Використання *UML* та *Rational Rose (RR)* при проектуванні ПС. Діаграми взаємодії. Діаграми класів

Зміст

- Реалізація прецедентів. Використання діаграм послідовностей
- Анатомія діаграм послідовності
- Двохетапне розроблення діаграм послідовностей
- Узгодженість (цілісність) моделей
- Використання класів при проектуванні ПС
- Класи етапу аналізу:
  - прикордонні (*boundary*) або інтерфейсні класи;
  - класи-сутності (*entity*);
  - управляючі (*control*) класи (класи-менеджери).
- Класи етапу проектування
- Діаграми співробітництва
- Відношення між класами та їх виявлення:
  - узагальнення;
  - залежність;
  - асоціація;
  - агрегація;
  - композиція.
- Проектування класів, відношень між класами
- Пакетування класів

Спрощена стратегія використання *UML*-діаграм при моделюванні ПС

Спочатку для проектованої ПС варто розробити **(1)** ***діаграму прецедентів . . .***

Етап "Вимоги до ПС"

Етап "Аналіз"

Подальша робота над проектом може здійснюватись на основі ***моделі прецедентів.*** Зокрема, за прецедентами доцільно розробити **(2)** ***діаграми взаємодії***, якими уточнюється динамічні аспекти системи. Паралельно виявляються задіяні в такій реалізації прецедентів ***об'єкти*** і, враховуючи відношення між ними, розробляються **(3)** ***діаграми класів***.

Етап "Проектування"

Діаграми класів можуть використовуватись для **(4)** ***генерації каркасного програмного коду***.

Роль основних сценаріїв

Відштовхуючись від основних сценаріїв прецедентів можуть здійснюватись ***подальші кроки*** на шляху моделювання і, загалом, розроблення ПС: за ***основними сценаріями*** рекомендується розробляти ***діаграми послідовності***, паралельно виявляючи ***класи аналізу***.

Діаграми поведінки

Для опису динаміки використовуються ***діаграми поведінки (behavior diagrams)***, що підрозділяються на

- ***діаграми взаємодії (interaction diagrams),*** які у свою чергу підрозділяються на
  - ***діаграм послідовності (sequence diagrams);***
  - ***діаграм кооперації (співробітництва) (collaboration diagrams).***
- ***діаграми станів (statechart diagrams);***
- ***діаграми діяльності (активності) (activity diagrams);***

Реалізація прецедентів. Використання діаграм послідовностей

Прецедент можна розглядати як ***набір основних сценаріїв***. Кожен сценарій ***реалізується*** сукупністю ***об'єктів***, що ***взаємодіють*** між собою. (Сукупність об'єктів, що взаємодіють між собою є ***кооперацією***).

Діаграма послідовності надає можливість ***представляти сценарій*** прецедента ***графічно***, шляхом відображення ***послідовності повідомлень***, якими обмінюються ***об'єкти***.

Потік (***послідовність***) подій ===> ***Послідовність повідомлень***

Таким чином, виходячи з опису сценарію (***потоку*** подій), пропонується розробляти відповідну ***діаграму послідовності***.

Діаграми послідовностей та виявлення об’єктів

Виявлення об’єктів при розробленні діаграми слід розглядати як ***важливий крок*** на шляху до розроблення ***діаграм класів.***

Під час проектування ***діаграми послідовностей*** та ***діаграми класів*** доцільно розробляти паралельно (одночасно).

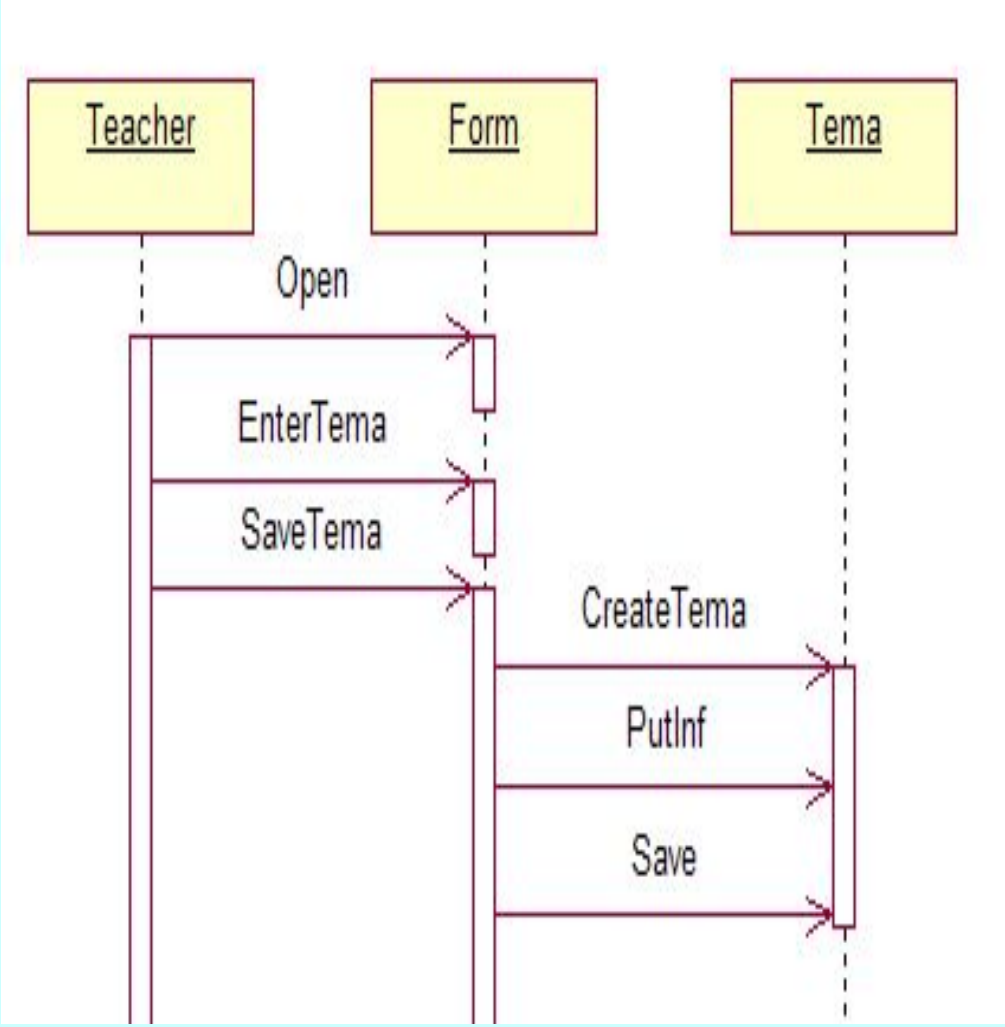
Як виявляти об'єкти?

Один з варіантів – шляхом дослідження іменників у сценаріях.

Але іноді атрибути ***об’єктів*** також “переростають” в об'єкти.

Анатомія діаграм послідовності

- Об'єкти зображуються у вигляді прямокутників і розміщуються над лініями життя (***lifeline***).
- Лінії життя зображаються вертикальними лініями.
- Часовий напрямок – згори вниз.
- Повідомлення позначаються горизонтальними стрілками з назвою повідомлення.
- Відправник та одержувач повідомлення визначаються за напрямком стрілки.
- Активізація об'єкту (***focus of control***) – прямокутник уздовж лінії життя.



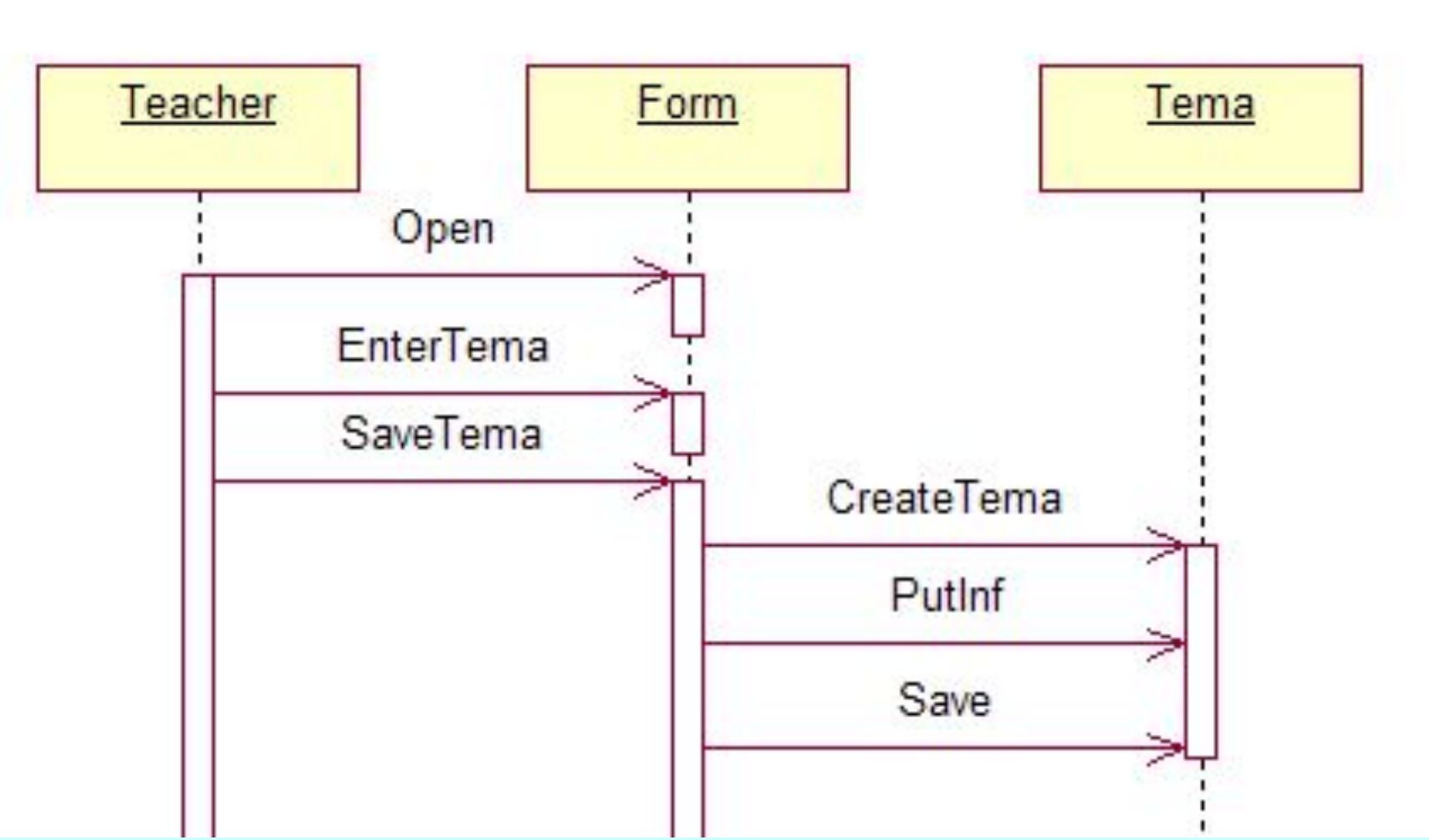
Два етапи розроблення діаграм послідовностей

При проектуванні діаграм послідовностей доцільно використовувати два етапи.

