СТАНДАРТ UML ВЕРСИИ 1.1-1

Стандарт UML версии 1.1, принятый 0MG в 1997 предлагает следующий набор диаграмм для моделирования:

* ***диаграммы вариантов использования (use case diagrams)*** — для моделирования бизнес-процессов организации (требований к системе);
* ***диаграммы классов (class diagrams)*** — для моделирования статической структуры классов системы и связей между ними;
* ***диаграммы поведения системы (behavior diagrams);***
* ***диаграммы взаимодействия (interaction diagrams)*** — для моделирования процесса обмена сообщениями между объектами. Существуют два вида диаграмм взаимодействия:
  * ***диаграммы последовательности (sequence diagrams);***
  * ***кооперативные диаграммы (collaboration diagrams);***

СТАНДАРТ UML ВЕРСИИ 1.1-2

* ***диаграммы состояний (statechart diagrams)*** — для моделирования поведения объектов системы при переходе из одного состояния в другое;
* ***диаграммы деятельностей (activity diagrams)*** — для моделирования поведения системы в рамках различных вариантов использования или моделирования деятельностей;
* ***диаграммы реализации (implementation diagrams):***
  * ***диаграммы компонентов (component diagrams)*** — для моделирования иерархии компонентов (подсистем) системы;
  * ***диаграммы размещения (deployment diagrams)*** — для моделирования физической архитектуры системы.

ВАРИАНТ ИСПОЛЬЗОВАНИЯ

Ивар Якобсон впервые ввел понятие "вариант использования" (use case) и придал ему такую значимость, что он превратился в основной элемент разработки и планирования проекта.

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой.

В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать, или целей, которые он преследует по отношению к разрабатываемой системе.

ПРИМЕР ВАРИАНТА ИСПОЛЬЗОВАНИЯ

Например, два типичных варианта использования обычного текстового процессора — **"сделать некоторый текст полужирным" и "создать индекс"**.

Даже на таком простом примере можно выделить ряд свойств варианта использования**:**

- **он охватывает некоторую очевидную для пользователей функцию,**
- **может быть как небольшим, так и достаточно крупным и**
- **решает для пользователя некоторую дискретную задачу.**

В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех, функций, которые он хотел бы реализовать.

ДЕЙСТВУЮЩЕЕ ЛИЦО

Действующее лицо (actor) - это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ.

Действующее лицо может также быть внешней системой, которой необходима информация от данной системы. Показывать на диаграмме действующих лиц следует только в том случае, когда им действительно необходимы некоторые варианты использования.

Действующие лица делятся на три основных типа –

- **пользователи системы,**
- **другие системы, взаимодействующие с данной, и**
- **время**. Время становится действующим лицом, если от него зависит запуск каких-либо событий в системе.

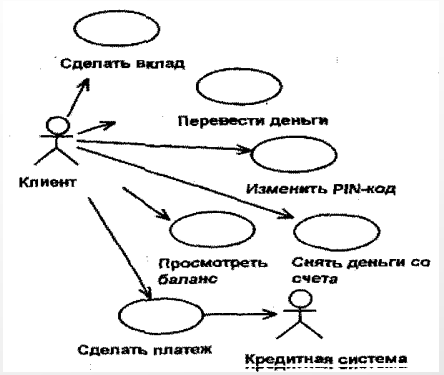
ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Для наглядного представления вариантов использования в качестве основных элементов процесса разработки ПО применяются **диаграммы вариантов использования**.

На такой диаграмме **человеческие фигурки обозначают действующих лиц, овалы - варианты использования, а линии и стрелки - различные связи между действующими лицами и вариантами использования**.

На диаграмме показано взаимодействие между вариантами использования и действующими лицами. **Она отражает требования к системе с точки зрения пользователя.** Варианты использования - это функции, выполняемые системой, а действующие лица - это заинтересованные лица (stakeholders) по отношению к создаваемой системе. Эти диаграммы показывают, какие действующие лица инициируют варианты использования.

