МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РОЗРОБКИ ПРОГРАМНИХ СИСТЕМ

Методичні вказівки та завдання
для студентів спеціальності 121 "Інженерія програмного забезпечення"

Київ 2020

Зміст

Загальні положення

1. Лабораторні роботи з дисципліни «Інструментальні програмні засоби розробки ІУС», які проводяться зі студентами спеціальності 015 «Професійна освіта», розраховані на 36 годин і охоплюють головні розділи курсу.
2. Мета виконання лабораторних робіт полягає в дослідженні основних принципів інструментальних програмних засобів розробки інформаційних управляючих систем.
   Задачами виконання лабораторних робіт є:
   - поглиблене вивчення основних теоретичних положень дисципліни «Інструментальні програмні засоби розробки ІУС»;
   - отримання практичних навичок розробки інформаційних систем і програмного забезпечення із застосуванням мови UML.
3. Порядок виконання лабораторних робіт: варіанти завдань студенти отримують за номером у списку.
   При виконанні кожної лабораторної роботи необхідно:
   - ознайомитись з постановкою задачі;
   - розробити вказану в роботі діаграму;
   - зберегти файл моделі та оформити звіт про виконану роботу.
4. Звіт до лабораторних робіт складається з наступних пунктів:
   - титульна сторінка;
   - тема лабораторної роботи;
   - мета лабораторної роботи;
   - стислі теоретичні відомості;
   - опис елементів діаграми, що застосовуються для побудови діаграми;
   - побудована діаграма;
   - висновки по роботі.

Теоретичні основи розробки інформаційних управляючих систем

Розробка інформаційних управляючих систем (ІУС) завжди починається з визначення мети проекту. Основне завдання будь-якого успішного проекту полягає в тому, щоб на момент випуску системи та протягом всього часу її експлуатації можна було забезпечити:

- достатню функціональність системи та ступінь адаптації умов її функціонування, що постійно змінюються;
- необхідний час реакції системи на запит;
- безпомилкову роботу системи та її доступність для обробки запитів користувачів;
- простоту експлуатації та підтримки системи;
- необхідний захист даних в системі.

Продуктивність є головним фактором, що визначає ефективність системи. Ефективне проектне рішення служить основою високопродуктивної системи.

Розробка ІУС охоплює три основні області:

- проектування об’єктів даних, які будуть реалізовані в базі даних;
- проектування програм, екранних форм та звітів, які будуть забезпечувати виконання запитів до даних;
- облік конкретного середовища: топології мережі, конфігурації апаратних засобів, архітектури, паралельної обробки, розподіленої обробки даних.

В реальних умовах проектування - це пошук способу, який задовольняє вимогам функціональності системи засобами наявних технологій з урахуванням заданих обмежень.

До будь-якого проекту висувається ряд вимог, наприклад, максимальний час розробки проекту, максимальні грошові вкладення в проект, тощо. Головна складність розробки ІУС полягає в тому, що вона є не такою структурованої, як аналіз вимог до проекту або реалізація того чи іншого проектного рішення.

Лабораторна робота №1

Введення в CASE-пакет Rational Rose

Мета роботи: вивчити принципи роботи з CASE-пакетом Rational Rose.

Призначення роботи: Основні етапи проведення проектування в Rational Rose. Поняття нотації. Створення нового проекту в Rational Rose. Основні моделі ООП і їх подання до Rational Rose.

Загальні відомості

Rational Rose - потужний CASE-засіб для розробки ІУС будь-якої складності. Однією з переваг цього програмного продукту є можливість використання діаграм на мові UML. Можна сказати, що Rational Rose є графічним редактором UML діаграм.

CASE-засіб (Computer Aided Software Engineering) - це інструмент, який дозволяє автоматизувати процес розробки інформаційної системи та програмного забезпечення.

UML (Unified Modeling Language) - це графічна мова моделювання загального призначення, призначена для специфікації, візуалізації, проектування та документування всіх атрибутів, що створюються при розробці програмних систем. В рамках мови UML всі уявлення про моделі складної системи фіксуються у вигляді спеціальних графічних конструкцій, що одержали назву діаграм.

У розпорядження користувача системи Rational Rose надає наступні типи діаграм, послідовне створення яких дозволяє отримати повне уявлення про всю систему та її окремі компоненти:

- Use case diagram (діаграми прецедентів, варіантів використання);
- Deployment diagram (діаграми топології);
- Statechart diagram (діаграми станів);
- Activity diagram (діаграми активності);
- Interaction diagram (діаграми взаємодії);
- Sequence diagram (діаграми послідовностей дій);
- Collaboration diagram (діаграми співробітництва);
- Class diagram (діаграми класів);
- Component diagram (діаграми компонент).

Інтерфейс Rational Rose

Після запуску програми Rational Rose автоматично створюється новий проект і в робочому вікні діаграми з'являється за замовчуванням вікно діаграми класів.

П'ять основних елементів інтерфейсу Rational Rose (рис. 1) - це браузер, вікно документації, панелі інструментів, вікно діаграми і журнал (log). Їх призначення полягає в наступному:

• браузер (browser) - використовується для швидкої навігації по моделі;

• вікно документації (documentation window) - застосовується для роботи з текстовим описом елементів моделі;

• панелі інструментів (toolbars) - застосовуються для швидкого доступу до найбільш поширеним командам;

• вікно діаграми (diagram window) - використовується для перегляду і редагування однієї або декількох діаграм UML;

• журнал (log) - застосовується для перегляду помилок і звітів про результати виконання різних команд.

Браузер

Браузер - це ієрархічна структура, що дозволяє здійснювати навігацію по моделі. Все, що додається в модель - дійові особи, варіанти використання, класи, компоненти - буде показано у вікні браузера. За допомогою браузера можна:

• додавати в модель елементи (дійові особи, варіанти використання, класи, компоненти, діаграми і т.д.);

• переглядати існуючі елементи моделі;

• переглядати існуючі зв'язки між елементами моделі;

• переміщати елементи моделі;

• перейменовувати ці елементи;

• додавати елементи моделі до діаграми;

• пов'язувати елемент з файлом або адресою Інтернет;

• групувати елементи в пакети;

• працювати з деталізованою специфікацією елемента;

• відкривати діаграму.

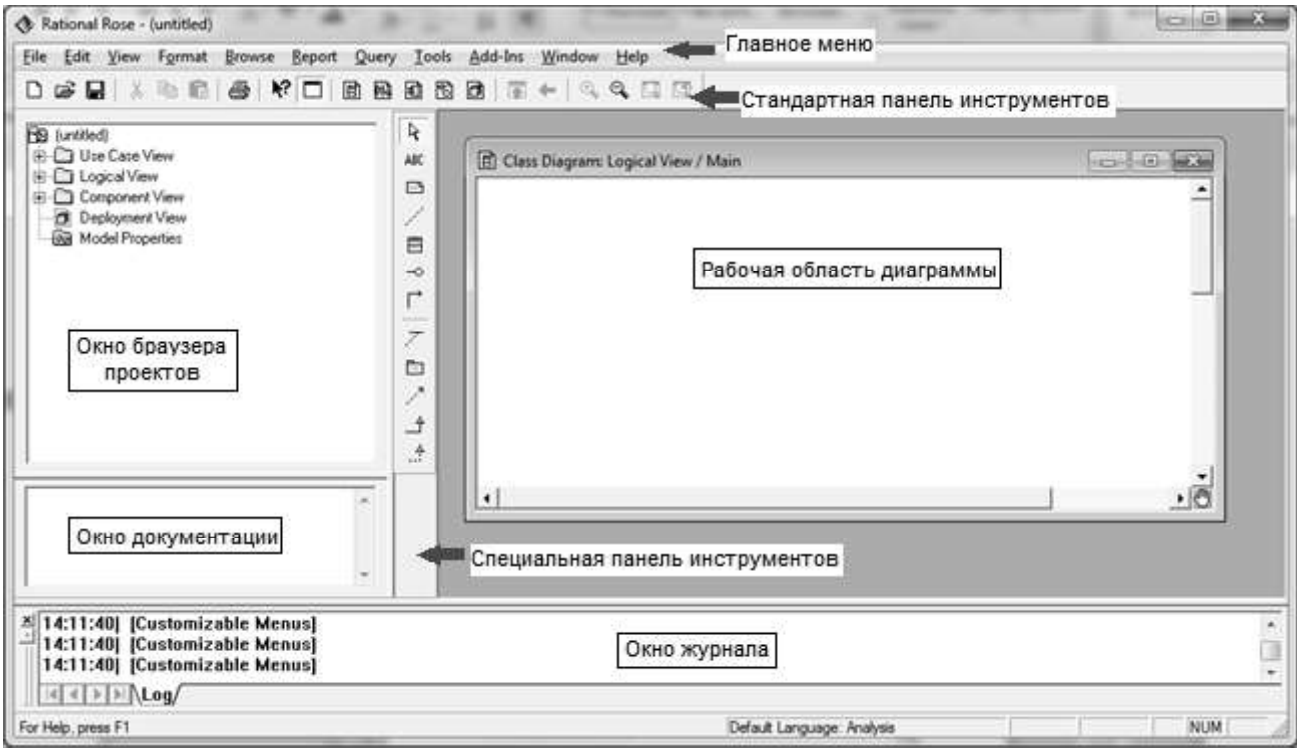

Рис. 1. Загальний вигляд робочого інтерфейсу CASE-засобу Rational Rose

Браузер підтримує чотири представлення (view): представлення варіантів використання, компонентів, розміщення і логічне уявлення.

Браузер організований у вигляді ієрархічної структури (дерева). Кожен елемент моделі може містити інші елементи, що знаходяться нижче його в ієрархії. Знак «-» біля елемента означає, що його гілка повністю розкрита. Знак «+» - що його гілка згорнута.

Вікно документації

З його допомогою можна документувати елементи моделі Rational Rose. Наприклад, можна зробити короткий опис кожної дійової особи. При документуванні класу все, що буде написано в вікні документації, з'явиться потім у вигляді коментаря в створеному коді, що позбавляє від необхідності згодом вносити ці коментарі вручну. Документація буде виводитися також в звітах, що створюються в середовищі Rational Rose.

Панелі інструментів

Панелі інструментів забезпечують швидкий доступ до найбільш поширеним командам. У цьому середовищі існує два типи панелей інструментів: стандартна панель і панель діаграми. Стандартна панель

видно завжди, її кнопки відповідають командам, які можуть використовуватися для роботи з будь-діаграмою. Панель діаграми своя для кожного типу діаграм UML.

Всі панелі інструментів можуть бути змінені і налаштовані користувачем. Щоб показати або приховати стандартну панель інструментів (або панель інструментів діаграми):

1.Виберіть пункт *Tools –> Options*.

2. Виберіть вкладку *Toolbars*.

