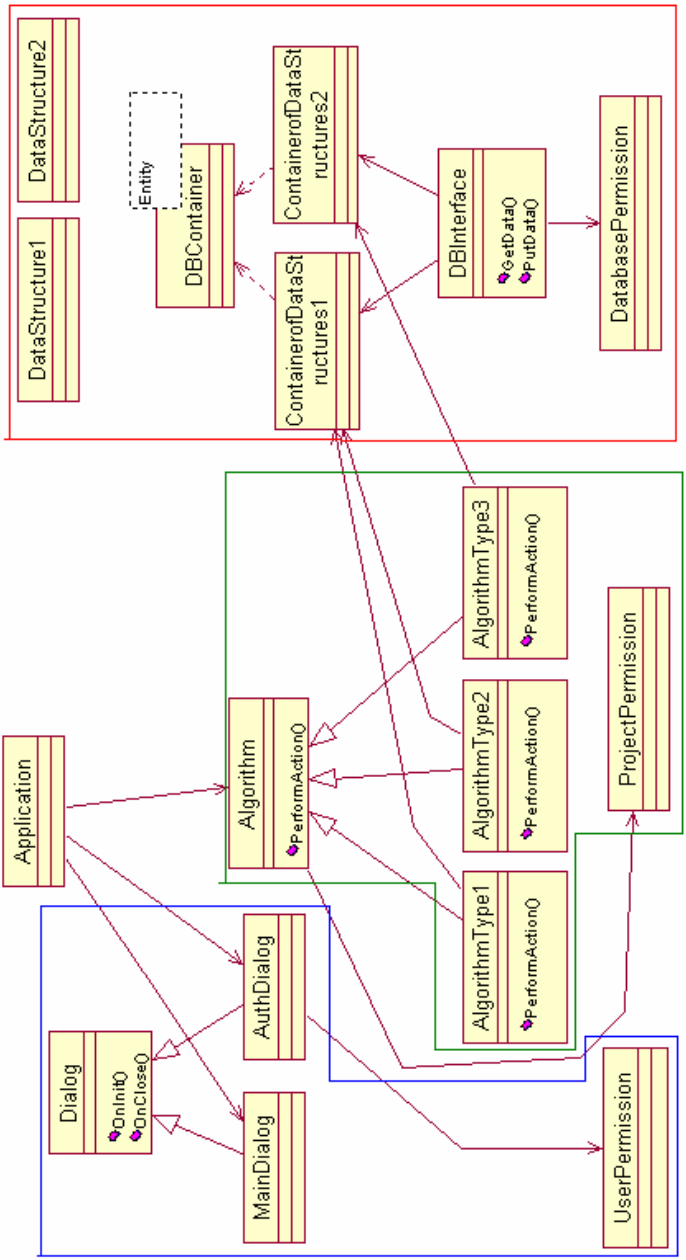


Рис. 24. Приклад діаграми класів для програмної системи з тришаровою архітектурою

Третій шар (сукупність класів у правій частині діаграми, обмежена лінією червоного кольору) забезпечує обмін даними з базою даних. Він складається зі специфічних структур даних, кожна з яких відповідає рядку відповідної таблиці бази даних, контейнерів структур даних, класу інтерфейсу бази даних та прав доступу до таблиць БД.

Програмний комплекс є поєднанням цих трьох шарів, кожен з яких може бути представленим окремою бібліотекою, які можуть бути реалізовані різними мовами програмування. На рис. 25 представлена діаграма компонентів для даного програмного комплексу.

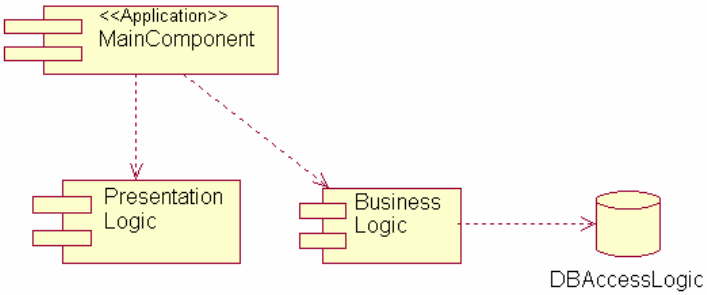


Рис. 25. Діаграма компонентів програмного комплексу

Розподіл класів за компонентами:

1. MainComponent має стереотип Application та містить єдиний клас Application.
2. Presentation Logic містить класи Dialog, MainDialog, AuthDialog, UserPermission.
3. BusinessLogic містить класи Algorithm, AlgorithmType1, AlgorithmType2, AlgorithmType3, ProjectPermission.
4. DBAccessLogic має стереотип Database та містить класи DBContainer, DBInterface, DatabasePermission, DataStructure1, DataStructure2, ContainerofDataStructures1, ContainerofDataStructures2.

Зверніть увагу, що класи UserPermission, ProjectPermission та DatabasePermission можуть належати до одного пакету Security діаграми класів, але реалізуються в різних компонентах. Тобто, незважаючи на схожість пакетів та компонентів, вони можуть не бути ідентичними.

Послідовність дій для призначення класів до компонентів:

1. У контекстному меню одного з компонентів обрати OpenSpecification.
2. Відкрити вкладку Realizes та призначити класи один за одним, натискаючи праву кнопку миші та обираючи пункт Assign.

Відношення залежності на діаграмі компонентів визначається відповідно до відношень класів компонентів на діаграмі класів. Клас Application пов'язаний відношенням направленої асоціації з класами компонентів PresentationLogic та BusinessLogic (залежить від них), тому між MainComponent та обома даними компонентами визначається відношення залежності (позначається штриховою стрілкою). Аналогічне твердження справедливе для пари компонентів BusinessLogic та DBAccessLogic.

Завдання

У середовищі Rational Rose створити діаграму компонентів для обраної програмної системи. Діаграма повинна містити не менше трьох компонентів. Розподілити всі класи між компонентами.

Контрольні запитання

1. Для чого призначена діаграма послідовності (sequence diagram)? Наведіть її основні елементи.
2. Дайте характеристику компонента. Що може виступати в ролі компонента? Наведіть приклад.
3. Назвіть головну відмінність між пакетами та компонентами.
4. Що таке 3 tier архітектура? Наведіть приклад системи, для якої можна використати дану архітектуру.
5. Які характеристики визначаються для кожного компонента на діаграмі?
6. Яким чином класи розподіляються за компонентами?
7. Яким чином визначаються відношення між компонентами?