ПРИМЕР ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

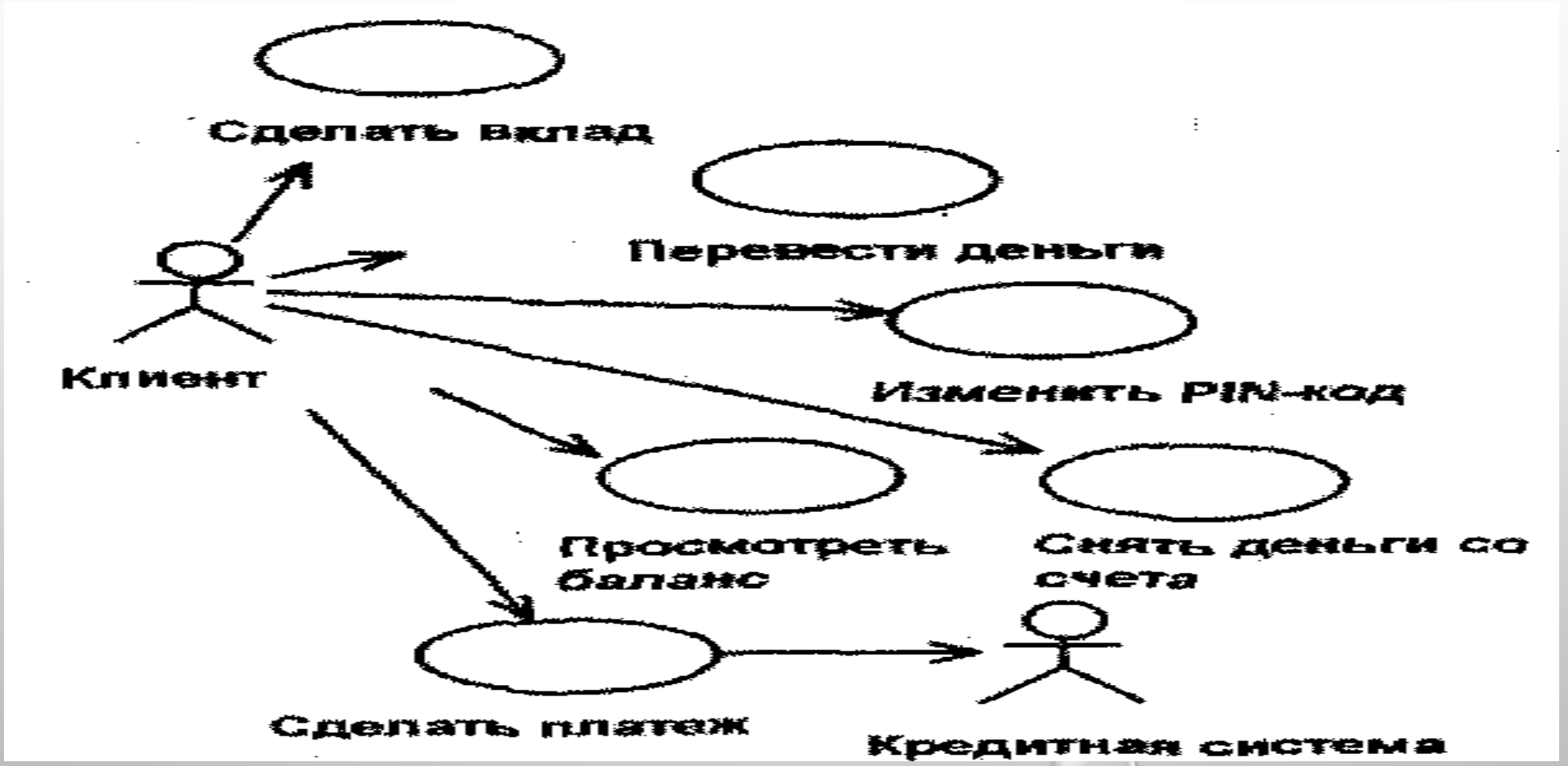


Рассмотрим диаграмму для банковской системы с банкоматами.

На диаграмме показаны **два действующих лица: клиент и кредитная система**. Также есть **шесть основных действий**, выполняемых моделируемой системой: **перевести деньги, сделать вклад, снять деньги со счета, просмотреть баланс, изменить PIN-код и сделать платеж**.

Стрелка показывает, что вариант использования предоставляет некоторую информацию, используемую действующим лицом. В данном случае вариант использования «Сделать платеж» предоставляет «Кредитной системе» информацию об оплате по кредитной карточке.

ПРИМЕР ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ



ПРАВИЛА ПРИ РАЗРАБОТКЕ ДИАГРАММ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Разрабатывая диаграммы вариантов использования, старайтесь придерживаться следующих правил:

* **не моделируйте связи между действующими лицами.** По определению действующие лица находятся вне сферы действия системы. Это означает, что связи между ними также не относятся к ее компетенции;

* **не соединяйте стрелкой два варианта использования непосредственно**. Диаграммы данного типа описывают, какие варианты использования доступны системе, а не порядок их выполнения. Для отображения порядка выполнения вариантов использования применяют диаграммы деятельностей;

• **каждый вариант использования должен быть инициирован действующим лицом**. Это означает, что всегда имеется стрелка, начинающаяся на действующем лице и заканчивающаяся на варианте использования.

USE CASE DIAGRAM

Конкретная **цель диаграмм вариантов использования - это документирование вариантов использования** (все, входящее в сферу применения системы), действующих лиц (все вне этой сферы) и связей между ними.

Хорошим **источником для идентификации вариантов использования служат внешние события.** Следует начать с перечисления всех событий, происходящих во внешнем мире, на которые система должна каким-то образом реагировать. Какое-либо конкретное событие может повлечь за собой реакцию системы, не требующую вмешательства пользователей, или, наоборот, вызвать чисто Пользовательскую реакцию. Идентификация событий, на которые необходимо реагировать, помогает идентифицировать варианты Использования.

ПОТОК СОБЫТИЙ

Для того чтобы фактически разработать систему, требуются конкретные детали. Эти детали описываются в документе, называемом «поток событий» (flow of events). **Целью потока событий является документирование процесса обработки данных, реализуемого в рамках варианта использования.** Этот документ подробно описывает, что будут делать пользователи системы и что - сама система.

Поток событий не должен зависеть от реализации. Цель - описать то, что будет делать система, а не как она будет делать это. **Обычно поток событий включает**:

- **краткое описание;**
- **предусловия (pre-conditions);**
- **основной поток событий;**
- **альтернативный поток событий;**
- **постусловия (post-conditions).**

ПОТОК СОБЫТИЙ. КРАТКОЕ ОПИСАНИЕ, ПРЕДУСЛОВИЯ

Каждый вариант использования должен иметь **краткое описание** того, что он будет делать. Например, вариант использования «Перевести деньги» может содержать следующее описание: Вариант использования «Перевести деньги» позволяет клиенту или служащему банка переводить деньги с одного счета до востребования или сберегательного счета на другой.