3. Щоб зробити видимою або невидимою стандартну панель інструментів, позначте (або зніміть позначку) контрольний перемикач *Show Standard ToolBar* (або *Show Diagram ToolBar*)

Щоб збільшити розмір кнопок на панелі інструментів:

1. Клацніть правою кнопкою миші на необхідної панелі.

2. Виберіть у контекстному меню пункт *Use Large Buttons.*

Щоб налаштувати панель інструментів:

1. Клацніть правою кнопкою миші на необхідної панелі.

2. Виберіть пункт *Customize* (налаштувати).

3. Щоб додати або видалити кнопки, виберіть відповідну кнопку і потім клацніть на кнопці *Add* (додати) або *Remove* (видалити), як показано на рис. 2.

Вікно діаграми

У вікні діаграми видно, як виглядає одна або кілька діаграм UML моделі. При внесенні в елементи діаграми змін Rational Rose автоматично оновиться браузер. Аналогічно, при внесенні змін до елемента за допомогою браузера Rational Rose автоматично оновляться відповідні діаграми. Це допомагає підтримувати модель в актуальному стані.

Журнал

Під час роботи над моделлю певна інформація буде розміщуватись у вікні журналу. Наприклад, туди поміщаються повідомлення про помилки, що виникають при генерації коду. Не існує способу закрити журнал зовсім, але його вікно може бути мінімізовано.

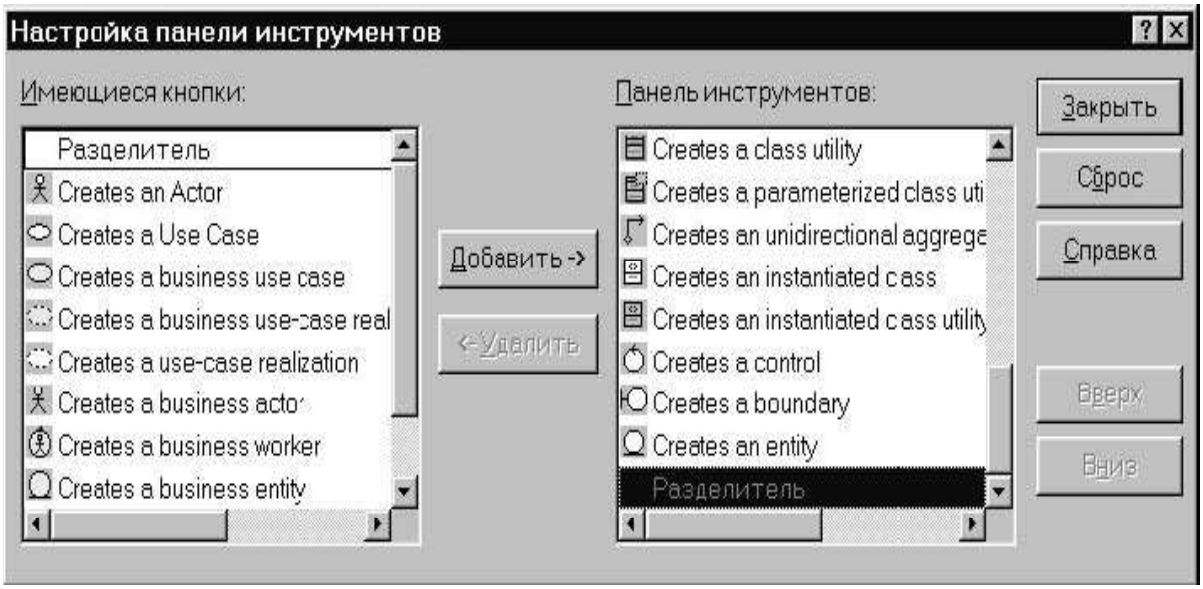

Рис. 2. Вікно для додавання Кнопок в панель інструментів

Контрольні запитання

1. Яке призначення програмного засобу Rational Rose?
2. Що таке CASE-засіб?
3. Що таке UML?
4. Що називають діаграмами UML?
5. Які типи UML діаграм Вам відомі?
6. Назвіть основні елементи вікна програми Rational Rose.

Лабораторна робота №2

Побудова діаграми прецедентів (варіантів використання)

Мета роботи: вивчення діаграм прецедентів; вивчення їх застосування в процесі постановки завдання.

Призначення роботи: ознайомлення з основними елементами діаграми прецедентів: дійові особи, прецеденти і зв'язки між ними.

Загальні відомості

Use Case Diagram - це графічне представлення всіх або частини акторів, прецедентів і їх взаємодії в системі. У кожній системі зазвичай є головна діаграма прецедентів, яка відображає межі системи (акторів) і основну функціональну поведінку системи (прецеденти). Інші діаграми прецедентів можуть створюватися при необхідності, наприклад:

- Діаграма, що показує всі прецеденти для певного актора.
- Діаграма, що показує всі прецеденти, реалізовані на даній ітерації.
- Діаграма, що показує певний прецедент і все його відносини.

Для створення головної діаграми прецедентів:

1. Виберіть пункт меню *Main* в розділі *Use Case View* (в списку браузера) - відкриється *Use Case Diagram*.
2. У списку браузера виберіть актора та перетягніть його на діаграму за допомогою миші.
3. Теж зробіть для інших акторів.
4. Аналогічним чином перетягніть на діаграму прецеденти.

Далі створимо зв'язкі в діаграмі.

Для створення комунікативних асоціацій:

1. На панелі інструментів клацніть по кнопці *Association* або по кнопці *Unidirectional Association* (односпрямований асоціативний зв'язок).
2. Клацніть по актору та перетягніть лінію зв'язку на потрібний прецедент.

Якщо потрібно додати стереотип:

1. Подвійний клік на лінії зв'язку щоб відкрити діалогове вікно *Specification*.

2. У списку *Stereotype* виберіть *communicate*.

3. Натисніть кнопку OK.

Для створення відношення << включає >> (<< доповнює >>):

1. На панелі інструментів клацніть по кнопці *Unidirectional Association*.

2. Клацніть на прецеденті та перетягніть лінію зв'язку на базовий.

3. Двічі клацніть по лінії зв'язку щоб відкрити вікно *Specification*.

4. У списку вибрати значення *include (extend)*.

5. Натисніть кнопку OK.

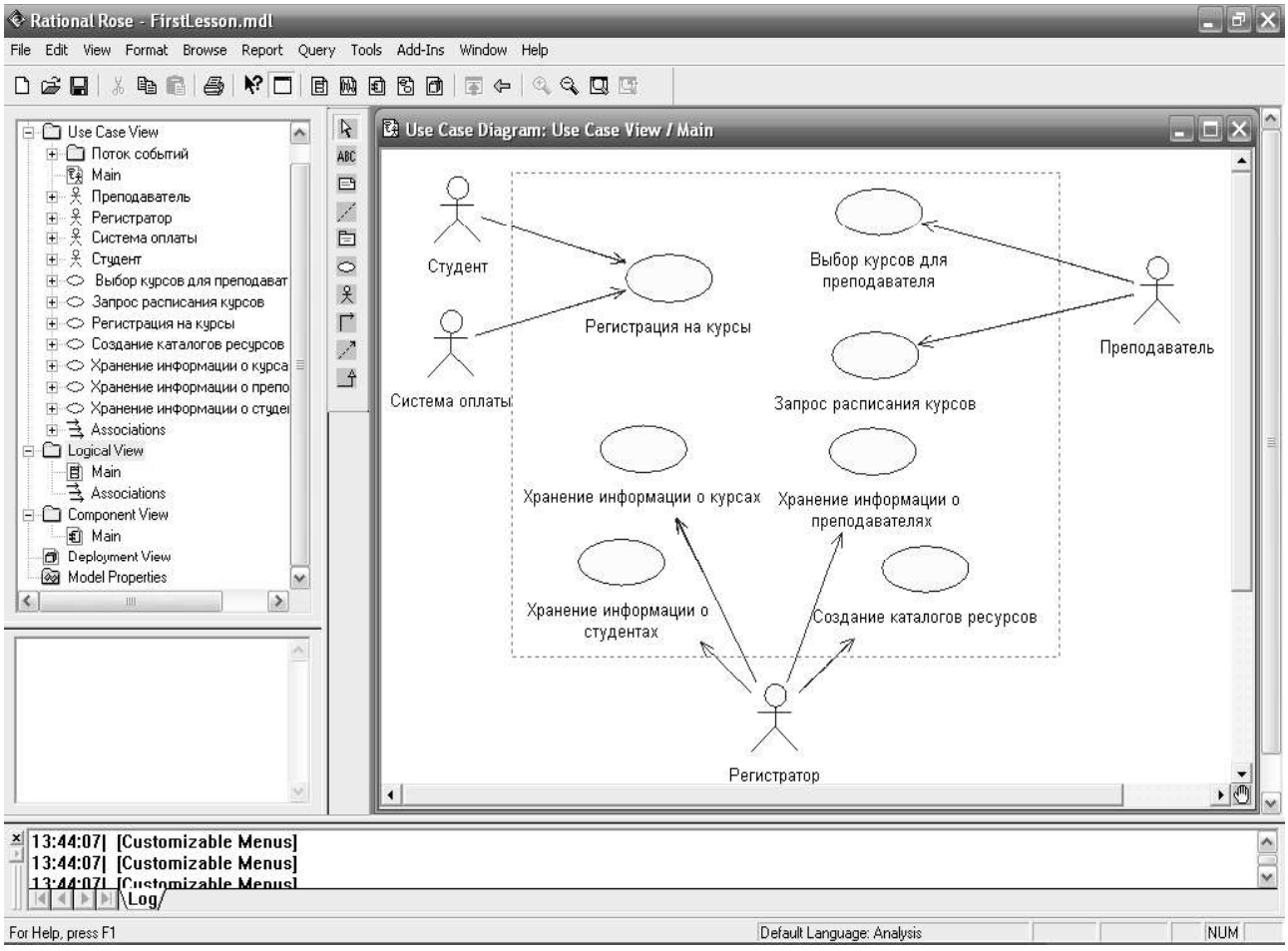

Рис. 3. Приклад діаграми прецедентів

Для створення додаткової діаграми прецедентів:

1. Клацніть правою кнопкою миші по розділу **Use Case View** в списку браузера.

2. У спливаючому меню виберіть *New –> Use Case Diagram*.

3. Введіть назву діаграми

4. Відкрийте діаграму і помістіть на неї необхідних акторів, прецеденти і зв’язки.

Контрольні запитання

1. У чому сенс варіанти використання?
2. Яке призначення діаграм варіантів використання?
3. Назвіть основні властивості варіантів використання.
4. Назвіть основні компоненти діаграм варіантів використання.
5. Що таке «дійова особа»?
6. Яку роль можуть грати дійові особи по відношенню до варіанту використання?
7. Яким чином аналіз зовнішніх подій дозволяє визначити варіанти використання системи?

Лабораторна робота №3

Побудова діаграми класів

Мета роботи: вивчення діаграм класів; вивчення їх застосування в процесі проектування.