На ***першому етапі*** діаграми розробляються на більш високому рівні абстракції (зокрема цей рівень є цілком прийнятним для стейкхолдерів при потребі подальшого уточнення вимог). Основною задачею цього етапу є виділення ***об'єктів*** та відображення подій окремого основного сценарію у ***послідовність повідомлень***, якими обмінюються ***об'єкти***.

На ***другому етапі*** додатково розробляються (об'єктні) ***класи*** та уточнюються ***класові операції***, що співставляються повідомленням. При цьому *IBM RR* дозволяє відслідковувати цілісність (узгодженість) моделей.

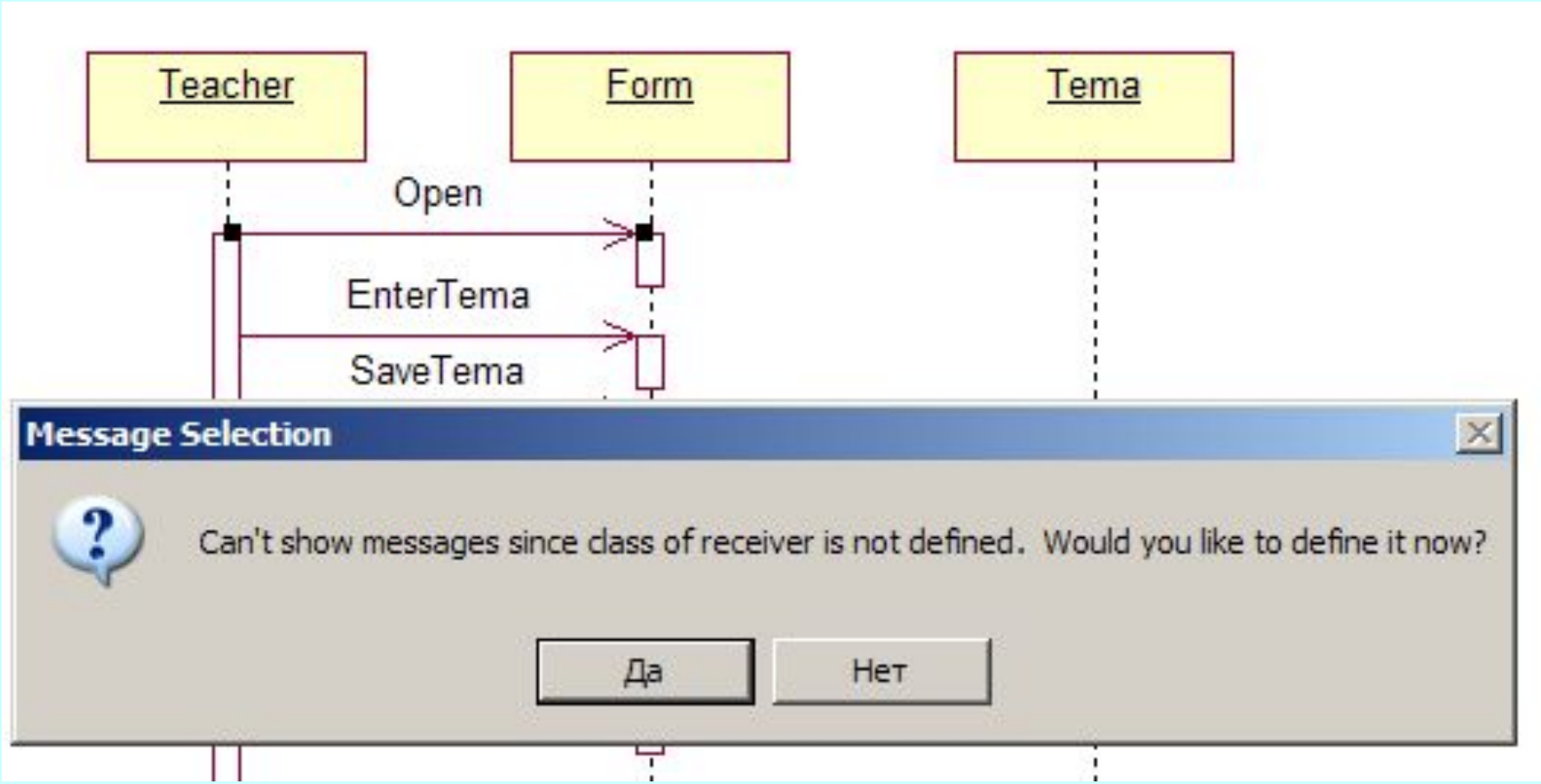
Діаграма послідовності до сценарію "Викладач формулює тему дипломної роботи" (перший етап)



Діаграма послідовності. Можливі попередження системи *IBM RR* (відслідковування цілісності ПС)

Цілісність (узгодженість) моделей:

- об'єкти – примірники ***визначених*** класів (визначених у діаграмі класів);
- повідомлення відповідають операціям класів.



Використання класів при проектуванні ПС. Етап аналізу

- Класи етапу ***аналізу***:
  - прикордонні (***boundary***) або інтерфейсні класи;
  - класи-сутності (***entity***);
  - управляючі (***control***) класи (класи-менеджери).

- Класи етапу ***проектування***:
  - це класи, що можуть залежати від мови програмування (*TForm*, *TDialog* тощо)
  - додаткові класи, задіяні при проектуванні інтерфейса користувача (у формах);
  - допоміжні класи, що додаються на етапі проектування (наприклад, для збереження паролів доцільно використати таблицю).

Клас – абстракція, що описує групу об'єктів із загальними:
- властивостями (атрибутами);
- поведінкою (операціями);
- відношеннями з об'єктами інших класів;
- семантикою, зокрема відповідальностями.

Класи етапу аналізу
Засоби розширення *UML*. Стереотипи

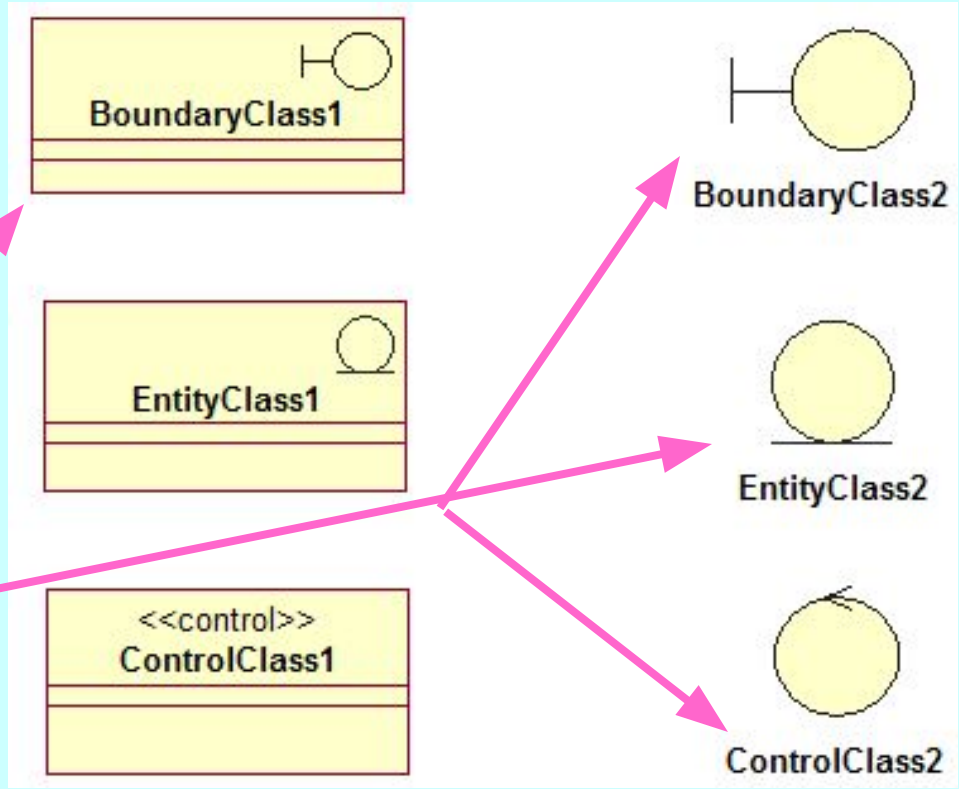
Стереотипи дозволяють створювати ***нові будівельні блоки*** як похідні від існуючих, але ***більш специфічні*** для розв'язуваної задачі. Стереотипи ***розширюють словник*** *UML*.