Практична робота № 9
ПОБУДОВА ДІАГРАМ ВПРОВАДЖЕННЯ, КОДОГЕНЕРАЦІЯ ТА ЗВОРОТНЕ ПРОЕКТУВАННЯ

Теоретичні відомості

Діаграми впровадження ***(deployment diagrams)*** представляють фізичну модель предметної області й відображають апаратну частину реалізації програмного комплексу. Головними елементами діаграми впровадження є процесори *(processors)*, пристрої *(devices*) та зв'язки між ними *(connections).*

В якості процесорів на діаграмі впровадження відображаються апаратні комплекси, сервери, робочі станції, контролери, тобто пристрої, що мають CPU. В якості пристроїв найчастіше виступає допоміжне обладнання, таке як пристрої вводу виводу, мережеве обладнання, таймери, датчики тощо. Графічне зображення процесорів та пристроїв наведено на рис. 26.

Рис. 26. Графічне зображення процесорів та пристроїв на діаграмі впровадження

Процесори та пристрої на діаграмі впровадження поєднуються зв'язками в залежності від їх фізичного поєднання в реальному світі. Розглянемо діаграму впровадження для абстрактного програмного комплексу з практичної роботи № 8. Програмний копмлекс має тришарову архітектуру. У якості першого шару виступає робоча станція з пристроями вводу-виводу; другий шар є веб-сервером, де виконується бізнес логіка програми; третім шаром виступає сервер баз даних, доступ до якого відбувається з веб-сервера (рис. 27).

Кодогенерація (code generation) – це процес автоматизованого перетворення елементів діаграм UML у програмний код.

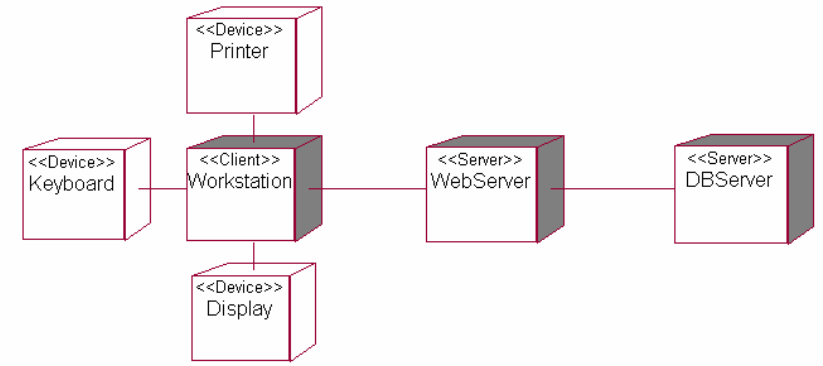


Рис. 27. Приклад діаграми впровадження для програмного комплексу з тришаровою архітектурою

Необхідними умовами проведення генерації програмного коду є наявність діаграми класів з переліком класів, які в сукупності формують програмний комплекс, а також діаграми компонентів, між якими дані класи розподілені. Кожен компонент повинен мати визначену мову реалізації. Послідовність кроків для проведення кодогенерації залежить від обраної мови програмування.

Перед проведенням кодогенерації необхідно впевнитись у тому, що для всіх атрибутів та операцій визначені типи даних, що властиві для мови програмування, визначеної у компоненті, до якого належить даний клас. Усі логічні типи даних (наприклад, такі як Integer, Boolean, String) мають бути замінені на типи даних відповідної мови програмування (у випадку C++ на int, bool, та char* відповідно).

Послідовність кроків для проведення кодогенерації з використанням ANSI C++

1. Визначити типи даних властиві для мови програмування C++ для всіх атрибутів та операцій всіх класів.
2. На жорсткому диску створити директорію для проведення кодогенерації.
3. На діаграмі компонентів для одного з компонентів у контекстному меню обрати пункт ANSI C++ / Open ANSI C++ Specification.
4. Задати параметри для генерації програмного коду (директорія для кодогенерації, директорії для h та cpp файлів, стиль для блоків коду, наявність коментарів, генерацію backup файлів тощо).
5. Для даного компоненту в контекстному меню обрати пункт ANSI C++ / Generate Code.

6. Вибрати зі списку класів, наведених у ListControl ті, для яких необхідно генерувати програмний код та натиснути Ок.
7. Програмний код буде згенеровано, а список помилок та попереджень виведено у лог.

Програмний код для конкретного класу можна згенерувати безпосередньо з діаграми класів, обравши в контекстному меню даного класу пункт ANSI C++ / Generate Code. За допомогою контекстного меню ANSI C++ / Class customization можна додати до класу специфічні операції (конструктори, деструктори, різноманітні оператори, селектори і модифікатори). Переглянути згенерований програмний код можна за допомогою команди ANSI C++ / Browse header та ANSI C++ / Browse body з використанням вбудованого редактора Rational Rose, а також із використанням будь-якого іншого текстового редактора безпосередньо в заданій для кодогенерації директорії.

Послідовність кроків для проведення кодогенерації з використанням Visual C++

1. Визначити типи даних, властиві для мови програмування C++ для всіх атрибутів та операцій усіх класів.
2. На діаграмі компонентів для одного з компонентів у контекстному меню обрати пункт Update Code.
3. У вікні Code Update Tool обрати ті компоненти (чи окремі класи), для яких необхідно згенерувати код.
4. Якщо компонент не було призначено до жодного проекту VC++ відкриється нове вікно, в якому необхідно задати шлях до файла workspace проекта. Натиснувши ..., можна призначити компонент до існуючого проекта (Existing), чи створити новий (New). Після цього необхідно обрати класи чи компоненти для кодогенерації ще раз.
5. Натиснути Next, і на екрані з'явиться перелік класів, які будуть згенеровані.
6. Натиснути Finish, у вікні Code Update Tool буде відображено лог повідомлень, серед яких при наявності будуть вказані помилки та попередження, що виникли у процесі кодогенерації.

Кодогенерацію можна виконати також для окремого класу на діаграмі класів, якщо для даного класу обрати в контекстному меню пункт Update Code. Для редагування класу можна скористатися інструментом Model Assistant (обравши пункт Model Assistant контекстного меню), який дозволяє додавати до класу атрибути, операції; створювати конструктори, деструктори, різноманітні оператори, селектори і модифікатори.

Зворотнє проектування (***reverse engineering***) – процес оновлення моделі проекту Rational Rose зі створеного, існуючого чи модифікованого програмного коду.

Генерація програмного коду є стартовою точкою розробки програмного забезпечення. В залежності від якості виконаного проектування у процесі розробки доводиться декілька разів змінювати дизайн із метою врахування додаткових вимог чи виправлення певних помилок. У процесі розробки кількість програмного коду, класів, інтейфесів, операцій та атрибутів критично зростає. Для відповідності моделі до програмного коду, який постійно змінюється виконується процес зворотнього проектування.