Призначення роботи: Діаграма класів показує класи і їхні стосунки, тим самим представляючи логічний аспект проекту. Окрема діаграма класів становить певний ракурс структури класів. На стадії аналізу діаграми класів використовуються, щоб виділити загальні ролі і обов'язки сутностей, що забезпечують необхідну поведінку системи. На стадії проектування діаграми класів використовуються, щоб передати структуру класів, які формують архітектуру системи.

Загальні відомості

Діаграма класів є центральною ланкою методології об'єктно-орієнтованих аналізу і проектування.

Діаграма класів показує класи і їхні стосунки, тим самим представляючи логічний аспект проекту. Окрема діаграма класів становить певний ракурс структури класів. На стадії аналізу діаграми класів використовуються, щоб виділити загальні ролі і обов'язки сутностей, що забезпечують необхідну поведінку системи. На стадії проектування діаграми класів використовуються, щоб передати структуру класів, які формують архітектуру системи.

Кожен клас повинен мати ім'я; якщо ім'я занадто довге, його можна скоротити або збільшити сам значок на діаграмі. Ім'я кожного класу повинна бути унікальною в містить його проекті.

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

- асоціації (наприклад, менеджер може вести кілька проектів),
- підтипи (працівник є різновидом особистості).

На діаграмах класів зображуються також атрибути класів, операції і обмеження, які накладаються на зв'язку між об'єктами. На рис. 4 представлена типова діаграма класів. Далі будуть розглянуті різні фрагменти діаграми.

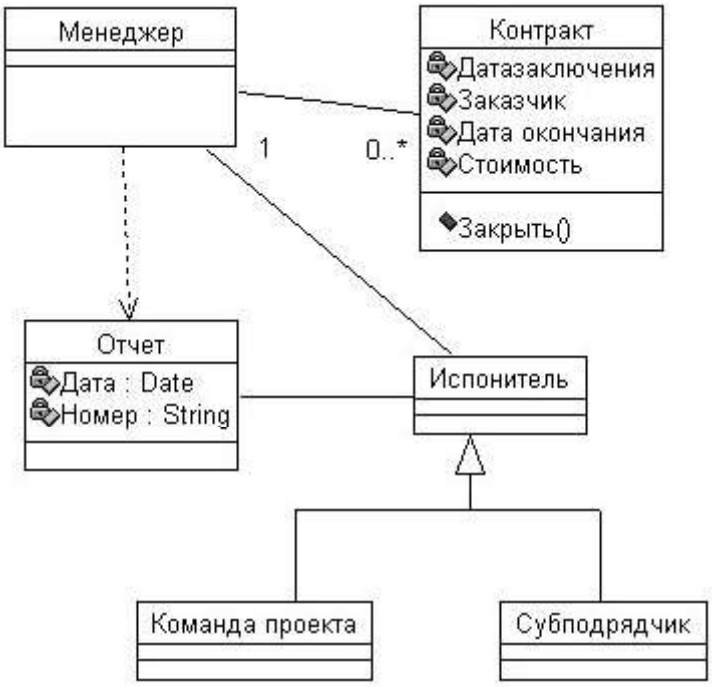

Рис. 4. Типова діаграма класів

Асоціації

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

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

Роль також володіє множинністю, яка показує, скільки об'єктів може брати участь в цих питань. На рис. 11.1 символ «0 .. *» над асоціацією між «Менеджером» і «Контрактам» показує, що з одним «Менеджером» може бути пов'язано багато «Контрактів»; символ «1» показує, що будь-який «Контракт» управляється одним «Менеджером».

В загальному випадку множинність показує нижню і верхню межі кількості об'єктів, які можуть брати участь в зв'язку. Для цього можуть використовуватися однина, діапазон або дискретна комбінація з чисел і діапазонів.

Для асоціації може бути позначений напрямок навігації. Якщо навігація вказана тільки в одному напрямку, то така асоціація називається односпрямованої (асоціація між «Менеджером» і «Звітом» на рис. 4). У двобічній асоціації навігація вказана в обох напрямках. У мові UML відсутність стрілок у асоціації трактується наступним чином: напрямок навігації невідомо чи асоціація двох напрямках.

Атрибути

Атрибути багато в чому подібні до асоціаціям. Різниця між ними полягає в тому, що атрибути припускають єдиний напрямок навігації - від типу до атрибуту.

На рис. 4 атрибути вказані для класів «Контракт» та «Звіт». Залежно від ступеня деталізації діаграми, позначення атрибута може включати ім'я атрибута, тип і значення, що привласнюється за замовчуванням. У синтаксисі UML це виглядає наступним чином: <ознака видимості> <ім'я>: <тип> = <значення за замовчуванням>, де ознака видимості може приймати одне з наступних чотирьох значень:

- загальний (public) - атрибут доступний для всіх клієнтів класу;
- захищений (protected) - атрибут доступний тільки для підкласів і друзів класу;
- секретний (Private) - атрибут доступний тільки для друзів класу.

Операції

Операції являють собою процеси, які реалізуються класом. Найбільш очевидне відповідність існує між операціями і методами над класом.

Повний синтаксис UML для операцій виглядає наступним чином: <ознака видимості> <ім'я> (<список-параметрів>): <тип-вирази-повертає-значення> = <рядок-властивостей>, де ознака видимості може приймати ті ж значення, що і для атрибутів;

- ім'я представляє собою символьний рядок;
- список-параметрів містить необов'язкові аргументи, синтаксис яких збігається з синтаксисом атрибутів;
- тип-вирази-повертає-значення є необов'язковою специфікацією і залежить від конкретної мови програмування;

- рядок-властивостей показує значення властивостей, які застосовуються до даної операції. Прикладом операції на рис. 4 є операція закрити () класу «Контракт».

Узагальнення

Типовий приклад узагальнення включає «Команду проекту» і «Субпідрядника» (рис. 4). Вони володіють деякими відмінностями, однак у них також багато спільного. Однакові характеристики можна помістити в узагальнений клас «Виконавець» (супертіп), при цьому «Команда проекту» і «Субпідрядник» виступатимуть в якості підтипів.

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

Узагальнення з точки зору реалізації пов'язано з поняттям спадкування в мовах програмування. Підклас успадковує всі методи і поля суперкласу і може перевизначати успадковані методи. Підтип можна також реалізувати, використовуючи механізм делегування.

Обмеження

При побудові діаграм класів основним заняттям є відображення різних обмежень. На рис. 4 показано, що «Контракт» може управлятися лише одним «Менеджером».

За допомогою конструкцій асоціації, атрибута і узагальнення можна уточняти найбільш важливі обмеження, але неможливо висловити їх все.

В UML відсутній суворий синтаксис опису обмежень, за винятком розміщення їх в фігурних дужках {}.

Для створення діаграми класів необхідно виконати дії, наведені нижче.

Активізувати робоче вікно діаграми класів можна декількома способами:

1. вікно діаграми класів з'являється за замовчуванням в робочому вікні діаграми після створення нового проекту (рис. 5);

2. клацнути на кнопці із зображенням діаграми класів на стандартній панелі інструментів;

3. розкрити логічне уявлення (*Logical View*) в браузері проекту і двічі клацнути на піктограмі *Main* (Головна);

4. виконати операцію головного меню: **Browse Class Diagram** (Огляд Діаграма класів).

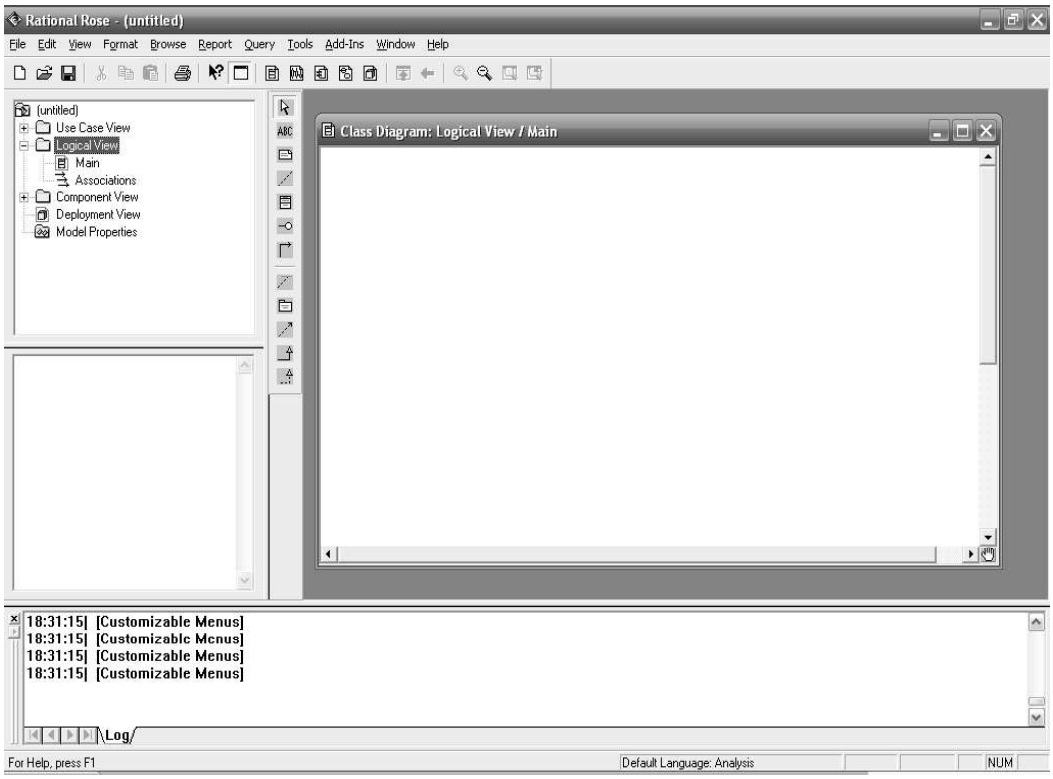

Рис. 5. Робоче вікно діаграми класів

Для додавання класу на діаграму класів потрібно за допомогою лівої кнопки миші натиснути кнопку із зображенням піктограми класу на спеціальній панелі інструментів, відпустити ліву кнопку миші і клацнути лівою кнопкою миші на вільному місці робочого листа діаграми.

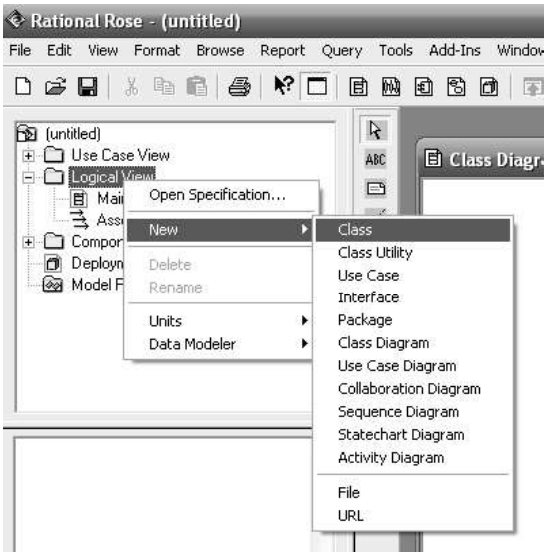

Рис. 6. Створення нового класу

На діаграмі з’явиться зображення класу з маркерами зміни його геометричних розмірів і запропонованим середовищем ім’ям за умовчанням *NewClass*.