Приклад. (Профіль для розробки ПС). ***Стереотипи класів*** для етапу аналізу ПС:

- ***прикордонні*** (***boundary***) або інтерфейсні класи;
- класи-***сутності*** (***entity***);
- ***управляючі*** (***control***) класи (класи-менеджери).

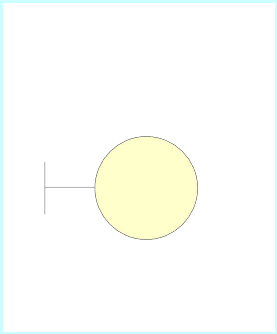
Stereotype Display (RationalRose):

- ***Decoration;***
- ***Icon;***
- ***Label.***



Прикордонні (*boundary*) класи.

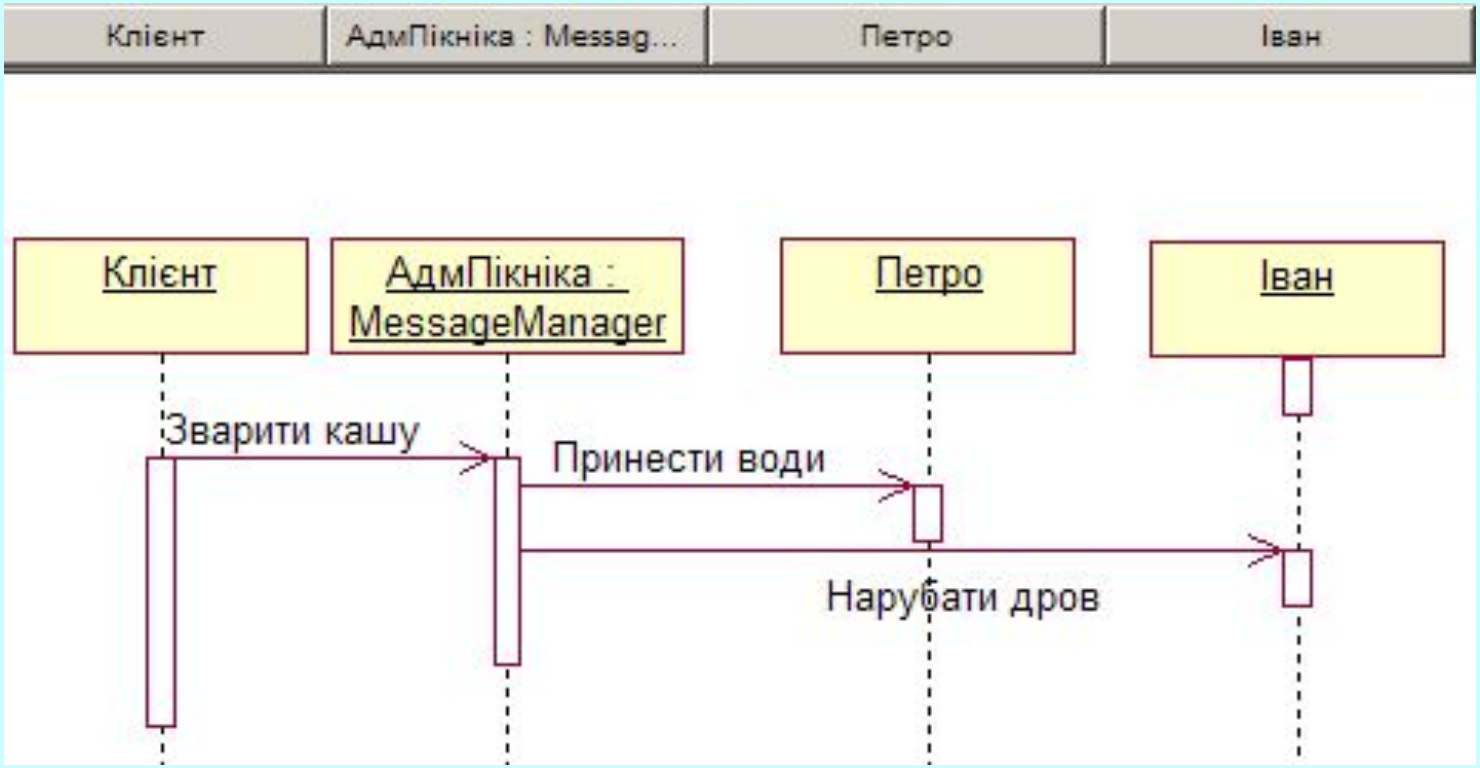
Принцип відокремлення інтерфейсу користувача від бізнес-логіки.



- Прикордонні (***boundary***) або інтерфейсні класи моделюють ***взаємодію*** (***інтерфейс***) між основним актором та прецедентом (зовнішньо залежна частина ПС, яка навряд чи може бути використана в інших системах).
- **Як визначати?** – Рекомендується створювати щонайменше по одному ***boundary***-класу (форма, діалогове вікно) на кожний зв'язок (<<communication>>) ***основний актор – прецедент*** у випадку, коли ініціатором зв'язку є основний актор. (Іноді, зрозуміло, можна обійтись однією формою для кількох пар ***основний актор – прецедент***).
- На етапі аналізу ***boundary***-класи можуть розглядатись просто як "порожні" форми чи діалогові вікна.

Управляючі (*control*) класи.

- ***Управляючі*** (***control***) класи або ***класи-менеджери*** відповідають за координацію дій, поведінки (об'єктів) у процесі реалізації деякої функціональності ПС, зокрема у процесі реалізації функціональності деякого прецеденту.
- **Як визначати?** – Рекомендується створювати ***по одному*** *control*-класу – ***менеджеру повідомлень*** – на кожний прецедент.

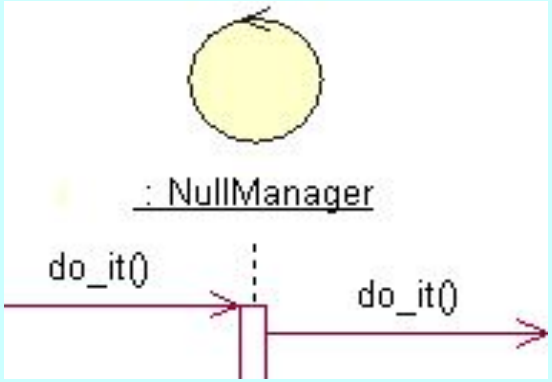


Менеджери повідомлень Додаткові рекомендації

Принцип відокремлення бізнес-логіки від логіки черговості повідомлень.

Додаткові рекомендації з використання менеджерів повідомлень:

- іноді менеджери повідомлень для різних прецедентів можна об'єднати в один, наприклад, коли вони мають справу з однією інформацією, зокрема, з одними "підпорядкованими" об'єктами.
- часто менеджери повідомлень мають тривіальний характер "прийняв повідомлення – передав повідомлення ". У такому випадку їх можна просто вилучити.

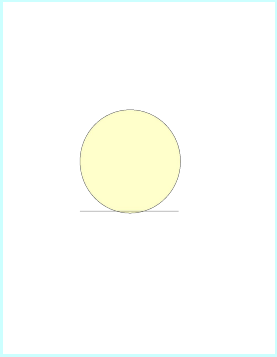


Приклади інших можливих класів-менеджерів у програмних системах

Приклади інших можливих класів-менеджерів у програмних системах:

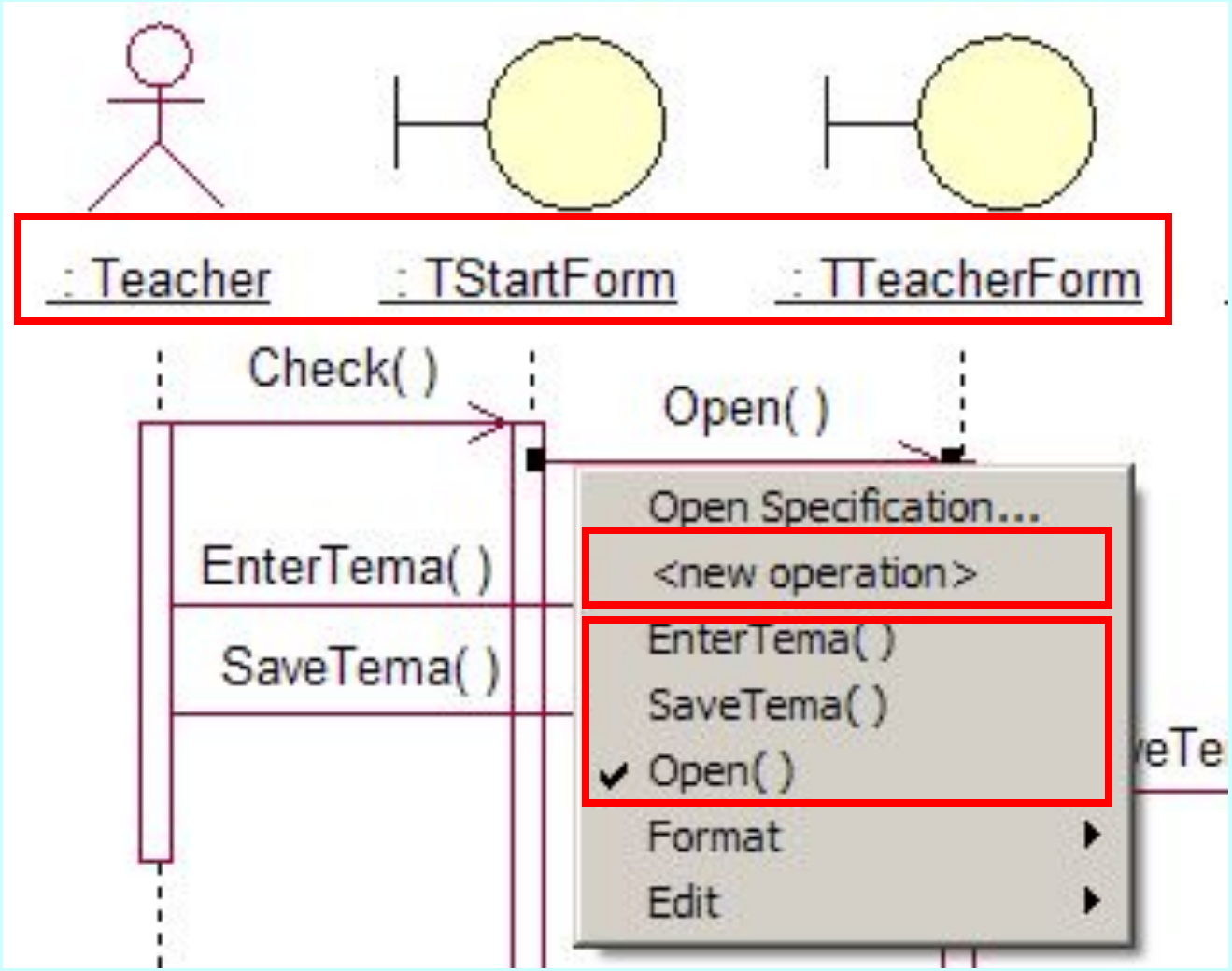
- ***менеджери транзакцій БД*** (вони "знають", як зберігати дані – що та у які таблиці; інкапсулюють особливості роботи з СУБД, забезпечуючи при потребі "безболісний перехід" на іншу СУБД);
- ***менеджери обробки помилок;***
- **менеджери безпеки;**
- **менеджери (контролери) зовнішніх пристроїв.**