Послідовність кроків для проведення зворотнього проектування з використанням ANSI C++

1. Модифікувати програмний код, що було згенеровано попередньо (додати нові чи видалити існуючі атрибути, операції, параметри) для певного переліку класів.
2. На діаграмі компонентів (чи діаграмі класів) обрати компонент (клас), для якого необхідно провести процес зворотнього проектування.
3. У контекстному меню визначеного об'єкта (об'єктів) обрати пункт ANSI C++ / Reverse Engineer.
4. Модель Rational Rose буде приведено у відповідність до існуючого програмного коду.

Послідовність кроків для проведення зворотнього проектування з використанням Visual C++

1. Модифікувати програмний код, що було згенеровано попередньо (додати нові чи видалити існуючі атрибути, операції, параметри) для певного переліку класів.
2. На діаграмі компонентів (чи діаграмі класів) обрати будь-який компонент (клас).
3. У контекстному меню визначеного об'єкта (об'єктів) обрати пункт Update Model.
4. Натиснути Next, у вікні Model Update Tool обрати компоненти (класи), для яких необхідно провести процес зворотнього проектування. Натиснути Next.
5. У вікні Model Update Tool буде відображатися список класів, для яких буде виконано процес зворотнього проектування. Натиснути Finish.
6. Модель Rational Rose буде приведено у відповідність до існуючого програмного коду. Відомості про попередження чи помилки у процесі виконання зворотнього проектування будуть занесені в лог.

Завдання

1. Створити в середовищі Rational Rose діаграму впровадження для обраного програмного продукту.
2. Задати для всіх атрибутів та операцій типи даних відповідно до обраної мови реалізації програмного продукту.
3. Провести кодогенерацію відповідно до обраної мови реалізації.
4. У разі необхідності (генерація з використанням ANSI C++) створити новий проект у середовищі Visual C++ та додати до нього файли згенерованого коду.
5. Провести компіляцію програмного коду, виправити помилки процесу кодогенерації.
6. Додати до довільно обраного класу перелік атрибутів та операцій.
7. Перевірити вплив процесу зворотнього проектування на модель проекту.

Вимоги

1. Діаграма впровадження повинна мати не менше 5 елементів.
2. Згенерований код має містити всі класи, присутні на діаграмі класів.
3. Модель Rational Rose і програмний код мають повністю відповідати один одному.
4. При компіляції програмного коду не повинно виникати помилок та попереджень.

Контрольні запитання

1. Для чого призначена діаграма впровадження (deployment diagram)? Наведіть її основні елементи.
2. У чому відмінність між процесорами та пристроями? Наведіть приклади для кожних з них.
3. Яким чином поєднуються елементи на діаграмі впровадження.
4. Дайте визначення процесу кодогенерації.
5. Що є необхідною умовою проведення кодогенерації?
6. Які UML діаграми впливають на генерацію програмного коду?
7. Як провести кодогенерацію, використовуючи стандарт ANSI C++?
8. Як провести кодогенерацію, використовуючи середовище MS Visual C++ 6.0?
9. Дайте визначення процесу зворотного проектування.
10. Як провести зворотне проектування, використовуючи стандарт ANSI C++?
11. Як провести зворотне проектування, використовуючи середовище MS Visual C++ 6.0?

Практична робота № 10
РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА

Теоретичні відомості

Головним завданням студентів у практичних роботах № 10-12 є створення працездатного програмного продукту на основі проекту, розробленого в середовищі Rational Rose під час виконання практичних робіт № 3-9.

Створення інтерфейсу користувача на початку детальної розробки програмного продукту дозволяє розширити уявлення замовника про майбутній програмний продукт та усунути певні незручності в його процесі експлуатації користувачами.

Завдання

1. Використовуючи середовище MS Visual C++, створити проект програмного продукту (Single Document, Multi Document, Dialog Based application на вибір).
2. Створити перелік діалогів, форм, вікон для забезпечення виконання функцій, визначених у практичній роботі № 3 (usecases).
3. Для кожного діалогу визначити перелік елементів управління (controls) для виконання специфічного варіанта використання. Передбачити можливість виклику діалогу за допомогою відповідного елементу управління (Button, Menu Item).
4. Для кожного елементу управління (EditBox, ListBox, ListControl, ComboBox) задати початкове абстрактне значення (список значень).

Приклад виконання роботи для системи продажу ліків у аптеці

У середовищі Visual C++ було створено Dialog Based проект. У процесі розробки інтерфейсу користувача створено перелік наступних діалогів:

1. ***Діалог авторизації*** (рис. 28) виконує варіант використання «Авторизація» для всіх акторів системи. Користувач вводить свій логін та пароль у відповідні поля діалогу і підтверджує їх натисканням кнопки Enter. При позитивному проходженні з'являється головний діалог системи, при негативному виводиться помилка і система просить користувача повторити введення. Кнопка Cancel ініціює вихід із системи.

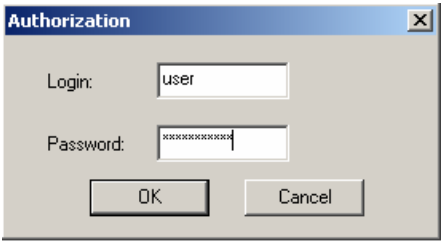


Рис. 28. Діалог авторизації системи продажу ліків

2. ***Головний діалог*** (рис. 29) виконує варіант використання «Переглянути список ліків» та дозволяє викликати діалог адміністрування системи (кнопка Administration), створення нового замовлення (кнопка Create Order) та отримання детальної інформації про ліки (кнопка ShowInfo). Головний ListCtrl представляє список ліків, для кожного з яких визначені ідентифікатор (ID), назва (Cure Name), виробник (Producer), одиниця виміру (Items) і наявна кількість одиниць (Avail). CheckBox Show available cures only дозволяє відображати лише ліки, які є в наявності, кнопка Cancel виконує вихід із системи. Кнопка Administration є активною лише для користувачів, які мають відповідні права доступу. Кнопка ShowInfo є активною, лише коли обрано певний елемент у списку ліків.