Інший спосіб – створити клас через пункт меню New–>Class (рис. 6), а потім перемістити створений клас з вікна браузера проекту в область вікна діаграми класів. В нашому випадку (АІС реєстрації навчальних курсів) потрібно створити 4 класу (рис.7):

- Користувач;
- Викладач;
- Студент;
- Навчальний курс (з об’єктів університету).

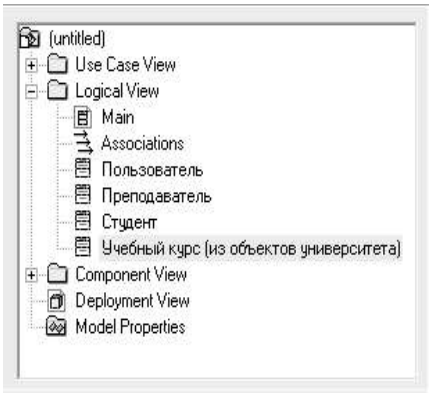

Рис. 7. Браузер проектів, що відображає всі створені класи

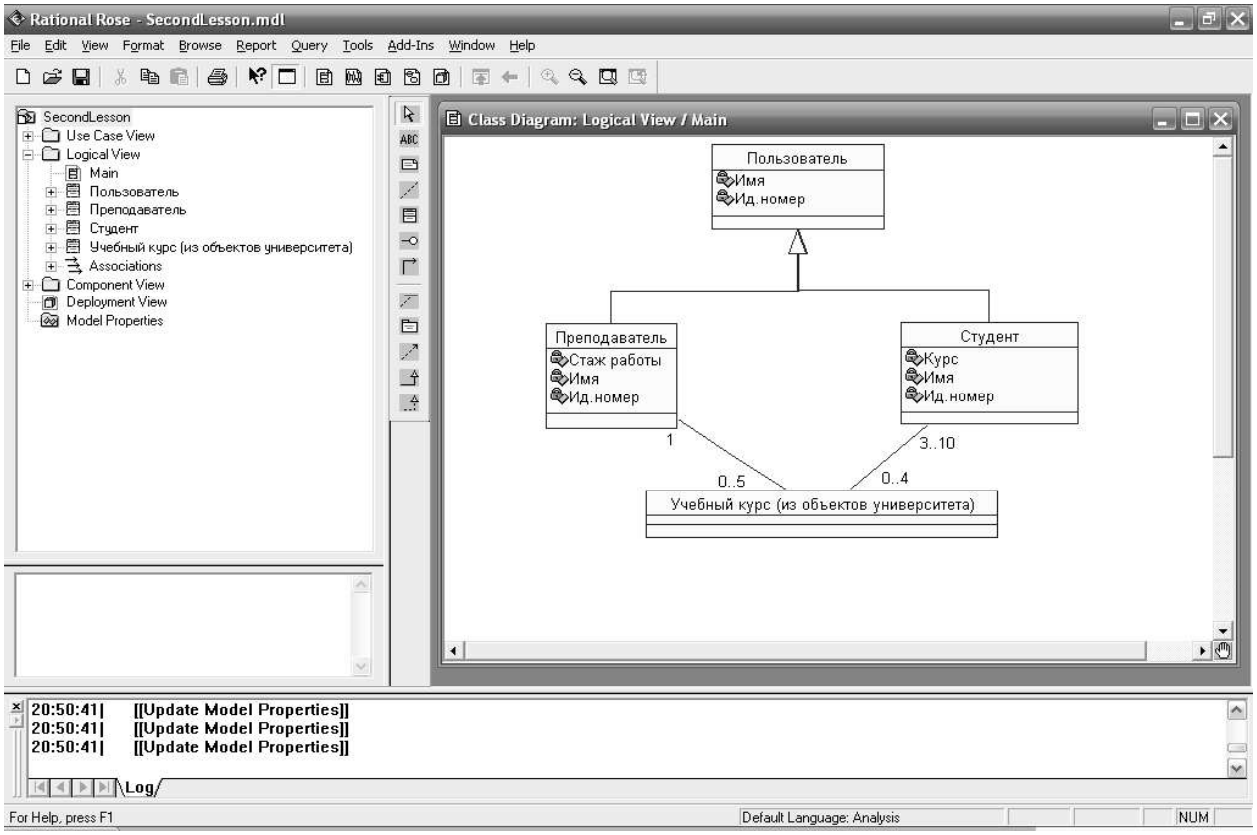

Рис. 8 Вікно діаграми класів - Фрагмент простий концептуальної схеми бази даних реєстрації навчальних курсів

Контрольні запитання

1. Що таке діаграма класів, класифікатор?
2. Що таке характеристика, структурна характеристика?
3. Що таке характеристика поведінки?
4. Що таке клас, активний клас?
5. Коротко розкажіть про ім’я класу.
6. Дайте коротку характеристику поняттю «атрибут класу».
7. Дайте коротку характеристику поняттю «операція класу».

Лабораторна робота №4
Побудова діаграм взаємодії

Мета роботи: вивчення діаграм взаємодії; вивчення їх застосування в процесі проектування.

Призначення роботи: діаграми взаємодії є моделями, що описують поведінку взаємодіючих груп об'єктів

Загальні відомості

Діаграми взаємодії є моделями, що описують поведінку взаємодіючих груп об'єктів. Як правило, діаграма взаємодії охоплює поведінку тільки одного варіанту використання. На такий діаграмі відображається ряд об'єктів і ті повідомлення, якими вони обмінюються між собою в рамках даного варіанту використання.

Існує два види діаграм взаємодії: діаграми послідовності (*Sequence diagrams*) і кооперативні діаграми (*Collaboration diagrams*).

Діаграми послідовностей (Sequence diagrams)

Функціональність елемента Use Case відображається графічно в діаграмі послідовностей. Вона відображає один з можливих шляхів в потоках подій елемента Use Case - наприклад, додавання студента до курсу.

Діаграми послідовності містять об'єкти і сполучення між об'єктивним тими, які показують реалізацію поведінки. Сценарій передбачає, що студент повинен заповнити інформацією (*fill in info*) реєстраційнийну форму (*registration form*), а потім форма пред'являється на розгляд (*submitted*).

Очевидно, що необхідний об'єкт registration form, який приймає інформацію від студента. Створіть діаграму послідовності для елемента *Use Case Register for Courses* (рис. 9).

Клацніть правою кнопкою по елементу Use Case Register for Courses в вікні браузера. Виберіть команду *New–>Sequence Diagram* в контекстному меню. Переконайтеся в додаванні в дерево вікна браузера діаграми послідовності з ім'ям New Diagram.

- Введіть ім'я діаграми *Add a Course*, поки значок нової діаграми залишається виділеним (рис. 9);

- Відкрийте діаграму подвійним клацанням по її значку у вікні браузера;
- Перемістіть актора Student в вікно діаграми (він ініціює сценарій);
- Дайте примірнику актора конкретне ім’я: Joe;

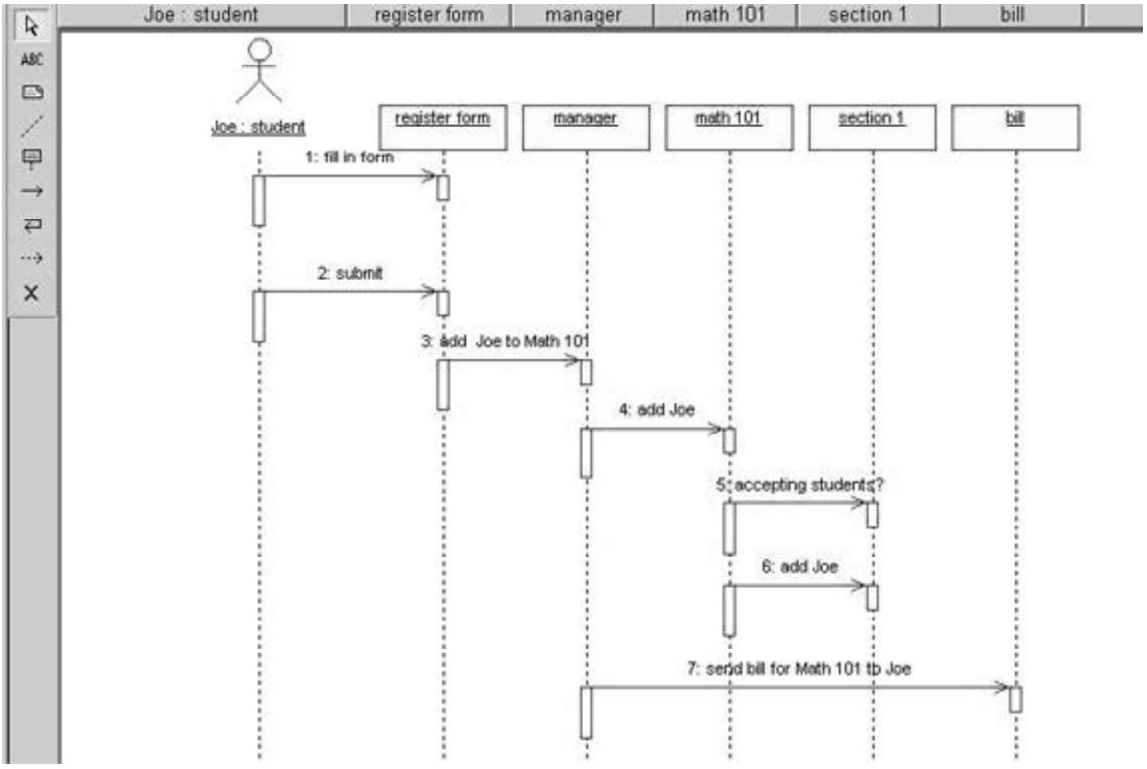

Рис. 9. Діаграма послідовностей Add a Course

- Створіть об’єкт реєстраційна форма (registration form): клацніть спочатку по значку об’єкта (прямокутника) на панелі інструментів і потім в потрібному місці діаграми; введіть ім’я registration form, поки об’єкт залишається виділеним;
- Створіть об’єктні повідомлення "fill in info" (форма повинна заповнюватися студентом) і "submit": клацніть по значку об’єктного повідомлення (стрілкою) на панелі інструментів; клацніть по пунктирною лінії під актором Student і перемістіть стрілку на пунктирну лінію під об’єктом registration form; введіть ім’я повідомлення - fill in information, поки стрілка залишається виділеної; повторіть попередні кроки для створення повідомлення submit.

Кооперативні діаграми (Collaboration diagrams)

Другим видом діаграм взаємодії є кооперативна діаграма (рис. 10).