Класи-сутності (*entity*)



- моделюють ***ключові абстракції*** предметної області, пов'язані з ***обробкою*** та ***збереженням інформації*** програмною системою (такі ключові абстракції, як правило, є незалежними від конкретної ПС, а отже можуть успішно використовуватись в інших ПС);
- класи-сутності часто містять інформацію, що має постійно (тривалий час) зберігатись в одній чи декількох ***таблицях БД***.

Контекстне меню для повідомлення та узгоджуваність моделей



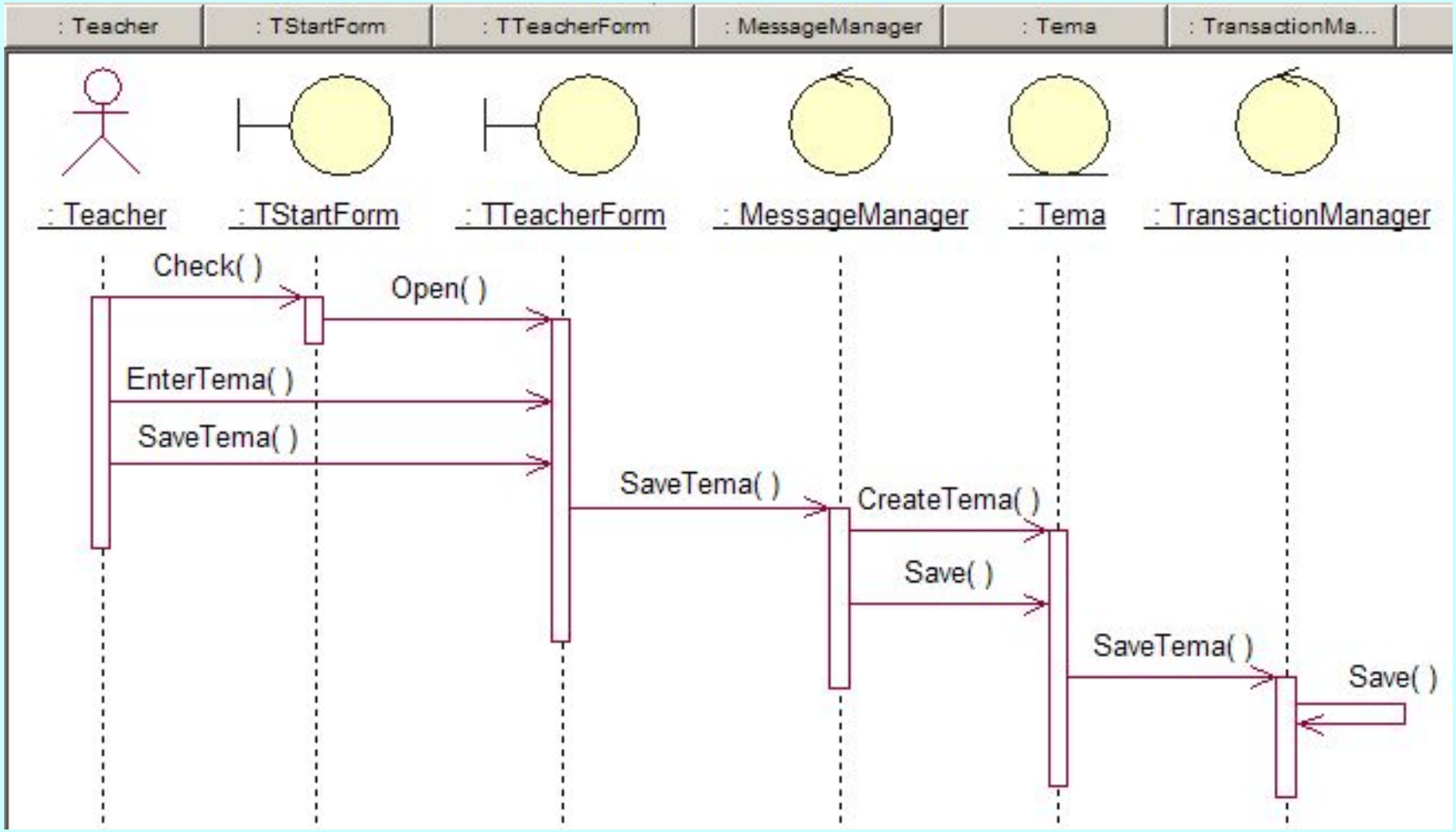
Використання класів при проектуванні ПС. Етап проектування (1/2)

- Класи етапу ***аналізу***:
  - прикордонні (***boundary***) або інтерфейсні класи;
  - класи-сутності (***entity***);
  - управляючі (***control***) класи (класи-менеджери).

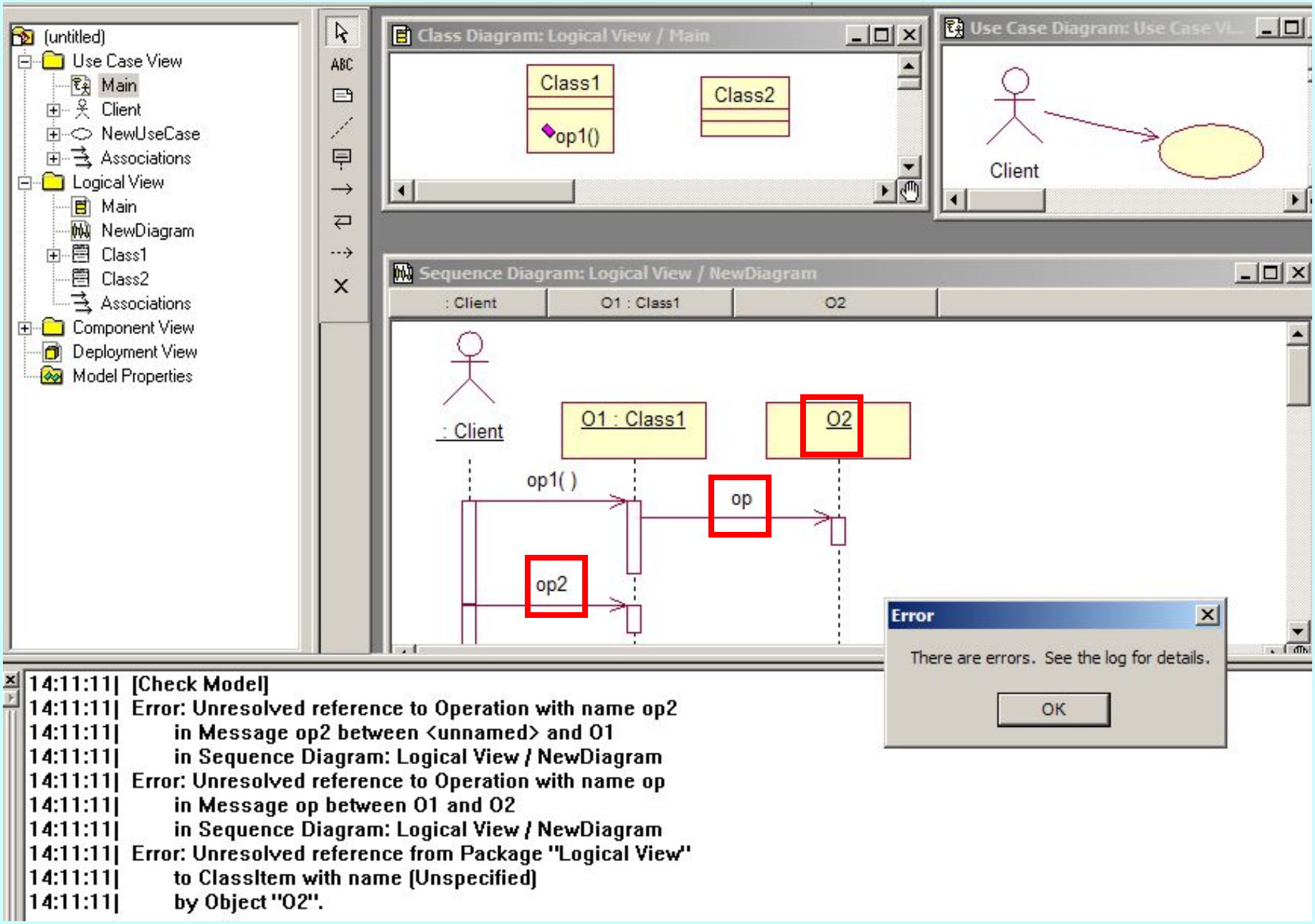
- Класи етапу ***проектування***:
  - це класи, що можуть залежати від мови програмування (*TForm*, *TDialog* тощо)
  - додаткові класи, задіяні при проектуванні інтерфейса користувача (у формах);
  - допоміжні класи, що додаються на етапі проектування (наприклад, для збереження паролів доцільно використати таблицю).