Предусловия варианта использования - это такие условия, которые должны быть выполнены, прежде чем вариант использования начнет выполняться сам. Например, таким условием может быть выполнение другого варианта использования или наличие у пользователя прав доступа, требуемых для запуска этого. Не у всех вариантов использования бывают предусловия. С помощью предусловий можно документировать порядок выполнения вариантов использования. Так, предусловием одного варианта использования может быть то, что в это время должен выполняться другой.

ПОТОК СОБЫТИЙ: ОСНОВНОЙ И АЛЬТЕРНАТИВНЫЙ

Конкретные детали вариантов использования описываются в основном и альтернативных потоках событий. Поток событий поэтапно описывает, что должно происходить во время выполнения заложенной в вариант использования функциональности. **Поток событий показывает, что будет делать система, а не как она будет делать это**, и описывает все это с точки зрения пользователя.

Основной и альтернативный потоки событий включают следующее описание:

- каким образом запускается вариант использования;
- различные пути выполнения варианта использования;
- нормальный, или основной, поток событий варианта использования;
- отклонения от основного потока событий (так называемые альтернативные потоки);
- потоки ошибок;
- каким образом завершается вариант использования

ПОТОК СОБЫТИЙ
СНЯТЬ ДЕНЬГИ СО СЧЕТА

Например, **поток событий варианта использования «Снять деньги со счета»** может выглядеть следующим образом:

Основной поток:

1. Вариант использования начинается, когда клиент вставляет карточку в банкомат.
2. Банкомат выводит приветствие и предлагает клиенту ввести свой PIN-код.
3. Клиент вводит PIN-код.
4. Банкомат подтверждает введенный код. **Если код не подтвержден**, выполняется **альтернативный поток событий А1.**
5. Банкомат выводит список доступных действий:

• сделать вклад;

• снять деньги со счета;

• перевести деньги.

ПОТОК СОБЫТИЙ
СНЯТЬ ДЕНЬГИ СО СЧЕТА

Клиент выбирает пункт «Снять деньги со счета».

6. Банкомат запрашивает, сколько денег надо снять.
7. Клиент вводит требуемую сумму.
8. Банкомат определяет, имеется ли на счете достаточно денег. **Если денег недостаточно,** выполняется **альтернативный поток А2**. Если во время подтверждения суммы **возникают ошибки**, выполняется **поток ошибок Е1**.
9. Банкомат вычитает требуемую сумму из счета клиента.
10. Банкомат выдает клиенту требуемую сумму наличными.
11. Банкомат возвращает клиенту его карточку.
12. Банкомат печатает чек для клиента.
13. Вариант использования завершается.

АЛЬТЕРНАТИВНЫЕ ПОТОКИ СОБЫТИЙ

Альтернативный поток А1. Ввод неправильного PIN-кода.

1. Банкомат информирует клиента, что код введен неправильно.
2. Банкомат возвращает клиенту его карточку.
3. Вариант использования завершается.

Альтернативный поток А2. Недостаточно денег на счете.

1. Банкомат информирует клиента, что денег на его счете недостаточно.
2. Банкомат возвращает клиенту его карточку.
3. Вариант использования завершается.

ПОТОК ОШИБОК Е 1

Поток ошибок Е1. Ошибка в подтверждении запрашиваемой суммы .

1. Банкомат сообщает пользователю, что при подтверждении запрашиваемой суммы произошла ошибка, и дает ему номер телефона службы поддержки клиентов банка.
2. Банкомат заносит сведения об ошибке в журнал ошибок. Каждая запись содержит дату и время ошибки, имя клиента, номер его счета и код ошибки.
3. Банкомат возвращает клиенту его карточку.
4. Вариант использования завершается.

ПОТОК СОБЫТИЙ. ПОСТУСЛОВИЯ

Постусловиями называются такие условия, которые всегда должны быть выполнены после завершения варианта использования.

Например, в конце варианта использования можно пометить флажком какой-нибудь переключатель. Информация такого типа входит в состав постусловий.

Как и для предусловий, с помощью постусловий можно вводить информацию о порядке выполнения вариантов использования системы.

Если, например, после одного из вариантов использования должен всегда выполняться другой, это можно описать как постусловие. Такие условия имеются не у каждого варианта использования.

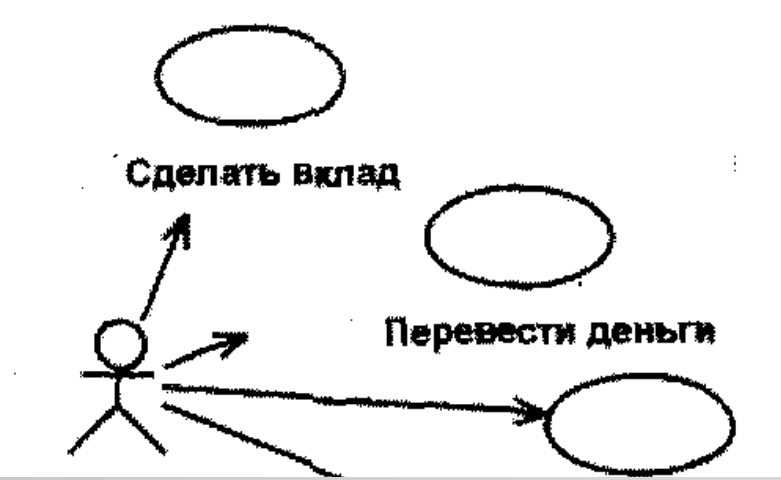
СВЯЗИ МЕЖДУ ВАРИАНТАМИ ИСПОЛЬЗОВАНИЯ И ДЕЙСТВУЮЩИМИ ЛИЦАМИ

В языке UML для вариантов использования и действующих лиц поддерживается несколько типов связей:

- **связи коммуникации (communication),**
- **включения (include),**
- **расширения (extend)**
- **обобщения (generalization).**

СВЯЗЬ КОММУНИКАЦИИ

Связь коммуникации - это связь между вариантом использования и действующим лицом. На языке UML связи коммуникации показывают с помощью однонаправленной ассоциации (линии со стрелкой). Направление стрелки позволяет понять, кто инициирует коммуникацию.