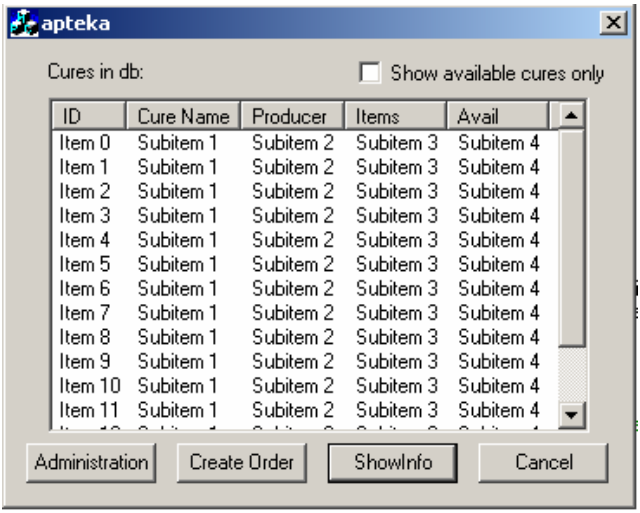


Рис. 29. Головний діалог системи продажу ліків

3. ***Діалог адміністрування*** (рис. 30) призначений для виконання варіантів використання «Переглянути список користувачів», «Додати користувача», «Видалити користувача». Даний діалог активується лише для користувачів, що мають права адміністратора. Для кожного користувача в ListCtrl відображається логін, пароль та права доступу до функціональності системи. Кнопка UserInfo викликає діалог детальної інформації про користувача.

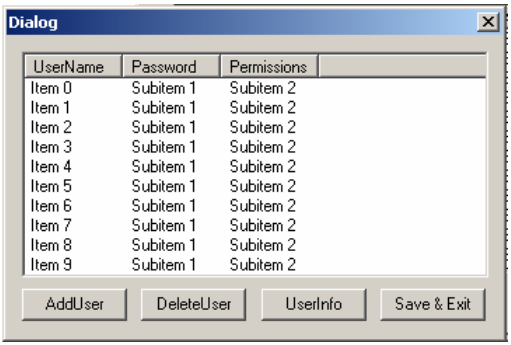


Рис. 30. Діалог адміністрування системи продажу ліків

4. ***Діалог інформації про користувача*** (рис. 31) дозволяє змінити права доступу користувача до функціональності системи. При цьому будь-якому користувачу можна надати як звичайні права користувача, так і адміністратора.

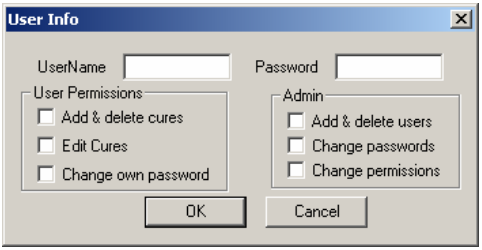


Рис. 31. Діалог інформації користувача системи продажу ліків

5. ***Діалог створення замовлення*** (рис. 32) дозволяє створити нове замовлення ліків. При цьому необхідно обрати з переліку ліків (ComboBox) відповідний пункт, визначити кількість та додати даний товар до замовлення кнопкою Add To Order. За допомогою кнопки Delete From Order можна видалити будь-який товар із списку замовлення.

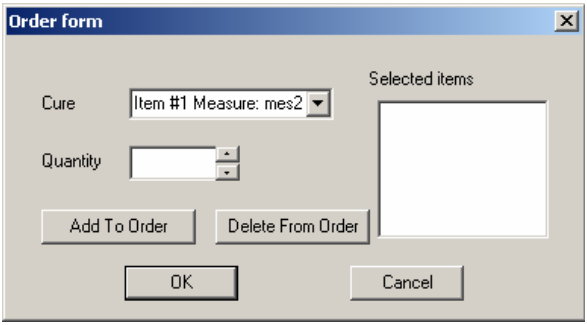


Рис. 32. Діалог замовлення системи продажу ліків

6. ***Діалог детальної інформації про обраний тип ліків*** (рис. 33) відображає в центральному EditBox детальну інформацію про даний тип ліків (назва, виробник, анотація, протипоказання, рекомендації тощо). Діалог відкривається при ініціюванні DoubleClick на будь-якому елементі зі списку ліків або при натисканні кнопки ShowInfo головного діалогу.

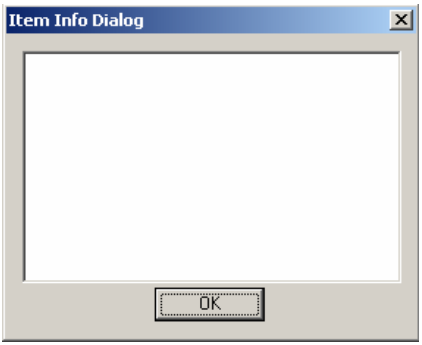


Рис. 33. Діалог інформації про товар системи продажу ліків

7. ***Діалог довідкової інформації*** (рис. 34) відображає автора, версію та дату створення програмного продукту.

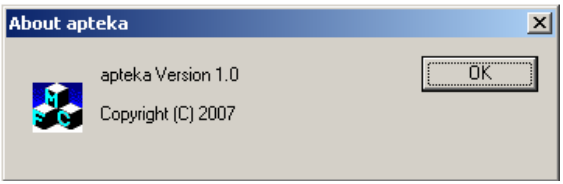


Рис. 34. Діалог адміністрування системи продажу ліків

Вимоги

1. Створити **не менше 5** діалогів (форм, вікон) для виконання різних варіантів використання.
2. Для кожного діалогу визначити **не менше 5** елементів управління.
3. Для кожного елементу управління повинні бути задані початкові значення параметрів, які вони відображають.
4. Кожен елемент управління повинен мати інтуїтивно зрозумілі для користувача назви.

У звіті представити

1. Тему роботи.
2. Завдання.
3. Графічний вигляд створених діалогів.
4. Детальний опис **кожного** діалогу (назва, варіант використання, точка виклику, опис елементів управління).
5. Висновок.

Практична робота № 11
ВЗАЄМОДІЯ ПРОГРАМНОЇ СИСТЕМИ З БАЗОЮ ДАНИХ

Завдання

1. Засобами СКБД (MS Access, MS SQL Server) створити реляційну базу даних із переліком таблиць, описаних у практичній роботі № 1. Для кожного поля таблиці задати відповідний тип даних.
2. Створити драйвер ODBC для бази даних (практична робота № 11 з курсу «СКБД» другого триместру).
3. Використовуючи клас CDatabase, створити підключення до бази даних. Використовуючи клас CRecordset, наповнити даними всі елементи управління, створені під час розробки інтерфейсу користувача. Приклад використання класів CDatabase та CRecordset дивіться нижче. При зчитуванні даних використовувати структури даних (класи), створені при кодогенерації.
4. Реалізувати взаємодію програмного забезпечення з базою даних. Використовуючи функціональність класу Cdatabase, передбачити можливість оновлення інформації (додавання, видалення, редагування) в базі даних.
5. Реалізувати логіку зміни вмісту елементів управління.