Клас – абстракція, що описує групу об'єктів із загальними:
- властивостями (атрибутами);
- поведінкою (операціями);
- відношеннями з об'єктами інших класів;
- семантикою, зокрема відповідальностями.

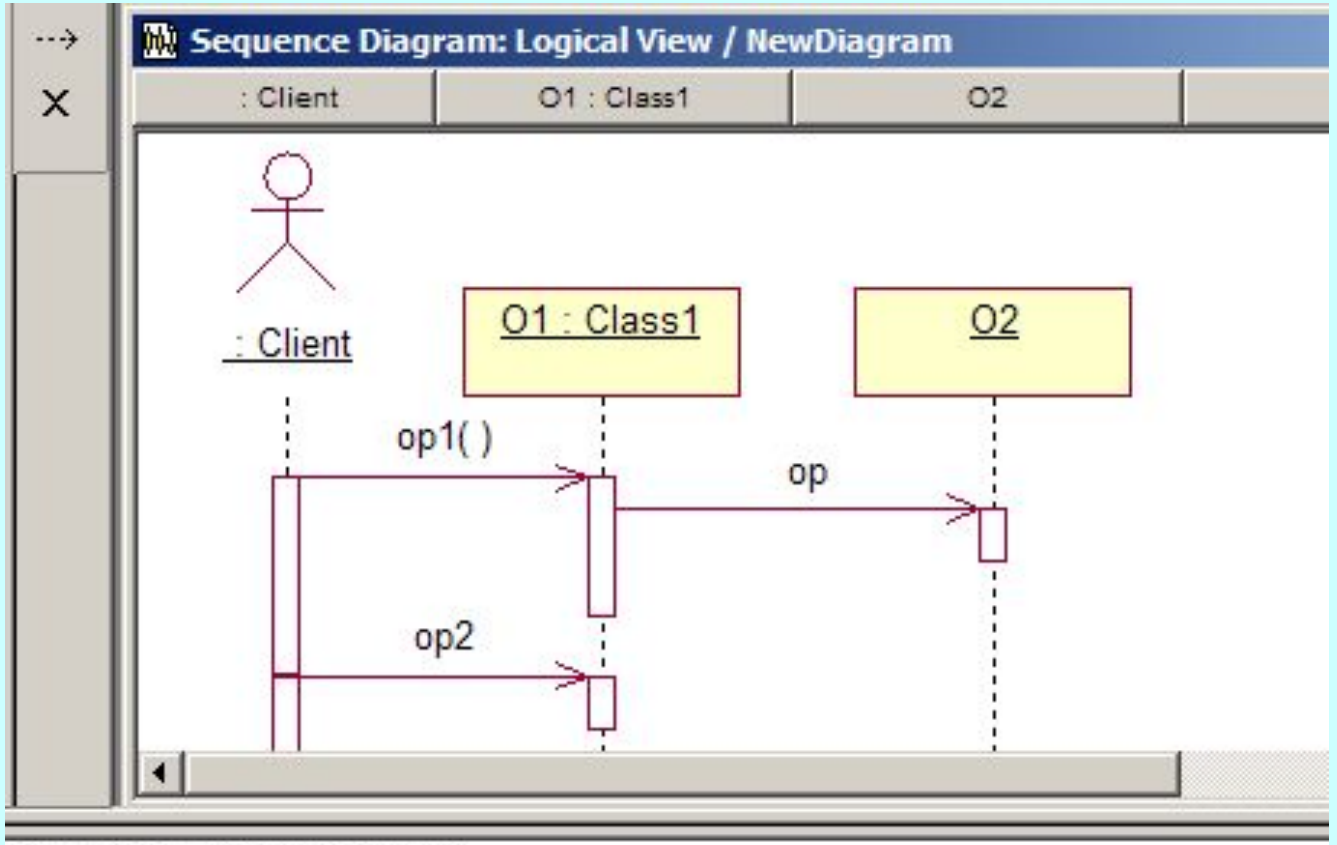
Діаграма послідовності для сценарію “Викладач формулює тему дипломної роботи” (другий етап)



Перевірка узгодженості моделі (*Tools - Check Model*). Приклад

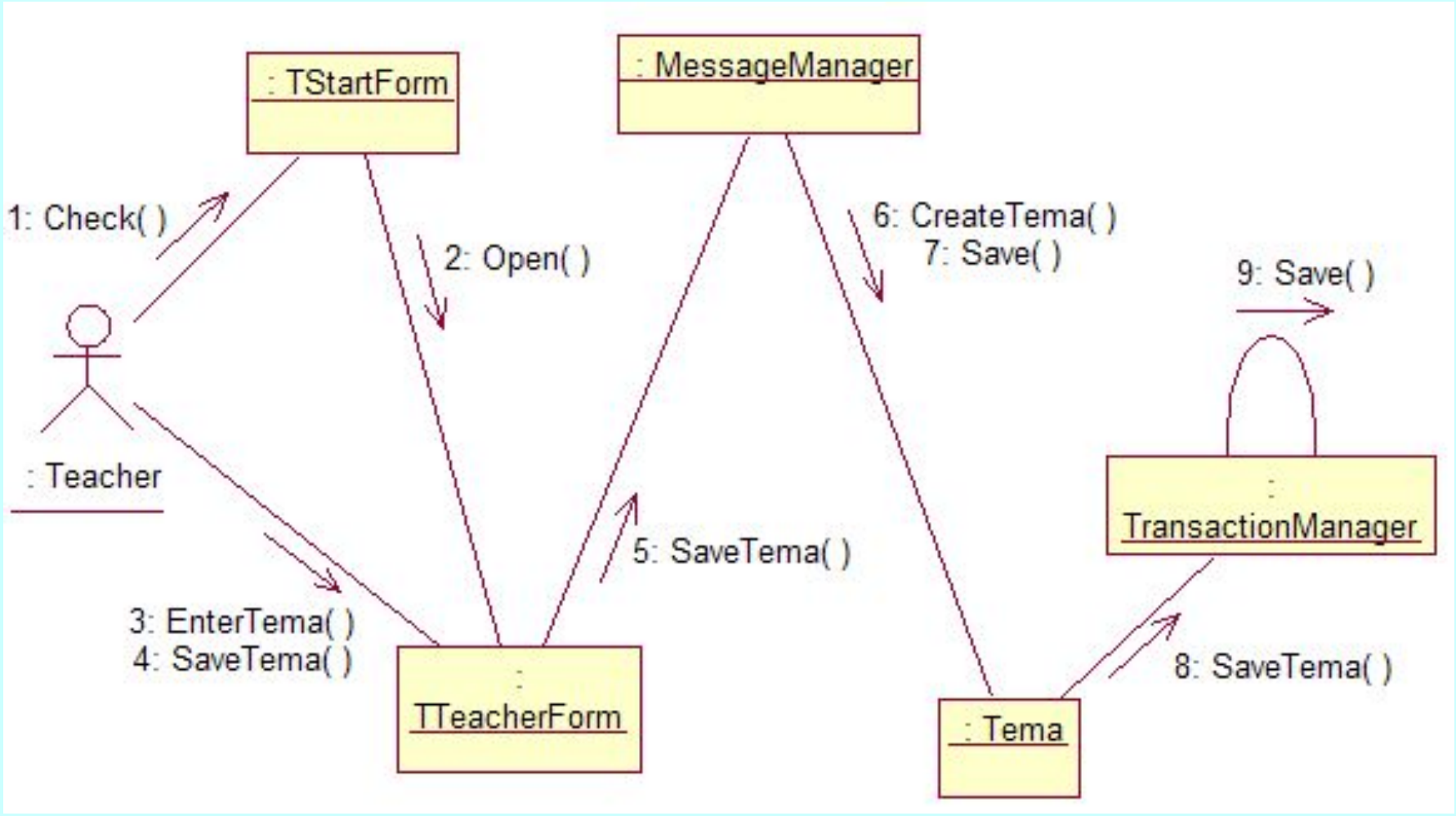


Перевірка моделі. Журнал повідомлень (*log*)




```
14:11:11|  [Check Model]
14:11:11|  Error: Unresolved reference to Operation with name op2
14:11:11|       in Message op2 between <unnamed> and O1
14:11:11|       in Sequence Diagram: Logical View / NewDiagram
14:11:11|  Error: Unresolved reference to Operation with name op
14:11:11|       in Message op between O1 and O2
14:11:11|       in Sequence Diagram: Logical View / NewDiagram
14:11:11|  Error: Unresolved reference from Package "Logical View"
14:11:11|       to ClassItem with name (Unspecified)
14:11:11|       by Object "O2".
```

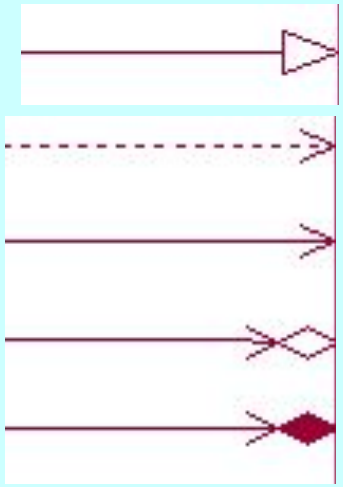
Діаграми співробітництва (*collaboration*)



Діаграми співробітництва та діаграми послідовності

- Діаграми послідовності:
  - більш корисні на етапі аналізу ПС;
  - акцент на часову послідовність повідомлень, якими обмінюються об'єкти;
  - більш зрозумілі для стейкхолдерів (уточнення вимог).
- Діаграми співробітництва:
  - більш корисні на етапі проектування ПС, оскільки відображають зв'язки між об'єктами в цілому – простіше оцінювати вплив на систему при змінюваності об'єктних класів;
  - дозволяють представляти потоки даних.