На кооперативної діаграмі екземпляри об’єктів показані у вигляді піктограм. Лінії між ними означають повідомлення, обмін якими здійснюється в рамках даного варіанту використання.

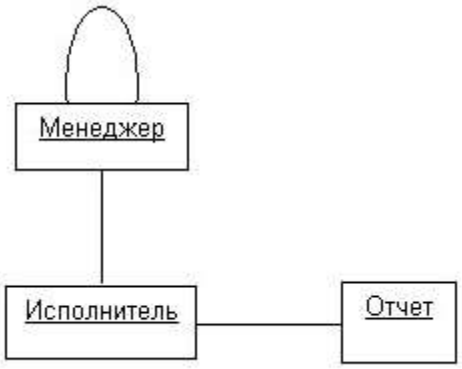

Рис. 10. Кооперативна діаграма

Кожен вид діаграм взаємодії має свої переваги, вибір зазвичай здійснюється виходячи з переваг розробника. На діаграмах послідовності робиться акцент саме на послідовності повідомлень, при цьому легше спостерігати порядок, в якому відбуваються різні події. У разі кооперативних діаграм можна використовувати просторове розташування об’єктів для того, щоб показати їх статичну взаємодію.

Контрольні запитання

1. Яке призначення діаграм взаємодії?
2. Як відносяться між собою діаграми варіантів використання і діаграми взаємодії?
3. Назвіть два види діаграм взаємодії.
4. Що таке «життєва лінія» на діаграмі послідовності?
5. Як на діаграмі послідовності представляються повідомлення?
6. Що показує активізація об’єкта?
7. У чому відмінність кооперативних діаграм від діаграм взаємодії?
8. Які переваги та недоліки кожного виду взаємодії?

Лабораторна робота №5

Побудова діаграми станів

Мета роботи: вивчення діаграм станів ті їх застосування в процесі проектування ІУС.

Призначення роботи: діаграми станів є відомим засобом опису поведінки систем. Вони визначають всі можливі стани, в яких може перебувати конкретний об'єкт, а також процес зміни станів об'єкта в результаті впливу деяких подій.

Загальні відомості

Діаграма станів (Statechart) призначена для відображення станів об'єктів системи, що мають складну модель поведінки (рис. 11).

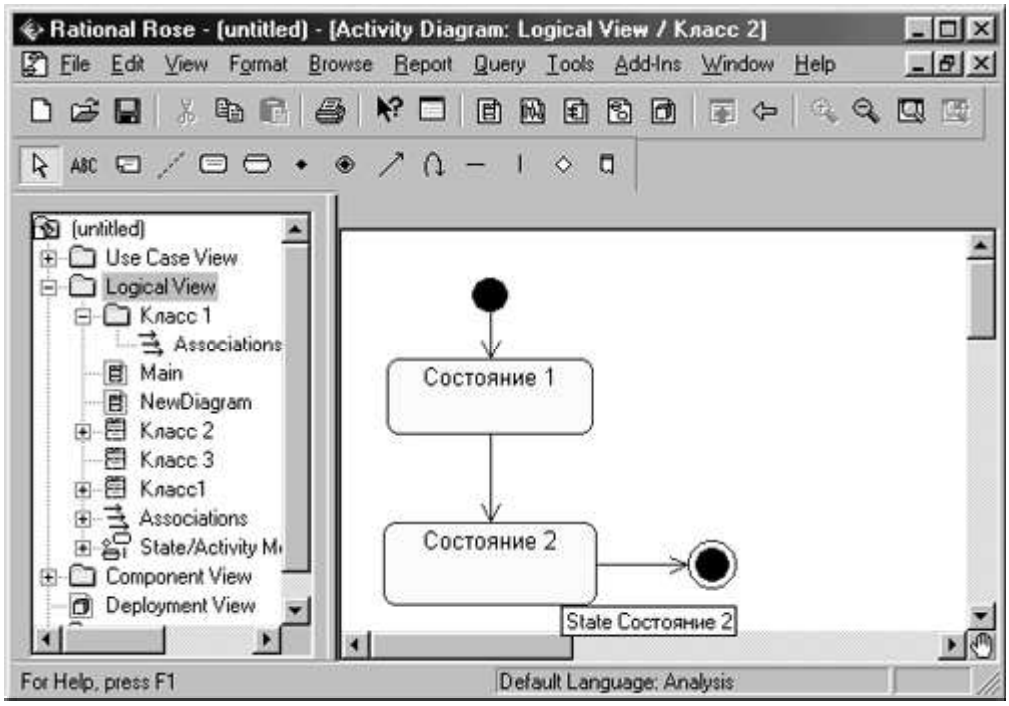

Рис. 11 Приклад графічного зображення діаграми станів

Для того щоб побудувати діаграму станів для класу, його спочатку необхідно створити і уточняти. Після цього виділити на діаграмі класів або в браузері. Почати побудова діаграми станів для обраного класу можна одним з таких способів:

- Розкрити логічне уявлення в браузері (*Logical View*), виділити розглянутий клас і вибрати пункт контекстного меню *Open State Diagram*

(Відкрити діаграму станів), що розкривається після клацання правою кнопкою миші.

- Через пункт меню *Browse–>State Diagram* (Браузер–>Діаграма станів).

Після виконання зазначених дій у вікні діаграми з'явиться чисте зображення для розміщення елементів цієї діаграми, які обирають за допомогою спеціальної панелі інструментів.

Після додавання стану або переходу на діаграму станів можна відкрити специфікацію обраних елементів і визначити їх специфічні риси, доступні на відповідних вкладках. При необхідності можна візуалізувати вкладеність станів і підключити історію окремих станів.

На рис. 12 наведена діаграма станів UML, що відображає поведінку звіту в системі управління проектами. На діаграмі зображені різні стани, в яких може перебувати звіт.

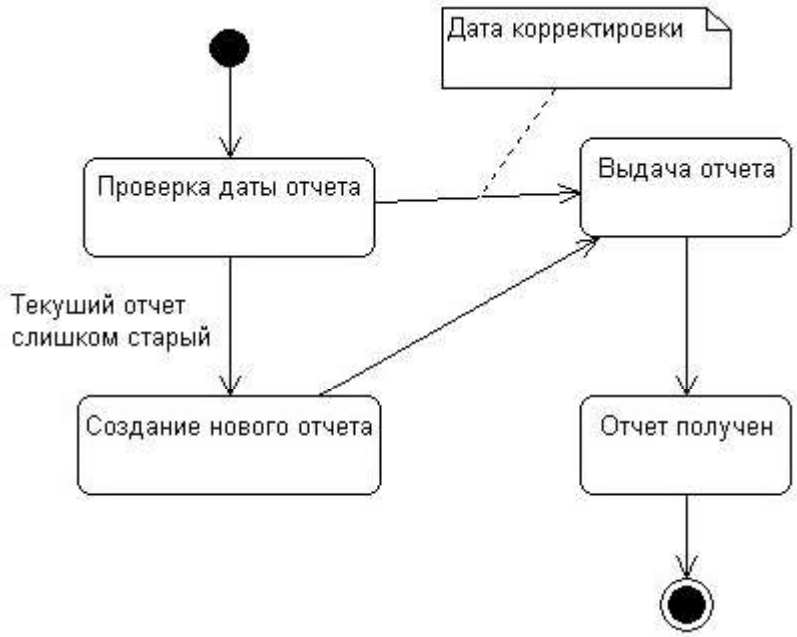

Рис. 12. Діаграма станів поведінки звіту в системі управління проектами

Процес починається з початкової точки, потім слід найперший перехід в стан «Перевірка дати звіту». У поведінці об'єкта в системі можна виділити дії, які відображаються переходами, і діяльності, які відображаються станами. Хоча і те й інше - це процеси, що реалізуються, як правило, деяким методом класу «Звіт», вони трактуються по-різному.

Дії пов’язані з переходами і розглядаються, як миттєві і неперервні. Діяльності пов’язані з станами і можуть тривати досить довго. Діяльність може бути перервана в результаті настання деякої події.

Перехід може містити мітку. Синтаксично мітка переходу складається з трьох частин, кожна з яких є необов’язковою: <Подія> [<Умова>] / <Дія>. Якщо мітка переходу не містить ніякої події, це означає, що перехід відбувається, як тільки завершується будь-яка діяльність, пов’язана з даними станом. Зі стану «Перевірка дати звіту» можливі два переходи. Мітка одного з них включає умову. Умова - це логічне умова, яке може приймати два значення: «істина» або «брехня». Умовний перехід виконується тільки в тому випадку, якщо умова приймає значення «істина», в іншому випадку виконується перехід, що не позначений умовою.

З конкретного стану в даний момент часу може бути здійснений тільки один перехід; таким чином, умови є взаємно виключають для будь-якої події. Існує два особливих стану: вхід і вихід. Будь-яка дія, пов’язане з подією входу, виконується, коли об’єкт входить в даний стан. Подія виходу виконується в тому випадку, коли об’єкт виходить з цього стану. Діаграми станів добре використовувати для опису поведінки деякого об’єкту в декількох різних варіантах використання. Вони не дуже придатні для опису поведінки ряду взаємодіючих об’єктів.

Рекомендується будувати діаграми станів тільки для тих класів, поведінка яких впливає на загальну поведінку системи, наприклад для класів користувальницького інтерфейсу і керуючих об’єктів.

Контрольні запитання

1. Яке призначення діаграм стану?
2. Як відображаються дії і діяльності на діаграмах стану?
3. Що таке умовний перехід і як він описується на діаграмі?
4. Які особливі стану об’єкта відображаються на діаграмі?
5. Які переваги та недоліки діаграм стану?

Лабораторна робота №6

Побудова діаграми пакетів

Мета роботи: вивчення діаграм пакетів та їх застосування в процесі проектування.

Призначення роботи: один з найважливіших питань методології створення ПО: як розбити велику систему на невеликі підсистеми? Саме з цієї точки зору зміни, пов'язані з переходом від структурного підходу до об'єктно-орієнтованого, є найбільш помітними. Одна з ідей полягає в угрупованні класів в компоненти більш високого рівня. В UML такий механізм угруповання носить назву пакетів (package).

Загальні відомості

Один з найважливіших питань методології створення програмного забезпечення - як розбити велику систему на невеликі підсистеми? Саме з цієї точки зору зміни, пов'язані з переходом від структурного підходу до об'єктно-орієнтованого, є найбільш помітними. Одна з ідей полягає в угрупованні класів в компоненти більш високого рівня. В UML такий механізм угруповання носить назву пакетів (package).

Діаграмою пакетів є діаграма, що містить пакети класів і залежності між ними. В загальному вигляді, пакети і залежності є елементами діаграми класів, тобто діаграма пакетів - це всього лише форма діаграми класів. Однак на практиці причини побудови таких діаграм різні.