Приклад програмного коду для взаємодії з БД

```
void TestOdbcObjects()
{
    CDatabase db; // об’єкт, що представляє базу даних

    // відкриваємо базу даних, вказуючи тип драйвера та шлях до неї
    db.OpenEx(_T("Driver={Microsoft Access Driver (*.mdb)};
DBQ=C:\\Temp\\db1.mdb"));

    // якщо база даних відкрита
    if (db.IsOpen())
    {
        CRecordset set; // Об’єкт, що здійснює послідовний доступ до даних
        set.m_pDatabase = &db; // Приєднуємо set до бази даних

        // Виконуємо insert та update запити, використовуючи об’єкт
бази даних
        db.ExecuteSQL(_T("insert into cures (id, cure) values(36, 'hhhh');"));
```

```
        db.ExecuteSQL(_T("update cures set cure='151515' where id=36;"));

        // Відкриваємо set використовуючи запит 'select * from cures'
        set.Open(CRecordset::forwardOnly, _T("select * from cures"));

        // Проходимо послідовно всі рядки таблиці, що відповідають
запиту в set
        while (!set.IsEOF())
        {
            CString id, cure;

            // Отримуємо значення відповідних стовбців таблиці
            set.GetFieldValue(_T("id"), id);
            set.GetFieldValue(_T("cure"), cure);
            CString out;
            AfxMessageBox((id).GetBuffer(256));
            AfxMessageBox((cure).GetBuffer(256));

            // Переміщуємо вказівник set на наступний рядок таблиці
            set.MoveNext();
        }

        set.Close(); // закриваємо set

        // Виконуємо delete запит, використовуючи об'єкт бази даних
        db.ExecuteSQL(_T("delete from cures where id > 32;"));

        db.Close(); // закриваємо базу даних
    }
}
```

Приклад головного діалогу для системи продажу ліків в аптеці:

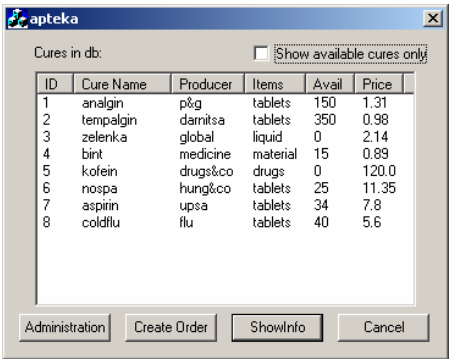


Рис. 35. Головний діалог системи продажу ліків, заповнений даними

Відображення тільки наявних в аптеці товарів:

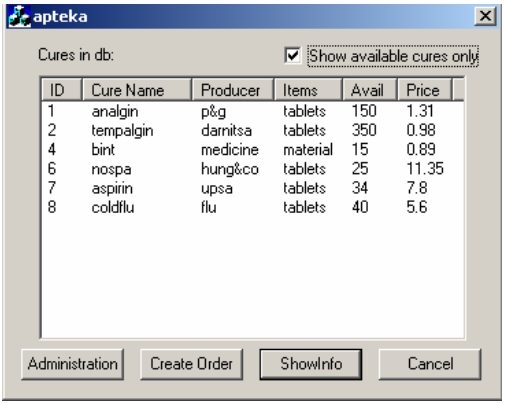


Рис. 36. Головний діалог системи продажу ліків із накладеним фільтром для даних

Вимоги

1. Реляційна база даних має містити не менше 5 таблиць. Кожна таблиця повинна мати не менше 5 полів.
2. Усі елементи управління повинні бути заповнені даними з бази даних.
3. Організувати оновлення даних (додавання, видалення, редагування) в БД для кількості таблиць >= 3.

У звіті представити

1. Тему роботи.
2. Завдання.
3. Перелік таблиць (полів) бази даних у графічному вигляді.
4. Перелік діалогів, заповнених даними.
5. Приклад програмного коду для зчитування даних із БД.
6. Приклад програмного коду для оновлення даних у БД.
7. Висновки.

Контрольні запитання

1. Поясніть структуру створеної реляційної бази даних (наведіть перелік таблиць, поясніть яким чином вони пов'язані).
2. Поясніть, які об'єкти БД, крім таблиць, використовуються у вашій роботі.

3. Для чого використовується клас CRecordset?
4. Для чого використовується клас CDatabase?
5. Напишіть довільний select/update/insert/delete запит (на вибір викладача).
6. Поясніть фрагмент програмного коду з вашої роботи (на вибір викладача).

Практична робота № 12
ГЕНЕРАЦІЯ ВИХІДНИХ ДАНИХ

Теоретичні відомості

Кожна прикладна програма дозволяє через інтерфейс користувача ввести вхідні дані (практична робота № 1), зберегти їх в базі даних чи оновити її (практична робота № 2) з метою отримати результат у вигляді набору вихідних даних. При цьому вихідними даними можуть бути різноманітні типи документів, таблиці, графіки, відео та аудіо файли тощо.

Завдання

1. З використанням OLE Automation для власної системи реалізувати можливість ***програмного*** створення трьох типів вихідних документів:
   - звіту в MS Word (файл .doc);
   - таблиці в MS Excel (файл .xls);
   - слайду в MS PowerPoint (файл .ppt).
2. Програмно реалізувати можливість, переглянути вихідні дані (відкрити файл) та зберегти їх під новим ім'ям.

Порядок виконання роботи

1. На початку методу InitInstance() класу Application проекту додати ініціалізацію COM dll:

```
BOOL CAutomApp::InitInstance()
{
  if (!AfxOleInit()) // Your addition srarts here
  {
      AfxMessageBox("Could not initialize COM dll");
      return FALSE;
  } // End of your addition
  AfxEnableControlContainer();

  // Standart initialization
```

2. Підключити до проекту відповідну OLB бібліотеку.
   Class Wizard \ Automation \ Add Class.. \ From a Type Library..
   Обрати директорію *C:\Program Files\Microsoft Office\Office* та OLB бібліотеку для відповідного програмного продукту:

MS Word	Msword<number>.olb
MS Excel	Excel<number>.olb
MS PowerPoint	Msppt<number>.olb

Де number – номер для різних версій MS Office:

Версія MS Office	Number
MS Office 97	8
MS Office 2000	9
MS Office XP	10
MS Office 2003	11

У діалозі Confirm Classes обрати в ListBox усі класи з отриманого переліку та імпортувати їх у проект. Перелік класів для кожного модуля MS Office помістити в окремий namespace.