Відношення між класами

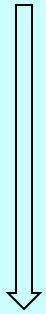


Типи відношень між класами:

- ***узагальнення;***
- ***залежність;***
- ***асоціація;***
- ***агрегація;***
- ***композиція.***

Ієрархія обмежень на "класові" відношення:

- ***залежність;***
- ***асоціація;***
- ***агрегація;***
- ***композиція.***



Напрямок посилення обмежень.

Відношення між класами та їх виявлення. Відношення залежності

Клас *A* залежить від класу *B*, якщо при вилученні класу *B* клас *A* вже ***не зможе забезпечити*** виконання покладених на нього ***відповідальностей***.

(Залежність може бути пряма та опосередкована: клас *A* --> клас *B* --> клас *C*).

Варіанти умов, коли має місце (пряма, безпосередня) ***залежність класів ПС***:

- наявність у першому класі ***поля***, тип якого засновано на другому класі;
- наявність у першому класі ***поля-вказівника***, тип якого засновано на другому класі;
- використання другого класу як типу ***локальної змінної*** чи ***параметра*** для деякої операції першого класу;
- використання ***статичної операції*** другого класу.

Відношення між класами та їх виявлення. Асоціація

У випадку асоціації клас-клієнт (або залежний клас) має ***"інформацію*** про ***місцезнаходження***" об'єкта-постачальника сервісу.

Варіанти умов, коли має місце ***асоціація класів ПС*** (!):

- наявність у першому класі ***поля***, тип якого засновано на другому класі;
- наявність у першому класі ***поля-вказівника***, тип якого засновано на другому класі.

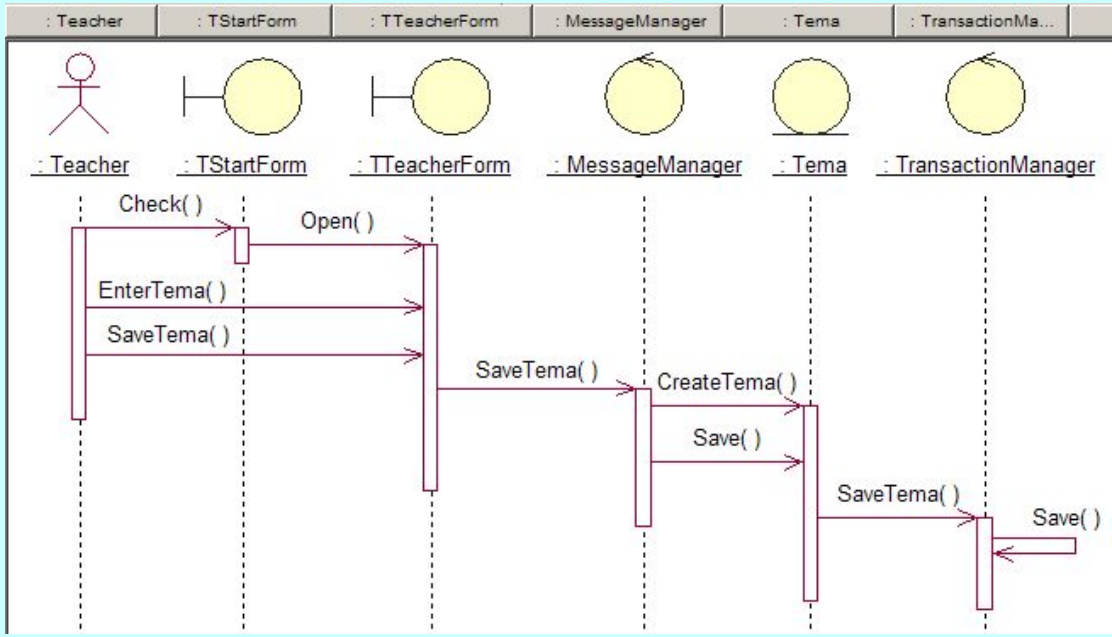
Якщо у діаграмі взаємодій використовується ***повідомлення*** між об'єктами двох класів, то ***рекомендується*** встановити відношення ***асоціації*** між відповідними класами (напрямок – від "клієнта" до "постачальника" сервісу).

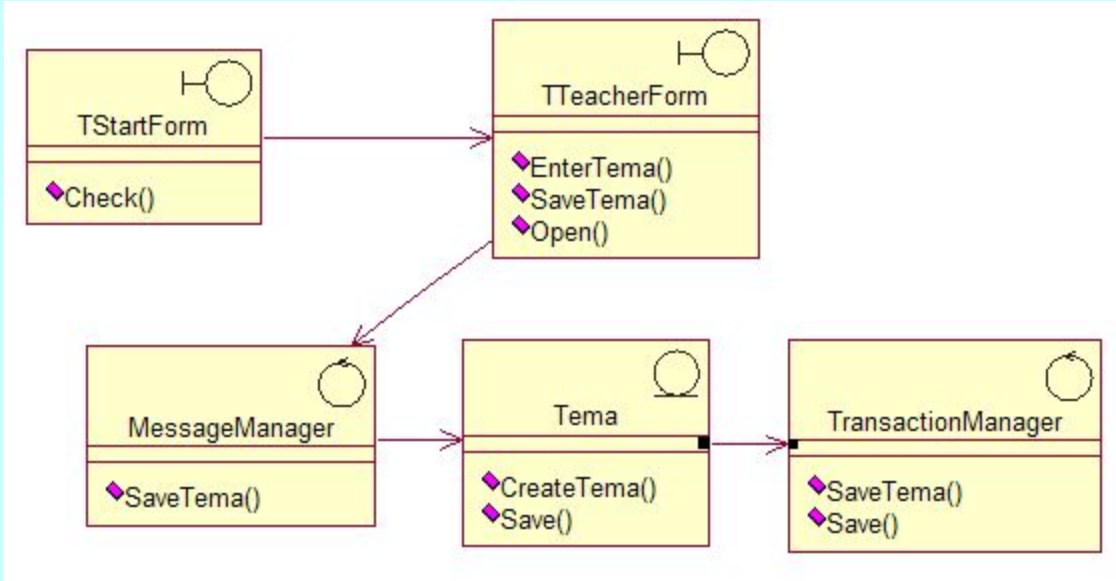
(Увага! Асоціація може бути ***одно***- чи ***двоспрямованою***).

Графічна реалізація прецедентів

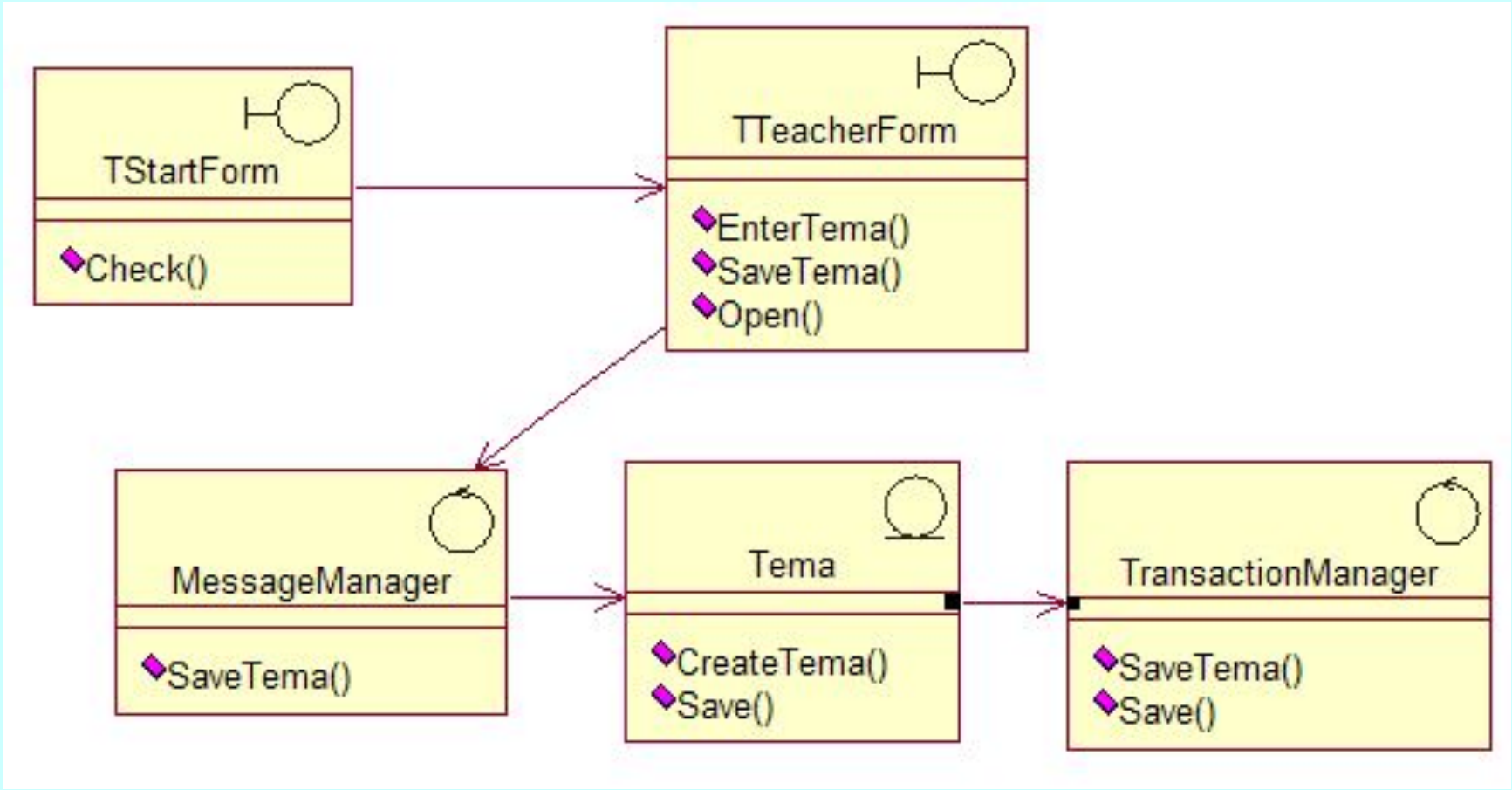
Графічна реалізація прецедентів полягає у створенні:

- однієї чи декількох діаграм взаємодії (послідовності чи кооперації), розроблених у відповідності до основних сценаріїв прецедентів;
- діаграм класів-учасників – ***VOPC*** (***View of Participating Classes***).