Залежність між двома елементами має місце в тому випадку, якщо зміни у визначенні одного елемента можуть спричинити за собою зміну в іншому. Що стосується класів, то причини залежностей можуть бути

самими різними: один клас посилає повідомлення іншому; один клас включає частину даних іншого класу; один клас посилається на інший як на параметр операції. Якщо клас змінює свій інтерфейс, то будь-яке повідомлення, яке він посилає, може стати неправильним.

В ідеальному випадку тільки зміни в інтерфейсі класу повинні впливати на інші класи. Мистецтво проектування великих систем включає в себе мінімізацію залежностей, яка знижує вплив змін і вимагає менше зусиль на їх внесення.

На рис. 13 представлено класи предметної області, що моделюють діяльність організації і згрупованими в два пакети: «Клієнти» і «Замовлення».

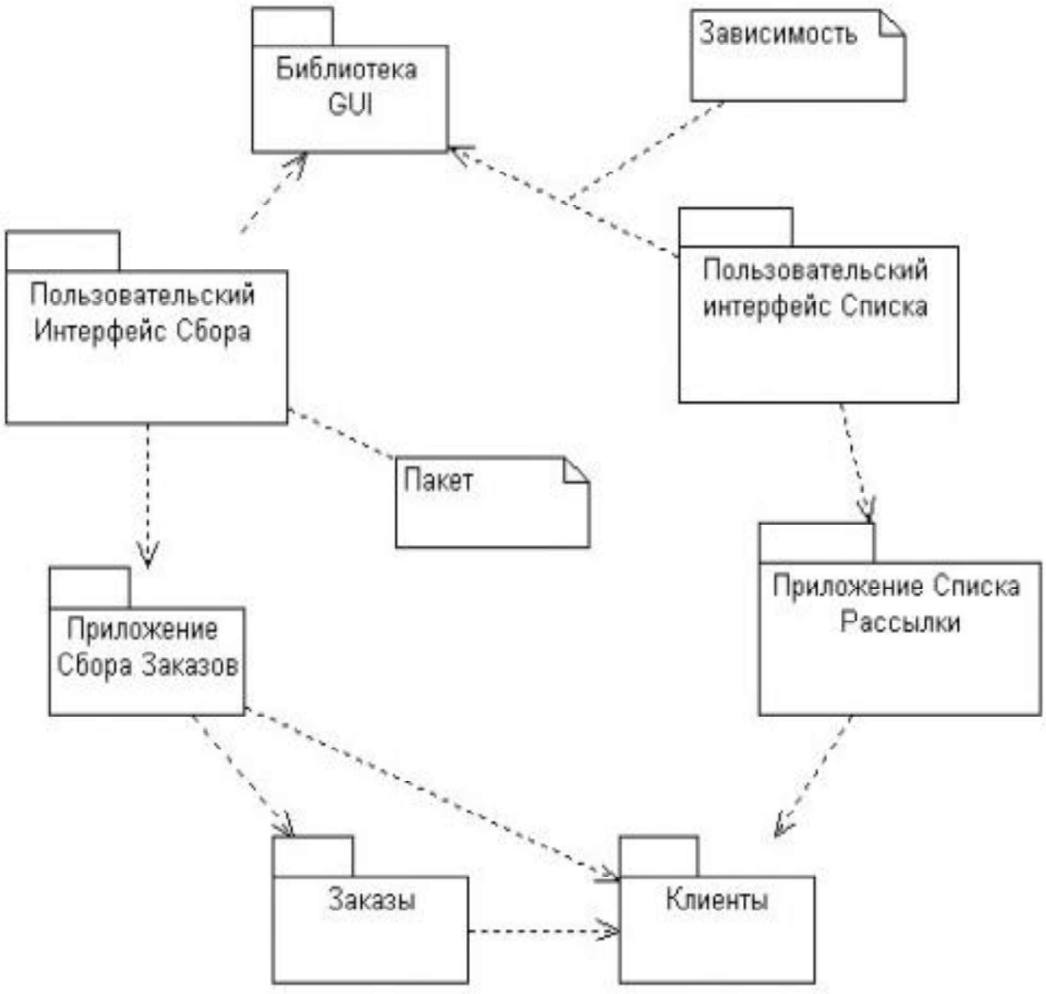

Рис. 13. Класи предметної області, що моделюють діяльність організації

«Додаток збору замовлень» має залежності з обома пакетами предметної області. «Інтерфейс збору замовлень» має залежності з «Додатком збору замовлень» і «Бібліотекою GUI».

Залежність між двома пакетами існує в тому випадку, якщо є будь-яка залежність між будь-якими двома класами в пакетах.

Наприклад, якщо будь-який клас в пакеті «Список розсилки» залежить від будь-якого класу в пакеті «Клієнти», то між відповідними пакетами існує залежність.

Пакети є життєво необхідним засобом для великих проектів. Їх слід використовувати в тих випадках, коли діаграма класів, що охоплює всю систему в цілому і розміщена на єдиному аркуші паперу формату А4, стає мають важко.

Пакети не дають відповіді на питання, яким чином можна зменшити кількість залежностей в системі, що розробляється, проте вони допомагають виділити ці залежності.

Зведення до мінімуму кількості залежностей дозволяє знизити зв’язаність компонентів системи. Але евристичний підхід до цього процесу далекий від ідеалу.

Пакети особливо корисні при тестуванні. Кожен пакет при тестуванні може містити один або кілька тестових класів, за допомогою яких перевіряється поведінка пакета.

Контрольні запитання

1. Яку проблему проектування покликані вирішити діаграми пакетів?
2. У чому відмінність діаграм пакетів від діаграм класів?
3. У чому сенс залежності між елементами діаграми пакетів?
4. Що таке інтерфейс класу?
5. За якими ознаками класи групуються в пакети?

Лабораторна робота №7

Генерація програмного коду в середовищі Rational Rose

Мета роботи: навчитись генерувати програмний код в середовищі Rational Rose.

Призначення роботи: ознайомлення з основними механізмами автоматичної генерації програмного коду.

Загальні відомості

Генерувати програмний код на даному лабораторному занятті будемо на прикладі діаграми класів. Відкрийте створений раніше проект (рис. 14).

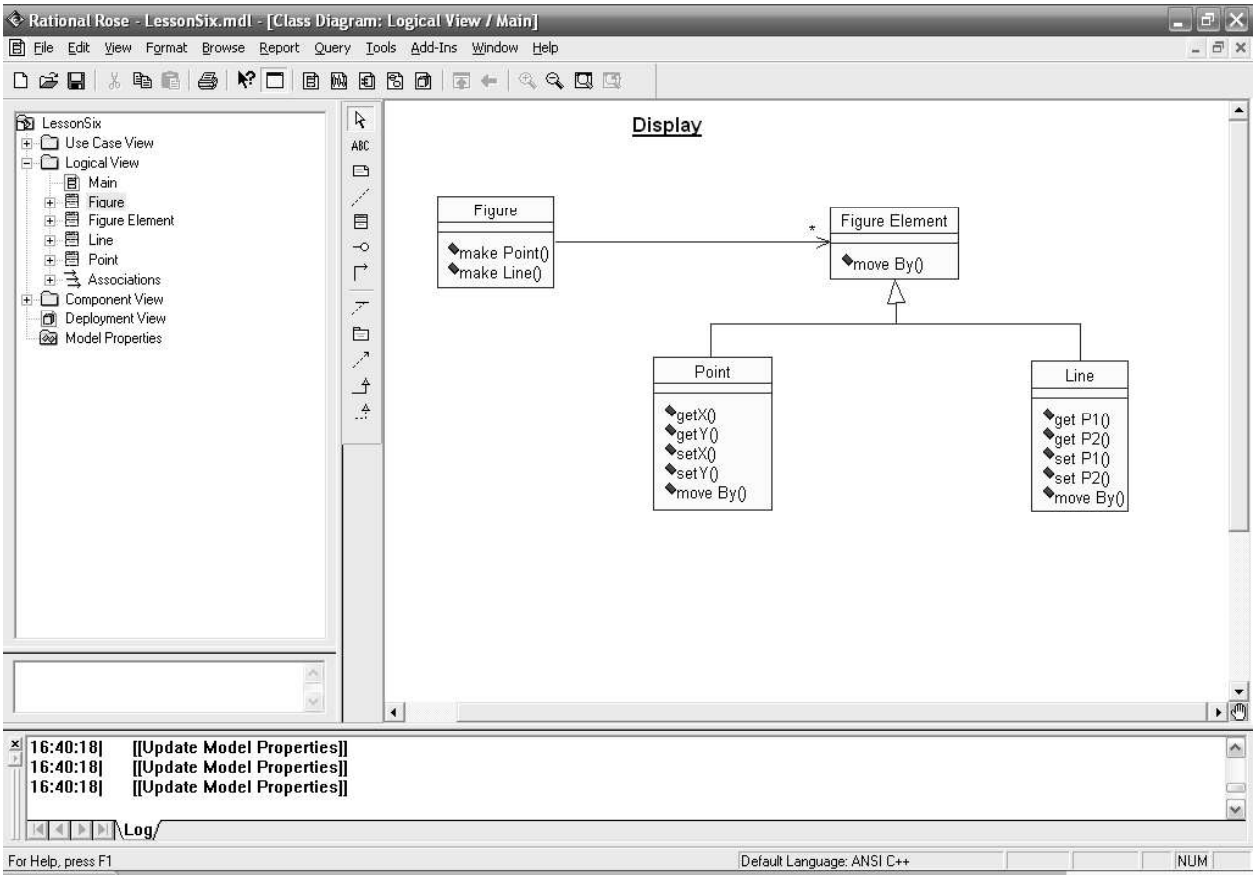

Рис. 14. Діаграма класів - приклад простого графічного редактора

Загальна послідовність дій, які необхідно виконати для генерації програмного коду в середовищі Rational Rose, складається з наступних етапів:

1. Перевірка моделі на відсутність помилок.

2. Створення компонентів для реалізації класів.

3. Відображення класів на компоненти.

4. Вибір мови програмування для генерації тексту програмного коду.

5. Встановлення властивостей генерації програмного коду.

6. Вибір класу, компонента або пакета.

7. Генерація програмного коду.

Особливості виконання кожного з етапів можуть змінюватися в залежності від вибору мови програмування або схеми бази даних.

Розглянемо особливості виконання кожного із зазначених вище етапів для мови реалізації моделі ANSI C++.

Мова ANSI C++ не допускає використання символів кирилиці в якості імен класів, атрибутів і операцій.

Перевірка моделі на відсутність помилок

Для перевірки моделі слід виконати операцію головного меню: *Tools Check Model* (Інструменти Перевірити модель). Результати перевірки розробленої моделі на наявність помилок відображаються у вікні журналу. Перш ніж приступити до генерації тексту програмного коду розробнику слід домогтися усунення всіх помилок і попереджень, про що має свідчити чисте вікно журналу (рис. 15).

Створення компонентів для реалізації класів

Оскільки генерація програмного коду відбувається на базі «Діаграми компонентів», необхідно її створити.