3. Виконуючи інструкції в наступних джерелах, автоматизувати створення одного з вихідних документів (приклади програмного коду для автоматизації MS Word, MS Excel та MS PowerPoint наведено нижче):

Модуль MS Office	Назва	Джерело
MS Word	How To Automate Microsoft Word...	http://support.microsoft.com/kb/220911/
MS Excel	How to use MFC to automate Excel...	http://support.microsoft.com/kb/179706/
MS PowerPoint	How to automate PowerPoint...	http://support.microsoft.com/kb/222960/

4. Для отримання інформації про імпортовані класи та їх методи користуватися довідкою: MS Word \ Service \ Macros \ Visual Basic Editor \ Help \ Help On Microsoft Visual Basic \ Microsoft Word Visual Basic Reference \ Microsoft Word Objects.
5. Спроектувати структуру відповідного документа та наповнити його реальними даними з БД.
6. Використовуючи імпортовані класи, реалізувати можливість перегляду створених документів.

Приклад генерації вихідних даних для системи продажу ліків в аптеці

1. MS Word – інформаційний лист до покупця про готовність відправки замовлення.

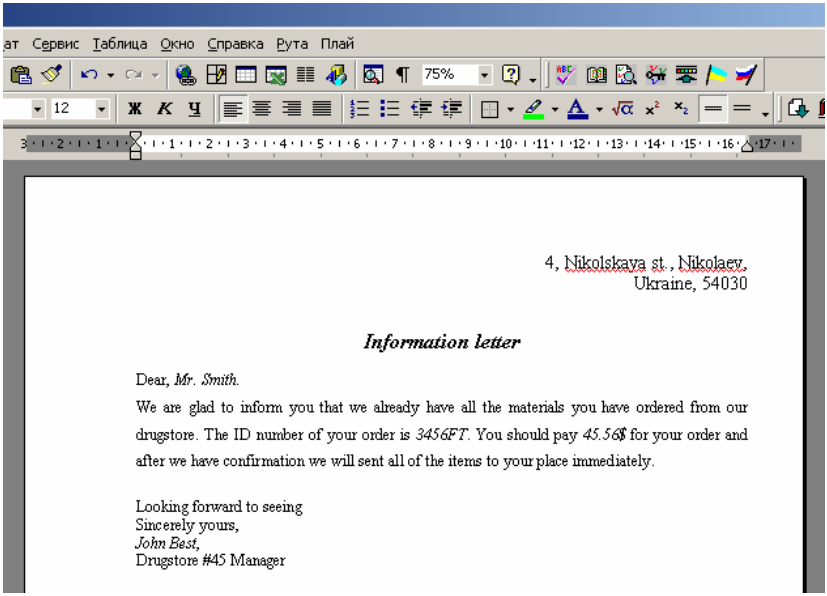
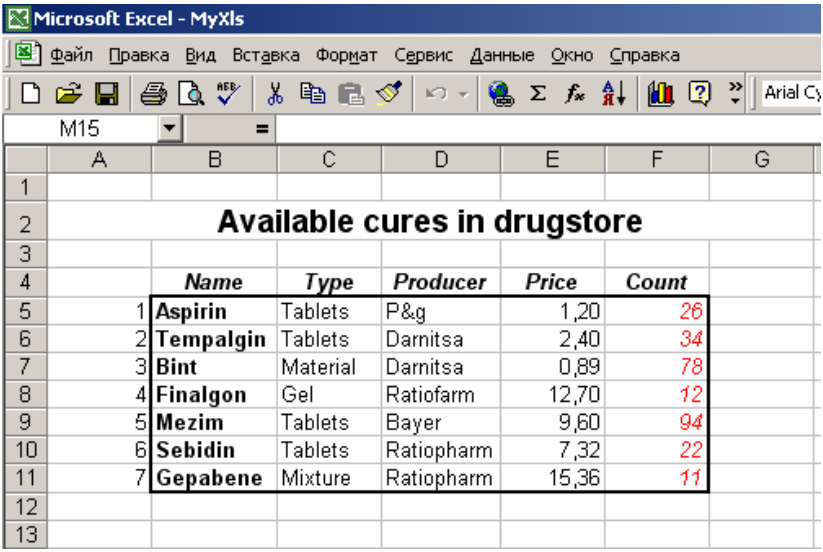


Рис. 37. Згенерований лист до користувача системи продажу ліків

2. MS Excel – таблиця реквізитів, наявних товарів в аптеці.



	A	B	C	D	E	F	G
1							
2		**Available cures in drugstore**					
3							
4		***Name***	***Type***	***Producer***	***Price***	***Count***	
5	1	**Aspirin**	Tablets	P&g	1,20	*26*	
6	2	**Tempalgin**	Tablets	Darnitsa	2,40	*34*	
7	3	**Bint**	Material	Darnitsa	0,89	*78*	
8	4	**Finalgon**	Gel	Ratiofarm	12,70	*12*	
9	5	**Mezim**	Tablets	Bayer	9,60	*94*	
10	6	**Sebidin**	Tablets	Ratiopharm	7,32	*22*	
11	7	**Gepabene**	Mixture	Ratiopharm	15,36	*11*	
12							
13							