СВЯЗЬ ВКЛЮЧЕНИЯ (ИСПОЛЬЗОВАНИЕ)

Связь включения (include) применяется в тех ситуациях, когда имеется какой-либо фрагмент поведения системы, который повторяется более чем в одном варианте использования, и нет необходимости копировать его описание в каждом из этих вариантов. **С помощью таких связей обычно моделируют многократно используемую функциональность**.

В примере банковской системы варианты использования «Снять деньги со счета» и «Сделать вклад» должны опознать (аутентифицировать) клиента и его PIN-код перед осуществлением самой транзакции. Вместо того, чтобы подробно описывать процесс аутентификации для каждого из них, можно поместить эту функциональность в свой собственный вариант использования под названием «Аутентифицировать клиента».

СВЯЗЬ РАСШИРЕНИЯ

Связь расширения (extend) применяется при описании изменений в нормальном поведении системы. Она позволяет варианту использования только при необходимости применять функциональные возможности другого.

Обе связи предполагают **выделение общих фрагментов поведения из нескольких вариантов использования в единственный вариант**. Этот вариант "используется“ (включает) в других вариантах или "расширяет" несколько других вариантов, но в каждом случае смысл связей с действующими лицами различный.

СХОДСТВА И РАЗЛИЧИЯ МЕЖДУ СВЯЗЯМИ РАСШИРЕНИЕ И ИСПОЛЬЗОВАНИЕ

В случае "расширения" у действующих лиц имеется связь с основным вариантом использования. При этом предполагается, что данное действующее лицо реализует как основной вариант использования, так и все его расширения.

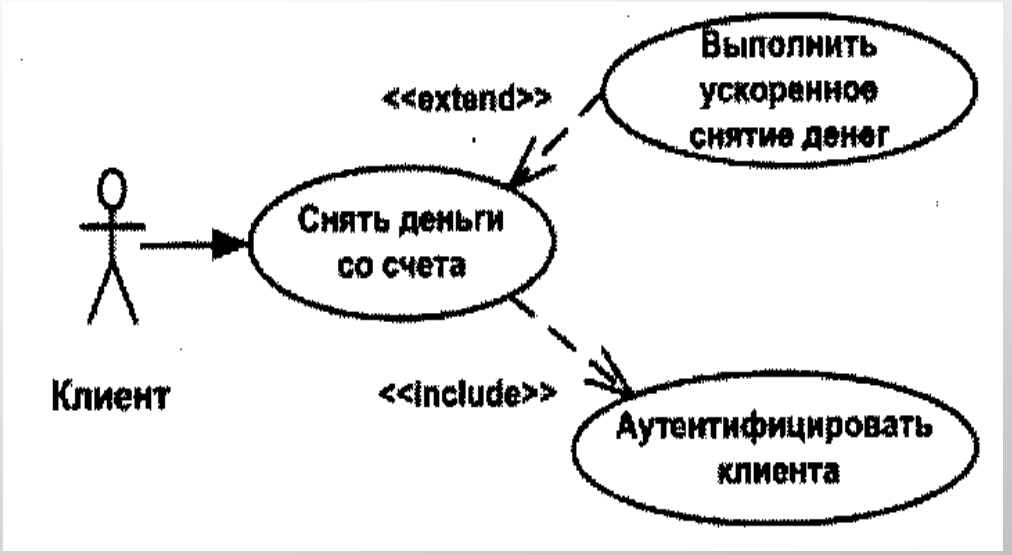
В случае применения связи **"использование" действующие лица, связанные с общим вариантом использования, как правило, отсутствуют**. Даже если имеются исключения, то такое действующее лицо не имеет отношения к реализации других вариантов использования.

ПРАВИЛА ВЫБОРА СВЯЗИ

Выбор применяемой связи определяется следующими правилами:

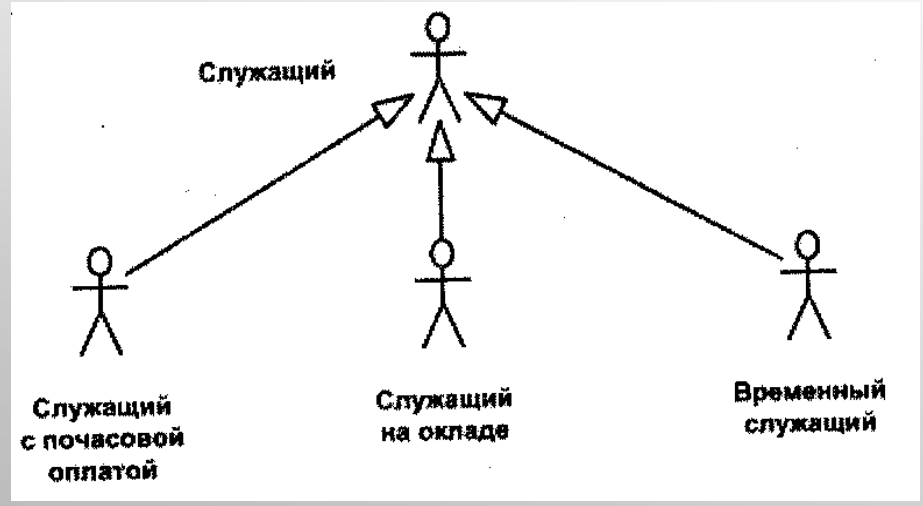
- связь "расширение" следует применять при описании изменений в нормальном поведении системы;
- связь "использование" следует применять для избегания повторов в двух (или более) вариантах использования.