Діаграма компонентів служить частиною фізичного представлення моделі і є необхідною для генерації програмного коду. Для розробки діаграм компонентів в браузері проекту призначене окреме подання компонентів (*Component View*), в якому вже міститься діаграма компонентів з порожнім змістом і ім’ям за умовчанням *Main* (Головна).

Активізація діаграми компонентів може бути виконана одним із таких способів:

- Клацнути на кнопці із зображенням діаграми компонентів на стандартній панелі інструментів.

- Розкрити представлення компонентів в браузері (*Component View*) та двічі клацнути на піктограмі *Main* (Головна).

- Через пункт меню **Browse Component Diagram** (Браузер Діаграма компонентів).

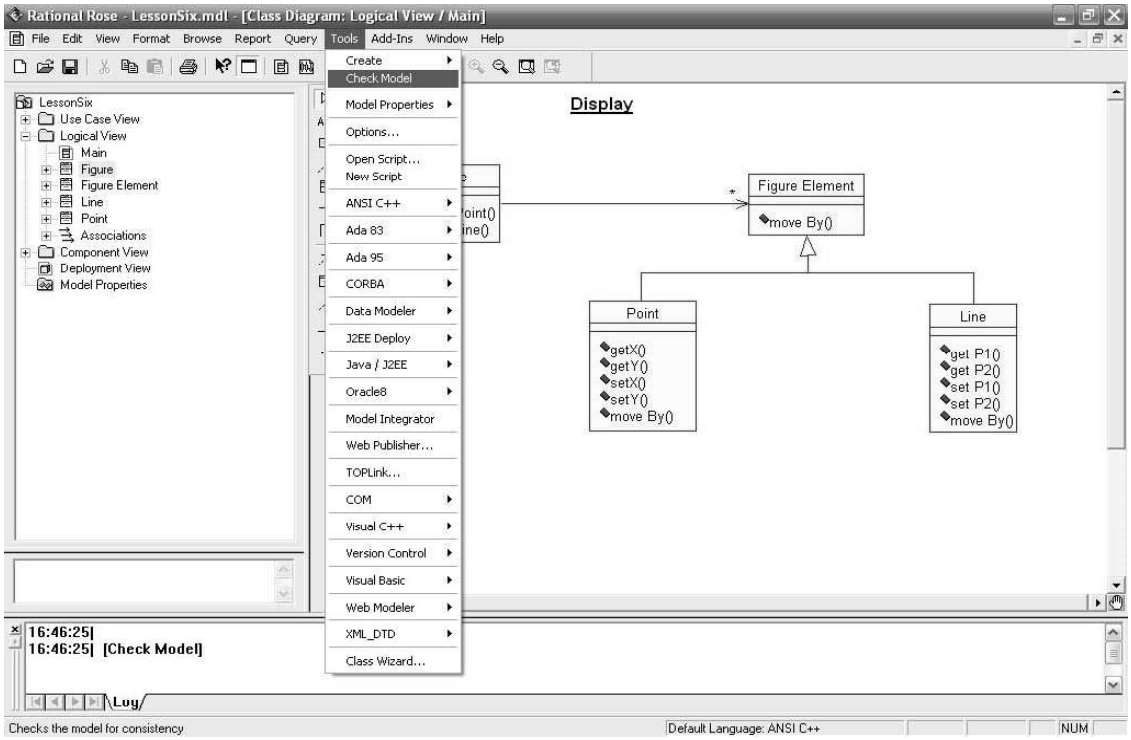

Рис. 15. Перевірка моделі на відсутність помилок

В результаті виконання цих дій з'являється нове вікно з чистим робочим листом діаграми компонентів і спеціальна панель інструментів, що містить кнопки із зображенням графічних примітивів, необхідних для розробки діаграми компонентів. Як «контейнера» і «головного компонента» одночасно, будемо використовувати елемент «*Component*» («Компонент») (рис. 16) Додамо його в область «діаграми компонентів» і поставимо ім'я «MainPaint.exe». «Exe» - приписка в назві, як би говорить про те, що при генерації коду можна отримати не просто файли з кодом програми, але і виконуваний файл.

Відображення класів на компонентах

Для відображення класів на компоненти можна скористатися вікном специфікації властивостей компонента, відкритого на вкладці *Realizes* (Реалізує). Для включення реалізації класу в даний компонент слід виділити необхідний клас на цій вкладці і виконати для нього операцію контекстного меню *Assign* (Призначити). В результаті перед ім'ям класу на цій вкладці з'явиться спеціальна позначка.

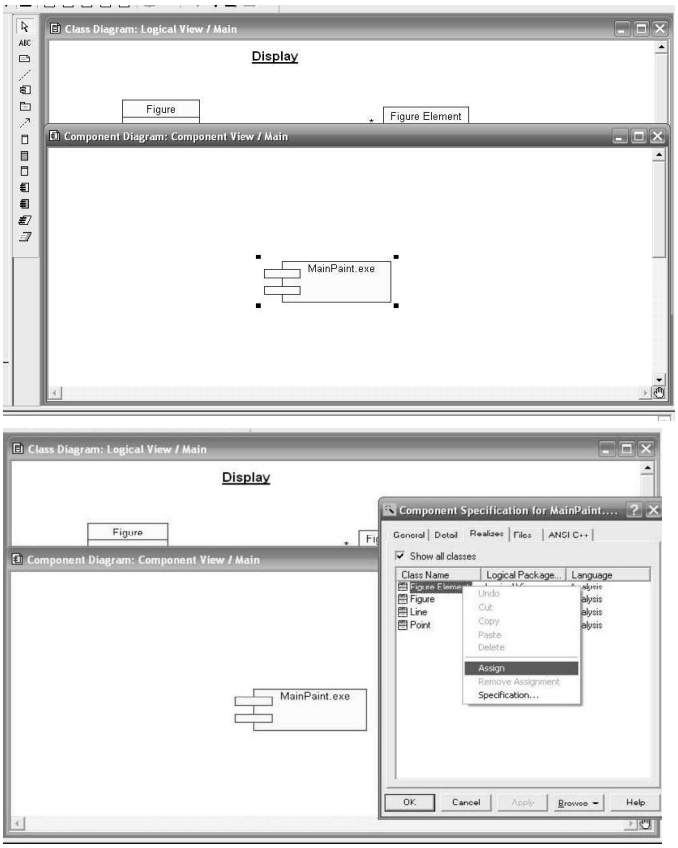

Рис. 16. Створили діаграму компонентів і додали "Компонент" «MainPaint.exe», зв’язали "Компонент" і класи

Вибір мови програмування для генерації тексту програмного коду і Установка властивостей генерації програмного коду

Для вибору мови ANSI C++ в якості мови реалізації моделі слід виконати операцію головного меню: Tools Options (Інструменти Параметри), в результаті чого буде викликано діалогове вікно настройки параметрів моделі. Далі на вкладці *Notation* (Нотація) в рядку *Default Language* (Мова за замовчуванням) з вкладеного списку слід вибрати мову - ANSI C++ (рис. 17).

Якщо з якоїсь причини мови ANSI C ++ не виявилося у вкладеному списку, то слід переконатися в тому, що ця мова програмування встановлений в якості розширення IBM Rational Rose. Для цього слід відкрити вікно встановлених розширень, виконавши операцію головного меню: *Add-Ins Add-In Manager* (Розширення Менеджер розширень), і переконатися в тому, що виставлена відмітка в рядку з ім’ям мови ANSI C++.

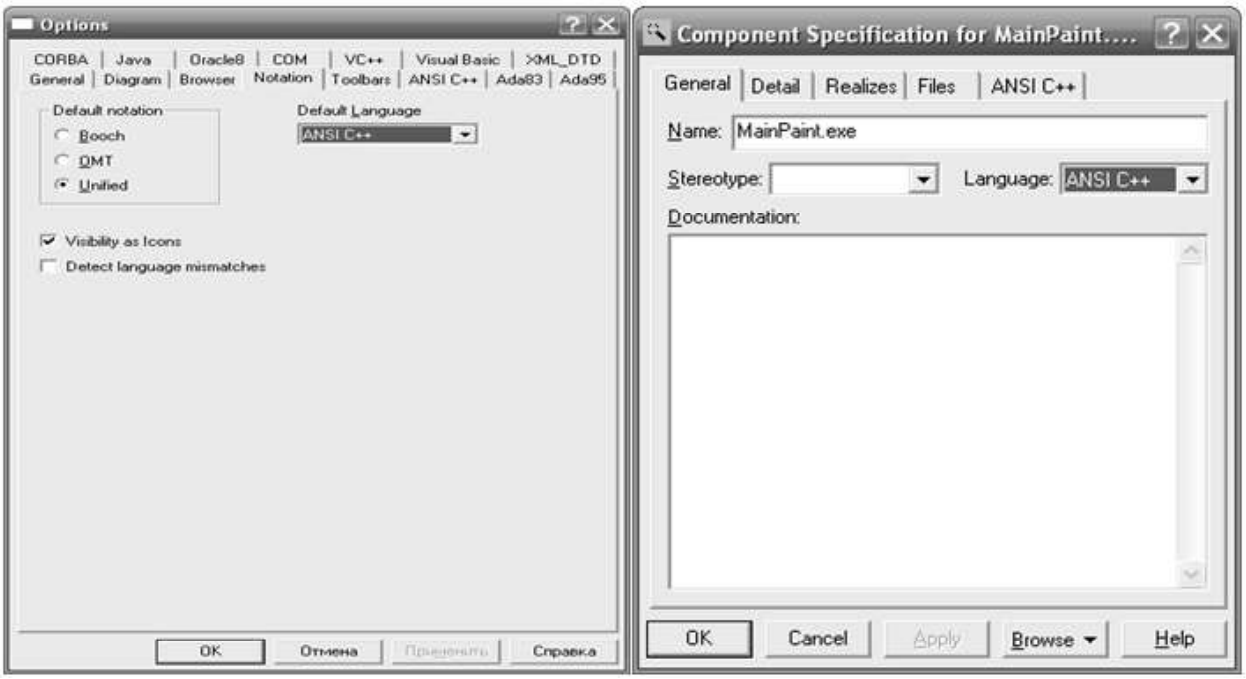

Рис. 17. Вибір мови програмування і мови реалізації компонентів

Вибір класу, компонента або пакета і генерація програмного коду

Генерація програмного коду в середовищі Rational Rose можлива для окремого класу або компонента. Для цього потрібний елемент моделі попередньо слід виділити в браузері проекту і виконати операцію контекстного меню: ANSI C++ *Generate Code_* (Мова ANSI C ++ Генерувати код). В результаті цього буде відкрито діалогове вікно з пропозицією щодо обрання класів для генерації програмного коду на обраною мовою програмування (рис. 18). Після вибору відповідних класів і натискання кнопки OK програма Rational Rose виконує генерація програмного коду.