Діаграма класів-учасників *VOPC* (*View of Participating Classes*) до сценарію "Викладач формулює тему дипломної роботи" .



Відношення між класами та їх виявлення. Агрегація та композиція

Агрегація – це асоціація з відношенням ”ціле-частина” між класами.

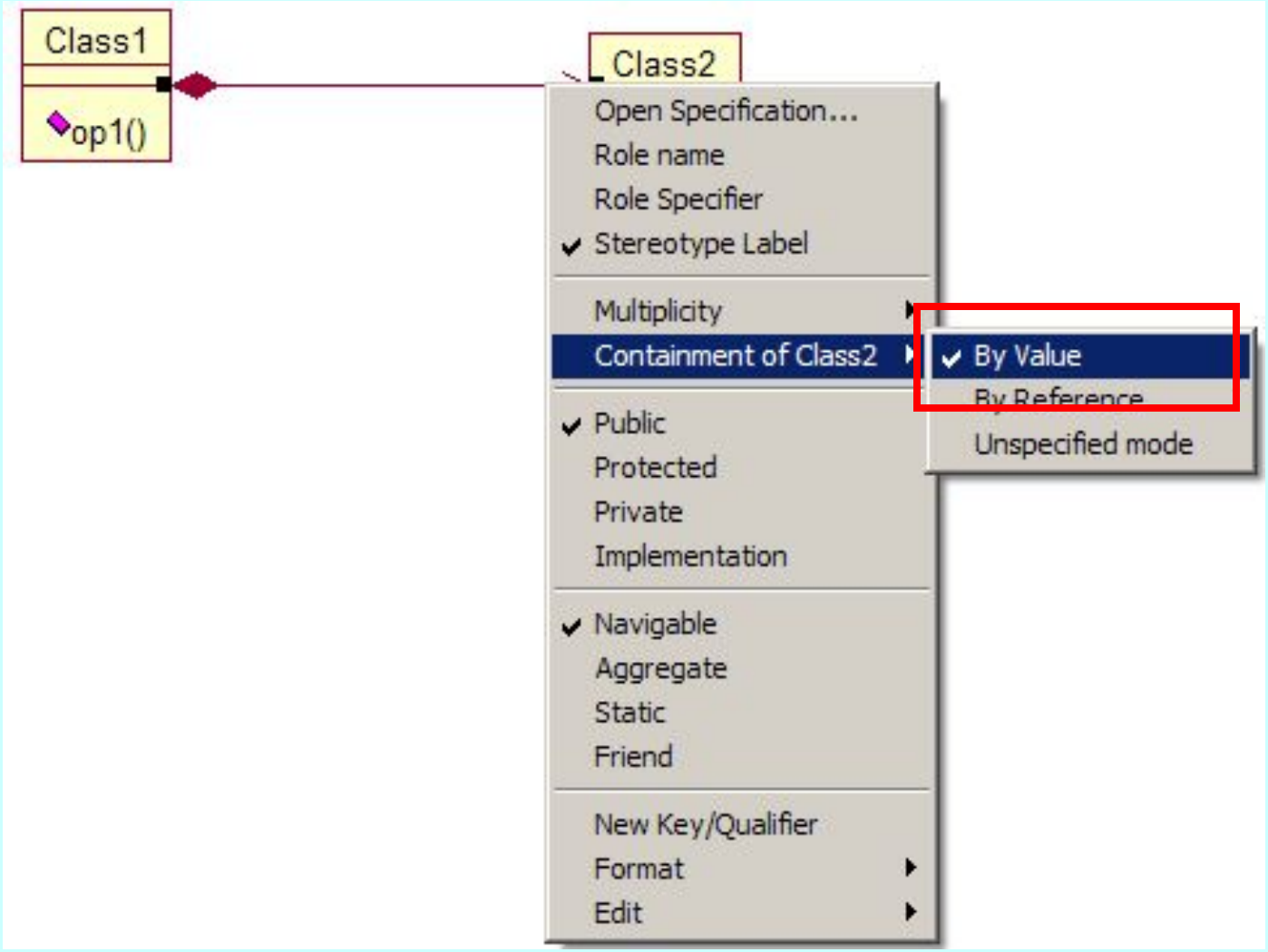
Композиція – це агрегація, коли існування частини повністю залежить від існування цілого (частина не може існувати без цілого). Для класів ПС з композицією пов'язане входження “за значенням” – ***by value*** (див. наступний слайд).

Варіанти умов, коли має місце ***асоціація між класами ПС***:

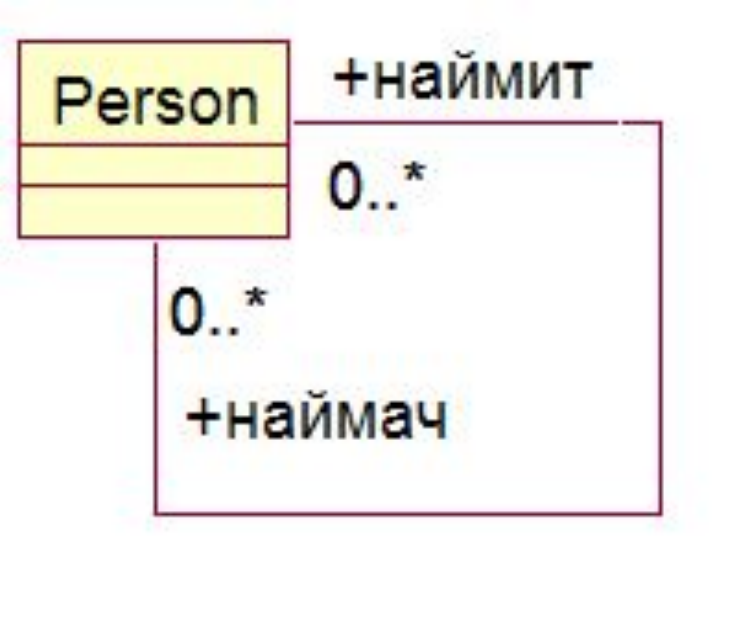
- наявність у першому класі ***поля***, тип якого засновано на другому класі (входження – ***by value***);
- наявність у першому класі ***поля-вказівника***, тип якого засновано на другому класі (входження – ***by reference***).

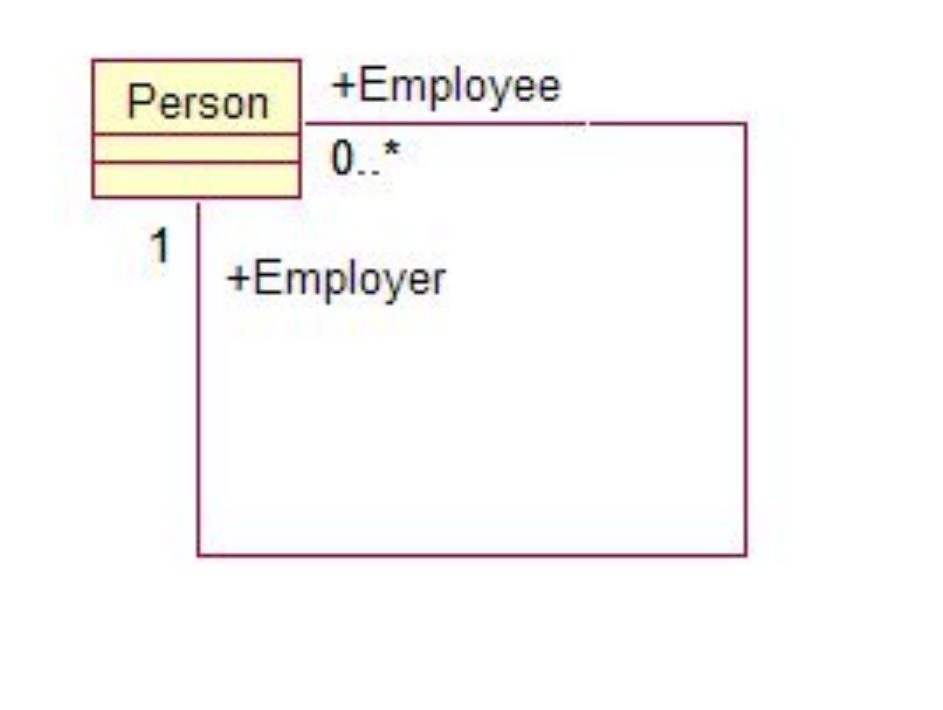
Композиція потребує наявності у першому класі ***поля*** (***не вказівникового***!), тип якого засновано на другому класі (входження – ***by value***).