На языке UML **связи включения и расширения показывают в виде зависимостей с соответствующими стереотипами.**



СВЯЗЬ ОБОБЩЕНИЕ

С помощью **связи обобщения** показывают, что у нескольких действующих лиц имеются общие черты. Например, клиенты могут быть двух типов: корпоративные и индивидуальные

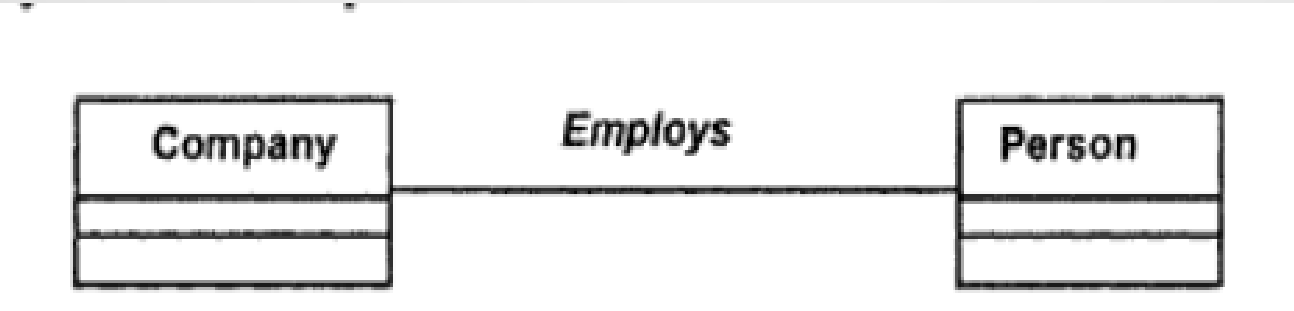


Нет необходимости всегда создавать связи этого типа. В общем случае они нужны, если поведение действующего лица одного типа отличается от поведения другого постольку, поскольку это затрагивает систему. Если оба подтипа применяют одни и те же варианты использования, показывать обобщение действующего лица не требуется

СВЯЗИ МЕЖДУ КЛАССАМИ

- **Ассоциация (association)**
- **Агрегация (aggregation)**
- **Зависимость (dependency)**
- **Обобщение (generalization)**

Ассоциация (association) — это семантическая связь между классами. Ее изображают на диаграмме классов в виде обыкновенной линии. Ассоциация отражает структурные связи между объектами различных классов .



АГРЕГАЦИЯ

Агрегация (aggregation) представляет собой форму ассоциации — более сильный тип связи между целым (составным) объектом и его частями (компонентными объектами).

Сильная форма агрегации является в UML **композицией**. В композиции **составной объект может физически содержать компонентные объекты**. Слабая форма агрегации в UML называется просто **агрегацией.** При этом **составной объект физически не содержит компонентный объект**.

Один компонентный объект может обладать несколькими связями ассоциации или агрегации. Агрегация изображается линией между классами с ромбом на стороне целого объекта. Сплошной ромб представляет композицию.

Рис 2.40. Агрегация

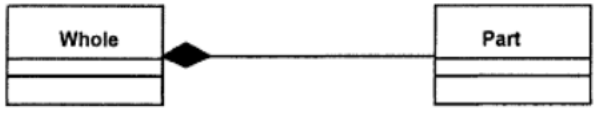


Рис 2.41. Композиция

НАПРАВЛЕНИЕ СВЯЗЕЙ

Хотя связи ассоциации и агрегации двунаправленные по умолчанию, часто накладываются **ограничения на направление навигации** (только в одном направлении).

Если введено ограничение по направлению, то добавляется стрелка на конце связи. Направление ассоциации определяют, изучая сообщения между классами.

Если все сообщения на них отправляются только одним классом и принимаются только другим классом, но не наоборот, между этими классами имеет место **однонаправленная связь.**

Если хотя бы одно сообщение отправляется в обратную сторону, ассоциация должна быть **двунаправленной.**

МОЩНОСТЬ СВЯЗИ

Мощность (multiplicity) показывает, как много объектов участвует в связи. **Мощность — это число объектов одного класса, связанных с одним объектом другого класса**. Понятие мощности связи в объектной модели аналогично понятиям мощности связи в модели «сущность-связь» (с точностью до расположения показателя мощности на диаграмме). Для каждой связи можно обозначить два показателя мощности — по одному на каждом конце связи.

В языке UML приняты следующие нотации для обозначения мощности.