Після вибору мови програмування за замовчуванням слід змінити мову реалізації кожного з компонентів моделі. З цією метою слід змінити мову в рядку *Language* (Мова) на вкладці *General* (Загальні) вікна специфікації властивостей компонента, для чого з вкладеного списку слід вибрати мову - ANSI C++.

Для перегляду і редагування створених файлів з текстом програмного коду на мові ANSI C ++ призначений вбудований текстовий редактор, який можна відкрити за допомогою операції контекстного меню: ANSI C++ *Browse Header_* (Мова ANSI C++ Переглянути заголовки) або ANSI C++ *Browse Body_* (Мова ANSI C++ Переглянути файл реалізації) для обраного класу в браузері проекту.

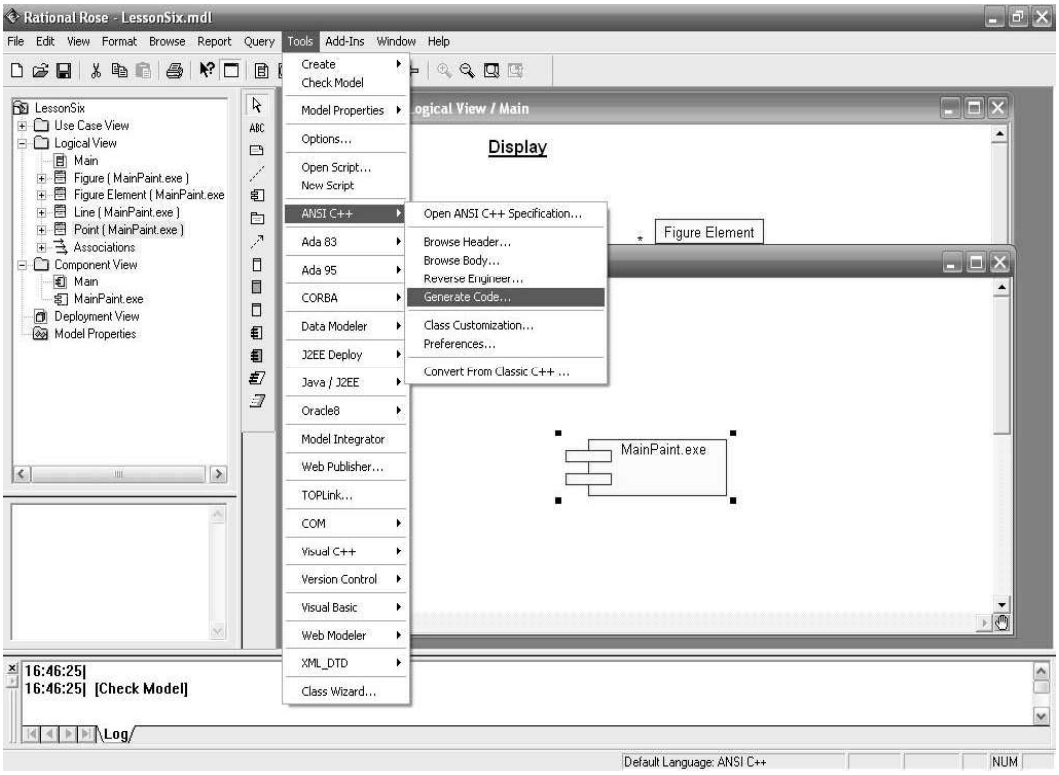

Рис. 18. Генерація програмного коду

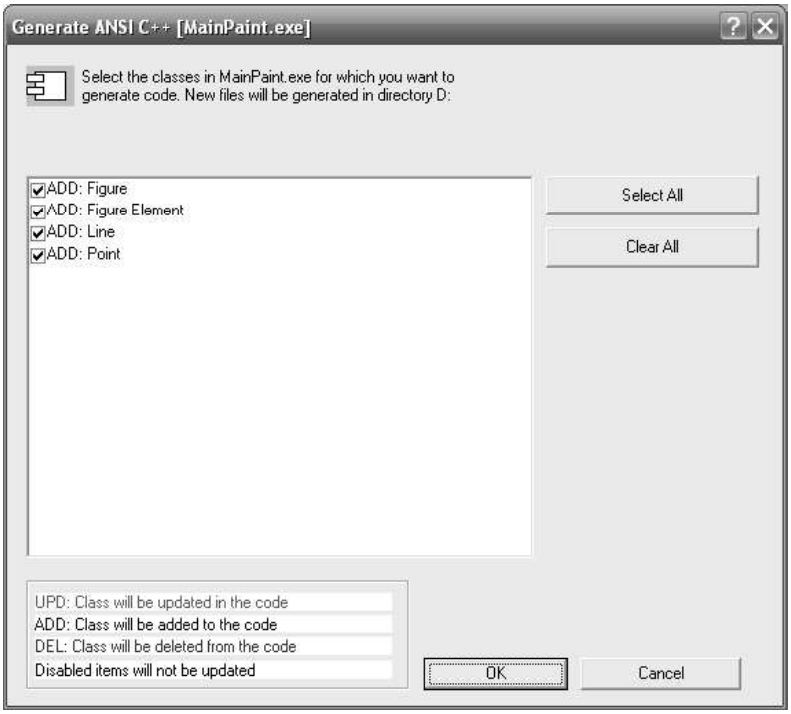

Рис. 19 Вікно вибору класів для генерації програмного коду

Після генерації програмного коду для компонента *MainPaint.exe* кожному класу, реалізованому в даному компоненті, буде відповідати 2 файли з текстом коду на мові ANSI C++ (рис. 20):

- заголовки з розширенням «h»;
- файл реалізації з розширенням «cpp».

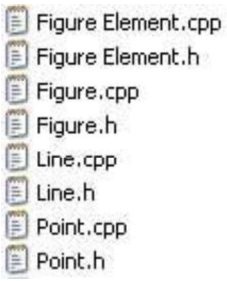

Рис. 20. Згенеровані файли

Приклад згенерованого коду - файл Line.cpp

```
#include "Line.h"
//##ModelId=48FB039303B9
Line::get P1()
{}
//##ModelId=48FB039C0196
Line::get P2()
{}
//##ModelId=48FB03A3035B
Line::set P1(Point P1)
{}
//##ModelId=48FB03CC0203
Line::set P2(Point P2)
{}
//##ModelId=48FB03D40261
Line::move By(Integer X, Integer Y)
{}
```

Приклад згенерованого коду — файл Line.h

```
#ifndef LINE_H_HEADER_INCLUDED_B6EFE6B3
#define LINE_H_HEADER_INCLUDED_B6EFE6B3
#include "Point.h"
//##ModelId=48FAFE8C0399

class Line : public Figure Element {
  public:
    //##ModelId=48FB039303B9
    get P1();
    //##ModelId=48FB039C0196
```

```
    get P2();
    //##ModelId=48FB03A3035B
    set P1(Point P1);
    //##ModelId=48FB03CC0203
    set P2(Point P2);
    //##ModelId=48FB03D40261
    move By(Integer X, Integer Y);
};
#endif /* LINE_H_HEADER_INCLUDED_B6EFE6B3 */
```

Контрольні запитання

1. З яких етапів складається генерація коду?
2. На базі якої діаграми генерується програмний код?
3. Як перевірити відсутність помилок у програмі?
4. Для чого використовується діаграма компонентів?
5. Яка мова програмування використовується при генерації коду?
6. Які переваги автоматичної кодогенерації?
7. Які види діаграм використовуються для генерації коду?
8. Які компоненти вихідного коду генерує Rational Rose?
9. Як у вихідному коді відображаються атрибути і операції класу?
10. Чому важливо детально коментувати компоненти моделі?

Перелік тем лабораторних робіт

1. Інформаційна система торгового підприємства.
2. Інформаційна система аптеки.
3. Інформаційна система відділу кадрів.
4. Програмне забезпечення мобільного телефону.
5. Програмне забезпечення роутера.
6. Програмне забезпечення принтера.
7. Програмне забезпечення МФУ.
8. Програмне забезпечення навігатора.
9. Програмне забезпечення відеореєстратора.
10. Програмне забезпечення мультимедійної клавіатури.
11. Текстовий редактор.
12. Мультимедійний плеєр.
13. Пошукова система.
14. Інформаційна система виробничого підприємства.
15. Інтернет-магазин.
16. Інформаційна система кав’ярні.
17. Інформаційна система університету.
18. Інформаційна система відділу кадрів.
19. Інформаційна система бібліотеки.
20. Інформаційна система складу товарів.

Список літератури

1. *Проектування* інформаційних систем: посібник / за редакцією В.С. Пономаренка. – К.: Видавничий центр «Академія», 2002. – 488 с.
2. *Буч Г.* Объектно-ориентированное проектирование с примерами применения: учебник / Г. Буч. – М.: Конкорд, 1992. – 519 с.
3. *Буч Г.* Язык UML. Руководство пользователя: пер. с англ. / Г. Буч, Д. Рамбо, А. Джекобсон. – М.: ДМК, 2000. – 432 с.
4. *Береза А.М.* Основи створення інформаційних систем: навч. посібник, 2-ге вид., перероб. і доп. / А.М. Береза. – К.: КНЕУ, 2001. – 214 с.
5. *Гринберг А.С.* Информационные технологии управления: учеб. пособие / А.С. Гринберг, Н.Н. Горбачёв, А.С. Бондаренко. – М.: ЮНИТИ, 2004. – 479 с.
6. *Павленко Л.А.* Корпоративні інформаційні системи: навч. посібник / Л.А. Павленко. – Х.: ВД ІНЖЕК, 2003. – 260 с.
7. *Сендзюк М.А.* Інформаційні системи в державному управлінні: навч. посібник / М.А. Сендзюк. – К.: КНЕУ, 2004. – 339 с.
8. *Системный* анализ в экономике и организации производства / под общ. ред. С.А. Валуева, Н.В. Волковой. – Л.: Политехника, 1991. – 396 с.
9. *Шегда А.В.* Основы менеджмента / А.В. Шегда. – К.: Товаринство «Знання», КОО, 2008. – 512 с.
10. *Офіційний* сайт спільноти користувачів CASE – засобу Rational Rose [*Електонний ресурс*]. – Режим доступу: https://www.ibm.com/ (дата звернення: 01.10.2018).

Навчально-методичне видання

ІНСТРУМЕНТАЛЬНІ ПРОГРАМНІ ЗАСОБИ РОЗРОБКИ ІУС

Методичні вказівки та завдання
до виконання лабораторних робіт
для студентів спеціальності 015 «Професійна освіта»

Укладач: **Київська** Катерина Іванівна

Комп'ютерне верстання *Р.В. Шушпанової*

Підписано до друку Формат 60 х 84 1/16
Умовн.-друк. арк. Обл.-вид. арк.
Електронний документ. Вид. №

Видавець і виготовлювач
Київський національний університет будівництва і архітектури

Повітрофлотський проспект, 31, Київ, 03680
Свідоцтво про внесення до Державного реєстру суб'єктів
видавничої справи ДК № 808 від 13.02.2002 р.