Рис. 38. Згенерований звіт про наявність ліків

3. MS PowerPoint – графік продажу товарів (ліків) за перші вісім місяців поточного року

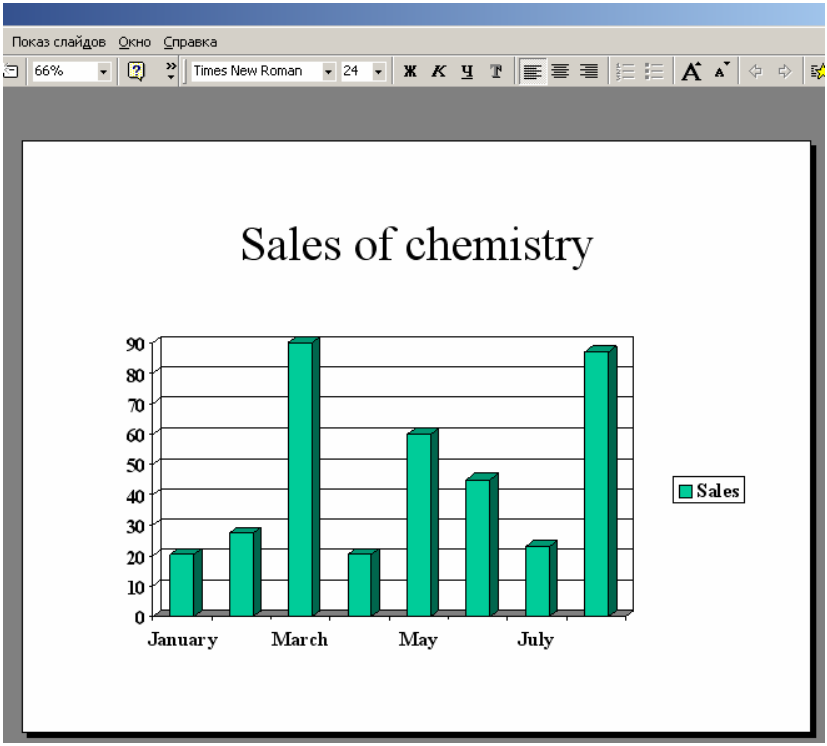


Рис. 39. Згенерований графік продажів за поточний рік

Приклад програмного коду для взаємодії з додатком MS Word:

```
void CAutoDlg::OnMSWord()
{
    using namespace MSWord;

    // класи додатку MS Word з olb бібліотеки
    _Application oWord;
    Documents oDocs;
    _Document oDoc;
    Selection oSelection;
    Paragraphs oParagraphs;
    Range oRange;
```

```
    COleVariant vtOptional((long)DISP_E_PARAMNOTFOUND,VT_ERROR),
        vtTrue((short)TRUE), vtFalse((short)FALSE);

    CString StrToAdd;

    // Запустити MS Word
    if (!oWord.CreateDispatch("Word.Application"))
    {
        // у випадку помилки, вивести повідомлення
        AfxMessageBox("Word failed to start!");
    }
    else
    {
        // показати MS Word
        oWord.SetVisible(TRUE);
        oDocs = oWord.GetDocuments();

        // створити новий документ
        oDoc = oDocs.Add(vtOptional,vtOptional,vtOptional,vtOptional);
        StrToAdd = "Black Sea State University";
        oSelection = oWord.GetSelection(); // отримати поточний курсор
        oParagraphs = oSelection.GetParagraphs();
        oParagraphs.SetAlignment(0); // встановити вирівнювання по лівому
краю
        oSelection.TypeText(StrToAdd); // вставити строку
        oSelection.TypeParagraph(); // вставити новий параграф
        oParagraphs.SetAlignment(1);
        oSelection.TypeText(StrToAdd);
        oSelection.TypeParagraph();
        oParagraphs.SetAlignment(2);
        oSelection.TypeText(StrToAdd);
        oSelection.TypeParagraph();
        COleVariant filename("C:\\Temp\\auto\\doc.doc");

        // зберегти документ
        oDoc.SaveAs( filename,
            vtOptional, vtOptional, vtOptional, vtOptional, vtOptional,
            vtOptional, vtOptional, vtOptional, vtOptional, vtOptional,
            vtOptional, vtOptional, vtOptional, vtOptional, vtOptional);
        oDocs.Close(vtFalse,vtOptional,vtOptional); // закрити всі документи
        oWord.Quit(vtFalse,vtOptional,vtOptional); // вийти з MS Word
        oWord.ReleaseDispatch(); // звільнити dispatch
    }
}
```

Приклад програмного коду для взаємодії з додатком MS Excel:

```
void CAutoDlg::MSExcel()
{
    using namespace MSExcel;
    COleVariant covTrue((short)TRUE), covFalse((short)FALSE),
    covOptional((long)DISP_E_PARAMNOTFOUND, VT_ERROR);
    _Application app;
    Workbooks books;
    _Workbook book;
    Worksheets sheets;
    _Worksheet sheet;
    Range range;
    Font font;

    // Запустити MS Excel (створити dispatch)
    if(!app.CreateDispatch("Excel.Application"))
    {
        // у випадку помилки, вивести повідомлення
        AfxMessageBox("Couldn't start Excel and get Application object.");
    }
    else
    {
        books = app.GetWorkbooks();
        book = books.Add(covOptional); // додати нову книгу

        //Get the first sheet.
        sheets =book.GetSheets();
        sheet = sheets.GetItem(COleVariant((short)1)); // отримати першу
сторінку книги

        // вставити заголовки таблиці в комірки A1, B1, C1, D1
        range = sheet.GetRange(COleVariant("A1"),COleVariant("A1"));
        range.SetValue(covOptional, COleVariant("First Name"));
        range = sheet.GetRange(COleVariant("B1"),COleVariant("B1"));
        range.SetValue(covOptional, COleVariant("Last Name"));
        range = sheet.GetRange(COleVariant("C1"),COleVariant("C1"));
        range.SetValue(covOptional, COleVariant("Full Name"));
        range = sheet.GetRange(COleVariant("D1"),COleVariant("D1"));
        range.SetValue(covOptional, COleVariant("Salary"));
        range = sheet.GetRange(COleVariant("A2"), COleVariant("D2"));
        range.SetValue(covOptional, COleVariant("Data"));
```

```
        // Додати до заголовку таблиці жирний шрифт і вирівнювання
по центру
        range = sheet.GetRange(COleVariant("A1"), COleVariant("D1"));
        font = range.GetFont();
        font.SetBold(covTrue);
        range.SetVerticalAlignment(COleVariant((short)-4108)); //xlVAlignCenter =
-4108
        app.SetVisible(TRUE); // показати MSExcel

        // зберегти документ
        book.SaveAs(COleVariant("c:\\temp\\auto\\xls.xls"),
            covOptional,covOptional,covOptional,covOptional,covOptional,
            0,covOptional,covOptional,covOptional,covOptional,covOptional);

        // закрити MS Excel
        app.Quit();

        // звільнити dispatch
        app.ReleaseDispatch();
    }
}
```