Значения мощности

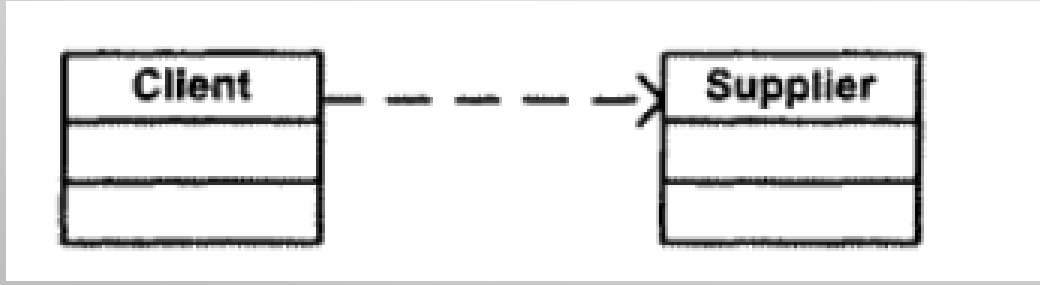
Мощность	Значение
*	Много
0	Нуль
1	Один
0..*	Нуль или больше
1..*	Один или больше
0..1	Нуль или один
1..1	Ровно один

ЗАВИСИМОСТЬ

Зависимость (dependency) — связь между двумя элементами модели, при которой **изменения в спецификации одного элемента могут повлечь за собой изменения в другом элементе**.

Зависимость — слабая форма связи между клиентом и сервером (клиент зависит от сервера и не имеет знаний о сервере). Зависимость изображается пунктирной линией, направленной от клиента к серверу.

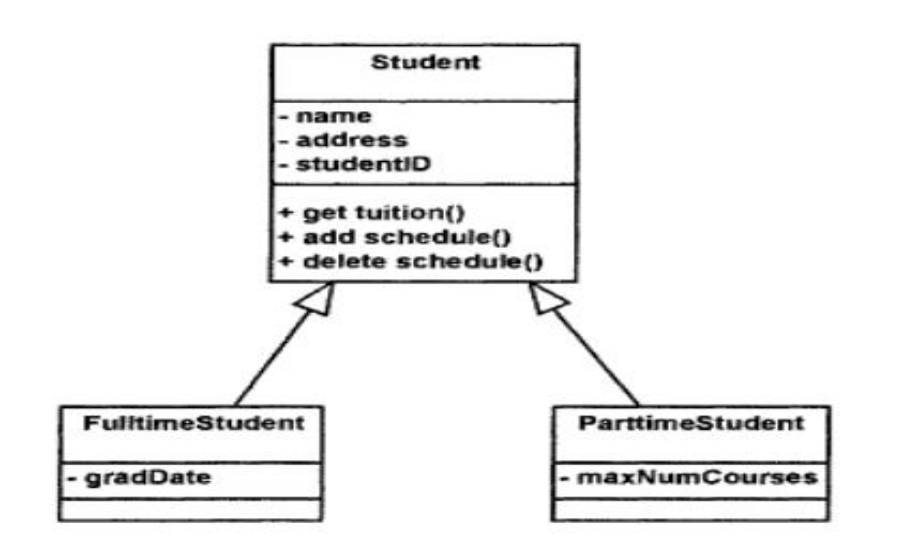
Причины для зависимостей могут быть самыми разными: один класс посылает сообщение другому; один класс включает часть данных другого класса; один класс использует другой в качестве параметра операции.



ОБОБЩЕНИЕ

Обобщение (generalization) — связь «тип-подтип» реализует механизм наследования (inheritance). Она позволяет одному классу наследовать все атрибуты, операции и связи другого. В языке UML связи наследования называют обобщениями и **изображают в виде стрелок от класса-потомка к классу-предку**.

Общие атрибуты, операции и/или связи отображаются на верхнем уровне иерархии. Помимо наследуемых, каждый подкласс имеет свои собственные уникальные атрибуты, операции и связи.



ГЛОССАРИЙ ПРОЕКТА

Глоссарий предназначен для описания терминологии предметной области. Он может быть использован как неформальный словарь данных системы.

Достоинства модели вариантов использования заключаются в том, что она:

- определяет пользователей и границы системы;
- определяет системный интерфейс;
- удобна для общения пользователей с разработчиками;
- используется для написания тестов;
- является основой для написания пользовательской документации;
- хорошо вписывается в любые методы проектирования (как объектно-ориентированные, так и структурные).

ДИАГРАММЫ ВЗАИМОДЕЙСТВИЯ

Диаграммы взаимодействия (interaction diagrams) описывают поведение взаимодействующих групп объектов.

Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме **отображаются ряд объектов и те сообщения, которыми они обмениваются между собой**.

Существуют два вида диаграмм взаимодействия:

диаграммы последовательности и кооперативные диаграммы.

Диаграммы последовательности упорядочены по времени, а

кооперативные диаграммы концентрируют внимание на связях между объектами.

СООБЩЕНИЕ

Сообщение (message) — средство, с помощью которого объект-отправитель запрашивает у объекта-получателя выполнение одной из его операций.

Информационное (informative) сообщение — сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

Сообщение-запрос (interrogative) — сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

Императивное (imperative) сообщение — сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

ДИАГРАММЫ ПОСЛЕДОВАТЕЛЬНОСТИ

Диаграммы последовательности отражают временную последовательность событий, происходящих в рамках варианта использования.

Один из способов **первоначального обнаружения некоторых объектов** - это **изучение имен существительных в потоке событий**. Поток событий для варианта использования «Снять деньги со счета» говорит о человеке, снимающем некоторую сумму денег со счета с помощью банкомата.

Не все объекты, показанные на диаграмме, явно присутствуют в потоке событий. Там, например, может не быть форм для заполнения, но их необходимо показать на диаграмме, чтобы позволить действующему лицу ввести новую информацию в систему или просмотреть ее. В потоке событий, скорее всего, также не будет и управляющих объектов (control objects). Эти объекты управляют последовательностью событий в варианте использования.