Композиція у *IBM RR*. Контекстне меню кінця композиції



Рефлексивні асоціації





Проектування відношень між класами

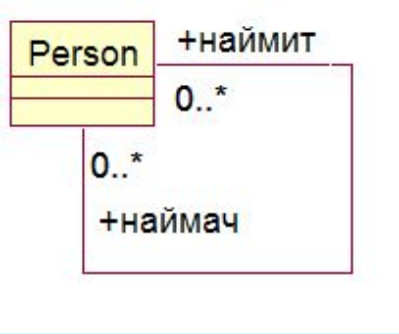
При проектуванні відношень між класами асоціації ***бажано*** створювати ***односпрямованими*** (такі асоціації легше реалізовувати та підтримувати).

Іноді доцільно односпрямовані асоціації перетворити у залежності (вони ще простіші у реалізації). У такому випадку інформація про знаходження постачальника сервісу найчастіше передається як параметр відповідної операції.

Деякі подальші ***важливі кроки*** проектування відношень:

- визначення кратностей (*cardinalities*) кінців відношень між класами;
- використання імен ролей. (Ролева ознака класу у відношенні).

  (Приклад: ролі з іменами “наймит” та “наймач” для класу Person).



Асоціації. Приклад

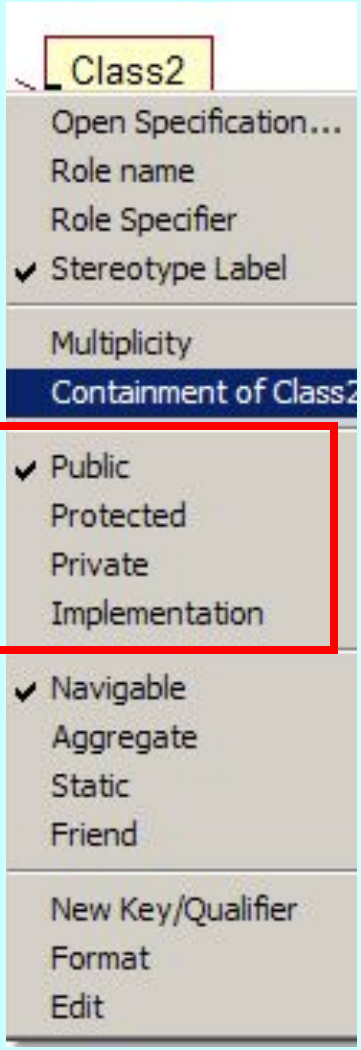
Проектування атрибутів та операцій

Видимість (***visibility***, ***export control***):

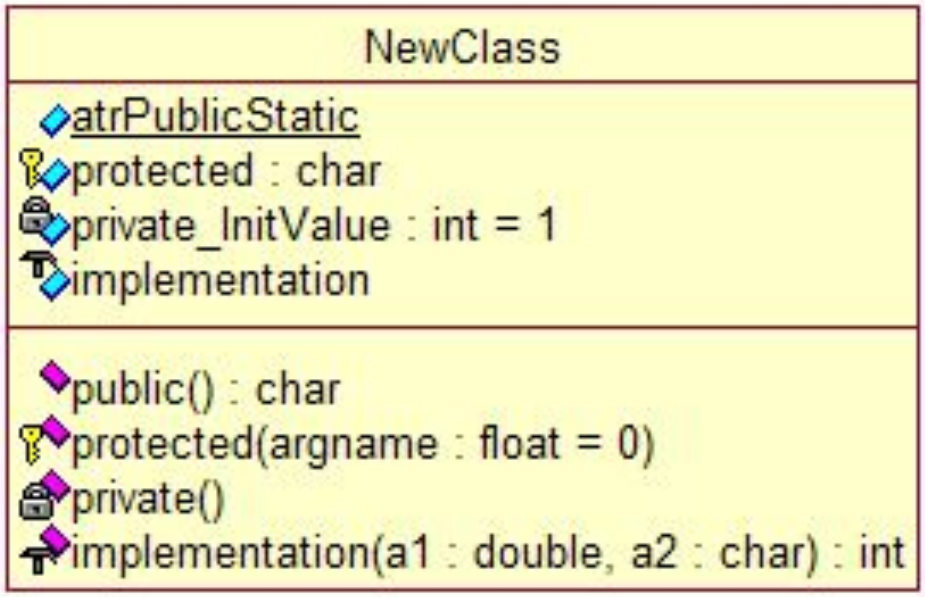
- ***public;***
- ***protected;***
- ***private;***
- ***implementation*** – видимість у межах пакету.

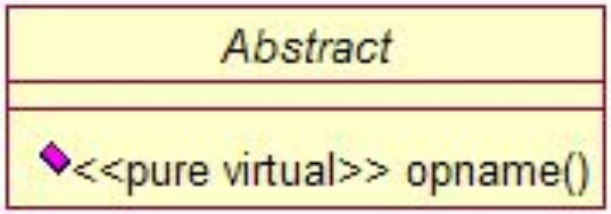
Класифікація операцій:

- операції реалізації (бізнес-функціональності);
- операції управління (конструктори, деструктори);
- операції доступу (***private*** *Val* ===> *SetVal, GetVal*);
- допоміжні операції (звичайно фігурують як ***private*** чи ***protected***).



До специфікації класів





Пакетування класів

Пакетування класів (організація класів у пакети) дозволяє отримувати моделі більш високого рівня абстракції.

Принципи організації класів у пакети:

1) виходячи з ***логіки, функціональності, відповідальності*** тощо (наприклад, класи з даними про особи, класи з даними рівня кафедри, класи з даними рівня факультету, класи звітів, класи безпеки, класи для обробки помилок);

2) виходячи зі ***стереотипів***, зокрема трьох стереотипів класів аналізу (***boundary, control*** та ***entity*** класи аналізу);

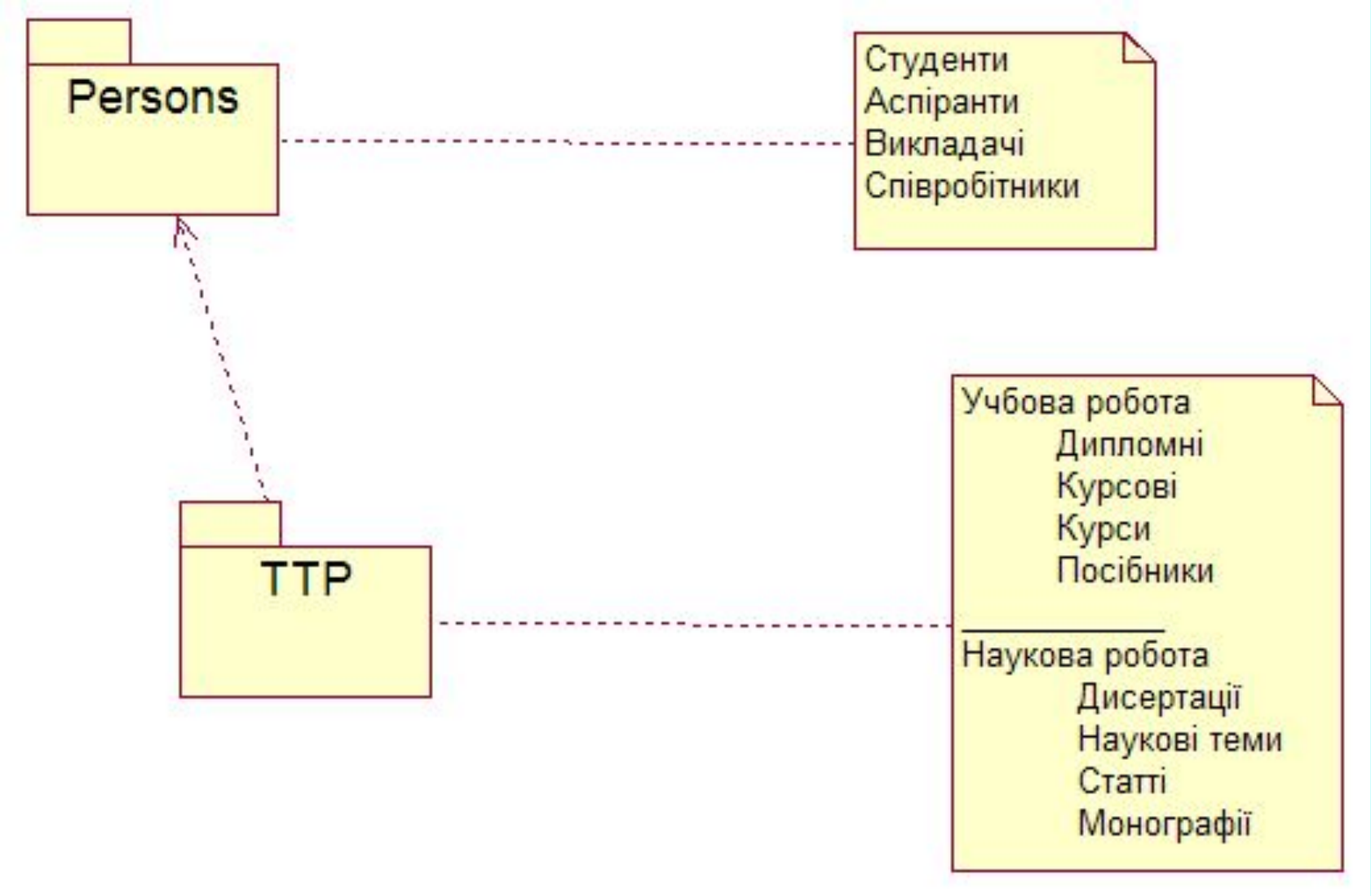
3) комбінуючи перші два на різних рівнях (наприклад, на верхньому – за логікою, на нижньому – за стереотипами).

Рекомендації до розроблюваних діаграм класів:

- головна діаграма – на рівні пакетів;
- кожен пакет має власну головну діаграму (“розкривається” – *DubleClick*).

Залежність між пакетами визначається залежністю між класами з цих пакетів.

Пакетування класів (приклад)



Пакетування класів (приклад)