Приклад програмного коду для взаємодії з додатком MS PowerPoint:

```
void CAutoDlg::OnMSPowerPoint()
{
    using namespace PowerPoint;
    using namespace Graph;
    _Application app;
    COleExceptione;

    // запустити MS PowerPoint
    if(!app.CreateDispatch("Powerpoint.Application", &e))
    {
        CString str;
        str.Format("CreateDispatch() failed w/err 0x%08lx", e.m_sc),

        // у випадку помилки, вивести повідомлення
        AfxMessageBox(str, MB_SETFOREGROUND);
        return;
    }

    // показати MS PowerPoint
    app.SetVisible(TRUE);
```

```
// отримати набір презентацій та створити нову
Presentations presSet(app.GetPresentations());
_Presentation pres(presSet.Add(TRUE));

// отримати колекцію слайдів
Slides slideSet(pres.GetSlides());

// додати новий слайд з графіком
_Slide slide2(slideSet.Add(1, 2));

// додати заголовок слайду
{
    Shapes shapes(slide2.GetShapes());
    Shape shape(shapes.Item(COleVariant((long)1)));
    TextFrame textFrame(shape.GetTextFrame());
    TextRange textRange(textFrame.GetTextRange());
    textRange.SetText("Diagram title");
}

// Додати нову гістограму на місці старого графіка
{
    // Отримати координати старого графіка
    float cTop, cWidth, cHeight, cLeft;
    Shapes shapes(slide2.GetShapes());
    Shape shape(shapes.Item(COleVariant((long)2)));
    cTop = shape.GetTop();
    cWidth = shape.GetWidth();
    cHeight = shape.GetHeight();
    cLeft = shape.GetLeft();

    // знищити старий графік
    shape.Delete();

    // додати гістограму
    shape = shapes.AddOLEObject(cLeft, cTop, cWidth, cHeight,
                                "MSGraph.Chart", "", 0, "", 0, "", 0);

    // отримати об'єкт графіка зі слайда презентації
    OLEFormat olefmt = shape.GetOLEFormat();
    Chart chart = olefmt.GetObject();
    Application appl = chart.GetApplication();

    // отримати сторінку з даними для гістограми
    DataSheet sheet = appl.GetDataSheet();
```

```
        Range cells = sheet.GetCells();
        cells.Clear(); // очистити дані

        // додати нові дані (значення для гістограми)
        cells.SetItem(COleVariant((short)2), COleVariant((short)1),
COleVariant("Items"));
        cells.SetItem(COleVariant((short)1), COleVariant((short)2),
COleVariant("a"));
        cells.SetItem(COleVariant((short)2), COleVariant((short)2),
COleVariant((short)2));
        cells.SetItem(COleVariant((short)1), COleVariant((short)3),
COleVariant("b"));
        cells.SetItem(COleVariant((short)2), COleVariant((short)3),
COleVariant((short)9));
        cells.SetItem(COleVariant((short)1), COleVariant((short)4),
COleVariant("c"));
        cells.SetItem(COleVariant((short)2), COleVariant((short)4),
COleVariant((short)12));
        cells.SetItem(COleVariant((short)1), COleVariant((short)5),
COleVariant("d"));
        cells.SetItem(COleVariant((short)2), COleVariant((short)5),
COleVariant((short)8));
        // оновити дані
        appl.Update();
    }

    CString filename("C:\\Temp\\MyPpt.ppt");
    pres.SaveAs(filename, 1, 1); // зберегти перзентацію
    app.Quit(); // вийти з програми
    app.ReleaseDispatch(); // звільнити dispatch
}
```

Вимоги

1. Для документа MS Word створити не менше 5 параграфів. До кожного параграфа додати не менше 5 строк. Застосувати до тексту параграфів різні шрифти, стилі, типи вирівнювання та форматування.
2. Для документа MS Excel створити таблицю, що має заголовок та поле таблиці розміром не менше 5 х 5 клітинок. Додати назви стовпців та номери рядків, заповнити комірки таблиці. Для назв стовпців, заголовка та комірок таблиці використовувати різне форматування.

3. Для документа MS PowerPoint створити слайд із гістограмою, яка повинна мати не менше 8 стовпців.
4. Усі документи повинні бути створені на основі реальних даних у БД.

У звіті представити

1. Тему роботи.
2. Призначення програмного продукту.
3. Завдання.
4. Для кожного типу вихідних даних:
   - опис призначення;
   - діалог (форму) та елемент управління, який відповідає за генерацію даного типу вихідних даних;
   - графічний вигляд відповідного документа MS Office;
   - програмний код для генерації відповідного документа;
   - наявність відповідних даних у таблиці БД.
5. Висновки.

СПИСОК ЛІТЕРАТУРИ

1. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition, Part 2, Chapter 6. (знаходиться в директорії …\Computer Science\OOM\Books).
2. Кулямин В.В. Технологии программирования. Компонентный поход. Лекция 4. Анализ предметной области и требования к ПО.
3. Леоненков. Самоучитель UML. Глава 11.
4. Гради Буч. Объектно-ориентированный анализ и проектирование.
5. How to find and use Office object model documentation http://support.microsoft.com/default.aspx?scid=kb;en-us;222101.
6. How to create an automation project using MFC and a type library http://support.microsoft.com/kb/178749/.
7. How To Automate Microsoft Word to Perform a Mail Merge Using Visual C++ and MFC http://support.microsoft.com/kb/220911/.
8. How to use MFC to automate Excel and create and format a new workbook http://support.microsoft.com/kb/179706/.
9. How to use MFC to automate Excel 2000 and Excel 2002 and obtain an array from a range in Excel 2000 and Excel 2002 http://support.microsoft.com/kb/186122/.
10. How to automate PowerPoint by using Visual C++ 5.0 or Visual C++ 6.0 with The Microsoft Foundation Classes http://support.microsoft.com/kb/222960/.

Додаток А
ПЕРЕЛІК ВАРІАНТІВ ЗАВДАННЯ

Варіанти практичних робіт:

1. GPS навігатор.
2. Електронний словник (перекладач).
3. Банкомат.
4. Система обліку товарів на складі.
5. Автоматизована бібліотечна система.
6. Інтернет-блог.
7. Інтернет-магазин.
8. Інтернет-форум.
9. Система підтримки складання розкладу занять.
10. Система продажу товарів у супермаркеті.
11. Instant Messenger.
12. Редактор зображень.
13. Система документообігу на підприємстві.
14. Робоче місце бухгалтера з нарахування заробітної платні.
15. Електронна система продажу нерухомості.
16. Система оплати комунальних послуг.
17. Електронна система продажу ліків в аптеці.
18. Система керування контентом мобільного оператора.
19. Електронна система для туристичного оператора.
20. Електронна система продажу квитків до кінотеатру.
21. Електронна система продажу залізничних квитків.
22. Торгівельний автомат.
23. Музичний автомат.
24. Система електронних переказів готівки.
25. Міні-АТС.
26. Електронна система обліку робочого часу працівника на підприємстві.

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

ПРАКТИКУМ
з об'єктно-орієнтованих методологій створення комп'ютерних систем

Методичні вказівки
до виконання практичних робіт
для студентів напряму «Комп'ютерні науки»

Випуск № 134

Редактор *Н. Савич.*
Технічний редактор *К. Буян*.
Комп'ютерна верстка *К. Дорофеєва.*
Друк *С. Волинець*. Фальцювально-палітурні роботи *А. Грубкіна*.

Підп. до друку 17.12.2009 р.
Формат 60x84 1/16. Папір офсет.
Гарнітура «Times New Roman». Друк ризограф.
Умовн. друк. арк. 3,95. Обл.-вид. арк. 2,03.
Тираж 100 пр. Зам. № 2959.

Видавець і виготовлювач: ЧДУ ім. Петра Могили.
54003, м. Миколаїв, вул. 68 Десантників, 10.
Тел.: 8 (0512) 50-03-32, 8 (0512) 76-55-81, e-mail: vrector@kma.mk.ua.
Свідоцтво суб'єкта видавничої справи ДК № 3460 від 10.04.2009 р.