ДИАГРАММЫ ПОСЛЕДОВАТЕЛЬНОСТИ

На диаграмме последовательности **объект** изображается в виде **прямоугольника на вершине пунктирной вертикальной линии**. Эта **вертикальная линия** называется **линией жизни (lifeline) объекта**. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

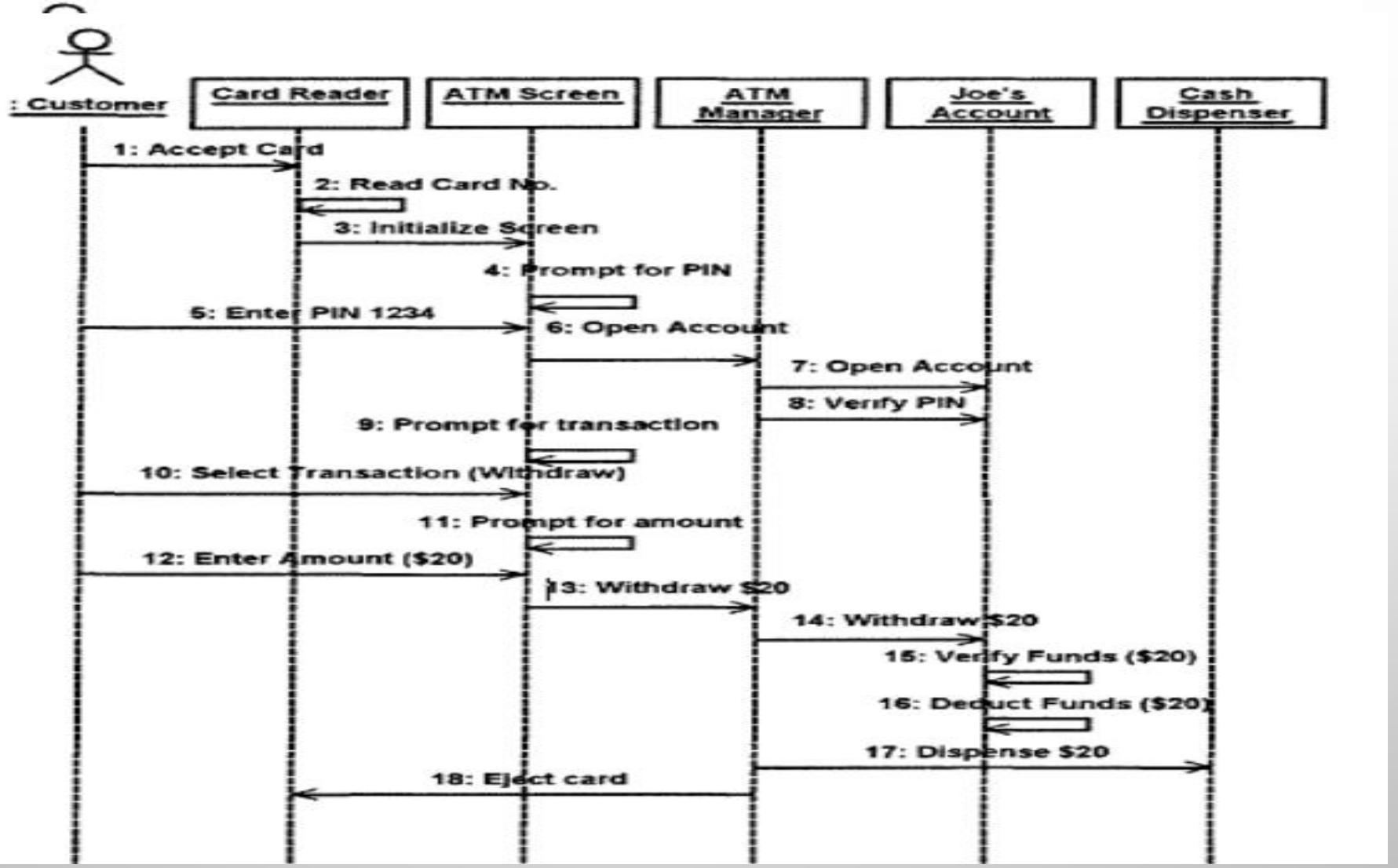
Каждое **сообщение** представляется в виде **стрелки между линиями жизни двух объектов**. Сообщения появляются в том порядке, как они показаны на странице сверху вниз. Каждое **сообщение помечается, как минимум, именем сообщения.** При желании можно добавить также аргументы и некоторую управляющую информацию. Кроме того, можно показать самоделегирование (self-delegation) — сообщение, которое объект посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни.

ДИАГРАММЫ ПОСЛЕДОВАТЕЛЬНОСТИ

Например, вариант использования **«Снять деньги со счета»** предусматривает несколько возможных потоков событий: снятие денег, попытка снять деньги, не имея их достаточного количества на счете, попытка снять деньги по неправильному PIN-коду и некоторых других.

Нормальный сценарий (основной поток событий) снятия некоторой суммы денег со счета показан на рисунке. Все действующие лица показаны в верхней части диаграммы. В приведенном примере изображено действующее лицо Клиент (Customer). Объекты, требуемые системе для выполнения варианта использования «Снять деньги со счета», также представлены в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

ПРИМЕР

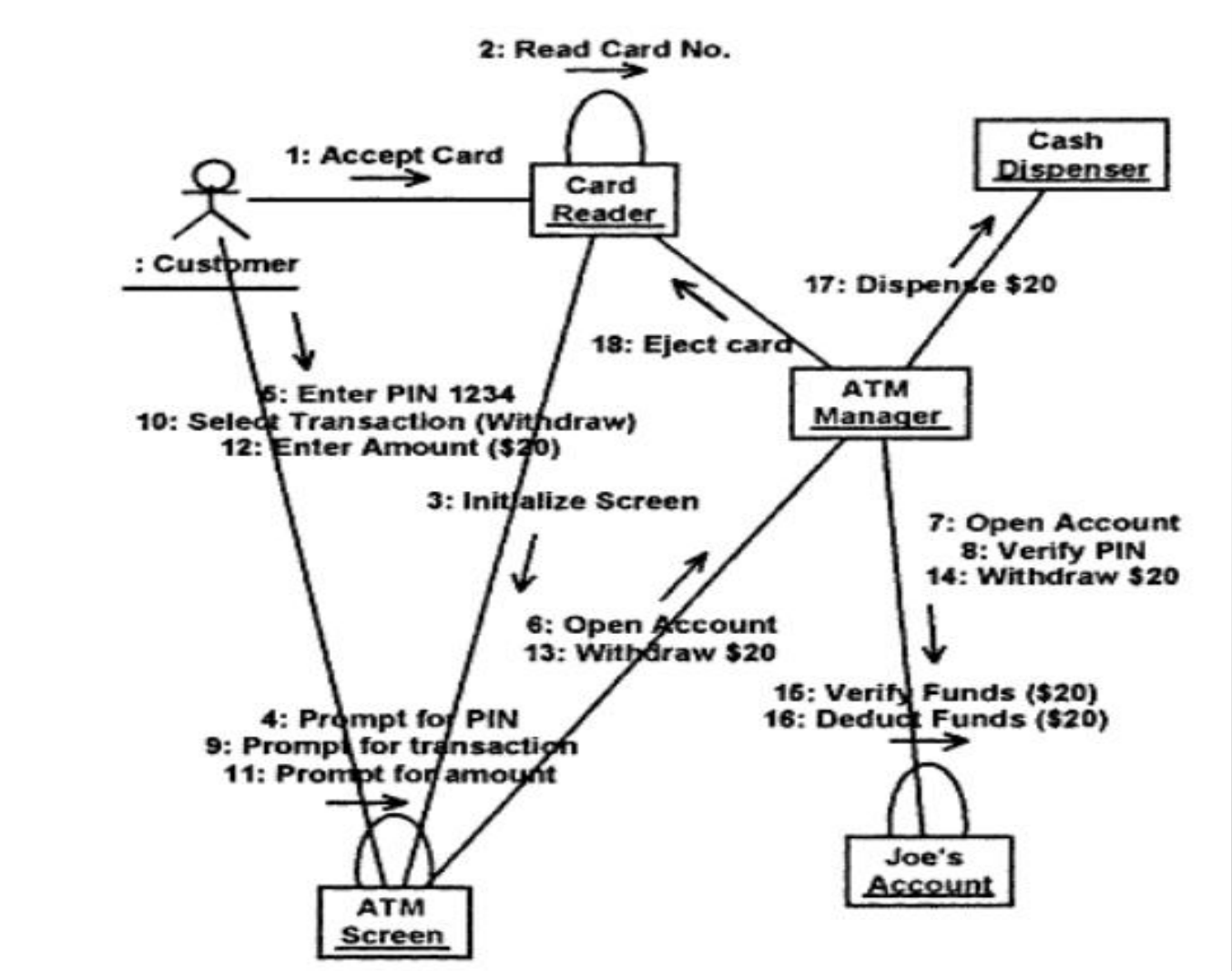

: Customer
Card Reader
ATM Screen
ATM Manager
Joe's Account
Cash Dispenser
1: Accept Card
2: Read Card No.
3: Initialize Screen
4: Prompt for PIN
5: Enter PIN 1234
6: Open Account
7: Open Account
8: Verify PIN
9: Prompt for transaction
10: Select Transaction (Withdraw)
11: Prompt for amount
12: Enter Amount ($20)
13: Withdraw $20
14: Withdraw $20
15: Verify Funds ($20)
16: Deduct Funds ($20)
17: Dispense $20
18: Eject card

КООПЕРАТИВНЫЕ ДИАГРАММЫ

Кооперативные диаграммы тоже отображают поток событий варианта использования. Но в отличие от диаграммы последовательности**,** которые упорядочены по времени, **концентрируют внимание на связях между объектами.**

На рисунке приведена кооперативная диаграмма, описывающая, как клиент снимает деньги со счета. На ней представлена та же информация, которая была на диаграмме последовательности. Но кооперативная диаграммы по-другому описывает поток событий. Из нее легче понять связи между объектами, однако труднее уяснить последовательность событий. По этой причине часто для какого-либо сценария создают диаграммы обоих типов. Хотя они служат одной и той же цели и содержат одну и ту же информацию, но представляют ее с различных точек зрения.

КООПЕРАТИВНЫЕ ДИАГРАММЫ



На кооперативной диаграмме так же, как и на диаграмме последовательности, **стрелки обозначают сообщения**, обмен которыми осуществляется в рамках данного варианта использования. Их временная последовательность, однако, указывается путем **нумерации сообщений.**

ДИАГРАММА СОСТОЯНИЙ

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий. Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой.

Диаграммы состояний не надо создавать для каждого класса, они применяются только в сложных случаях. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, для него может потребоваться такая диаграмма.

На диаграмме имеются два специальных состояния — начальное (start) и конечное (stop). Начальное состояние может быть только одно, а конечных состояний может быть столько, сколько нужно, или их может не быть вообще.

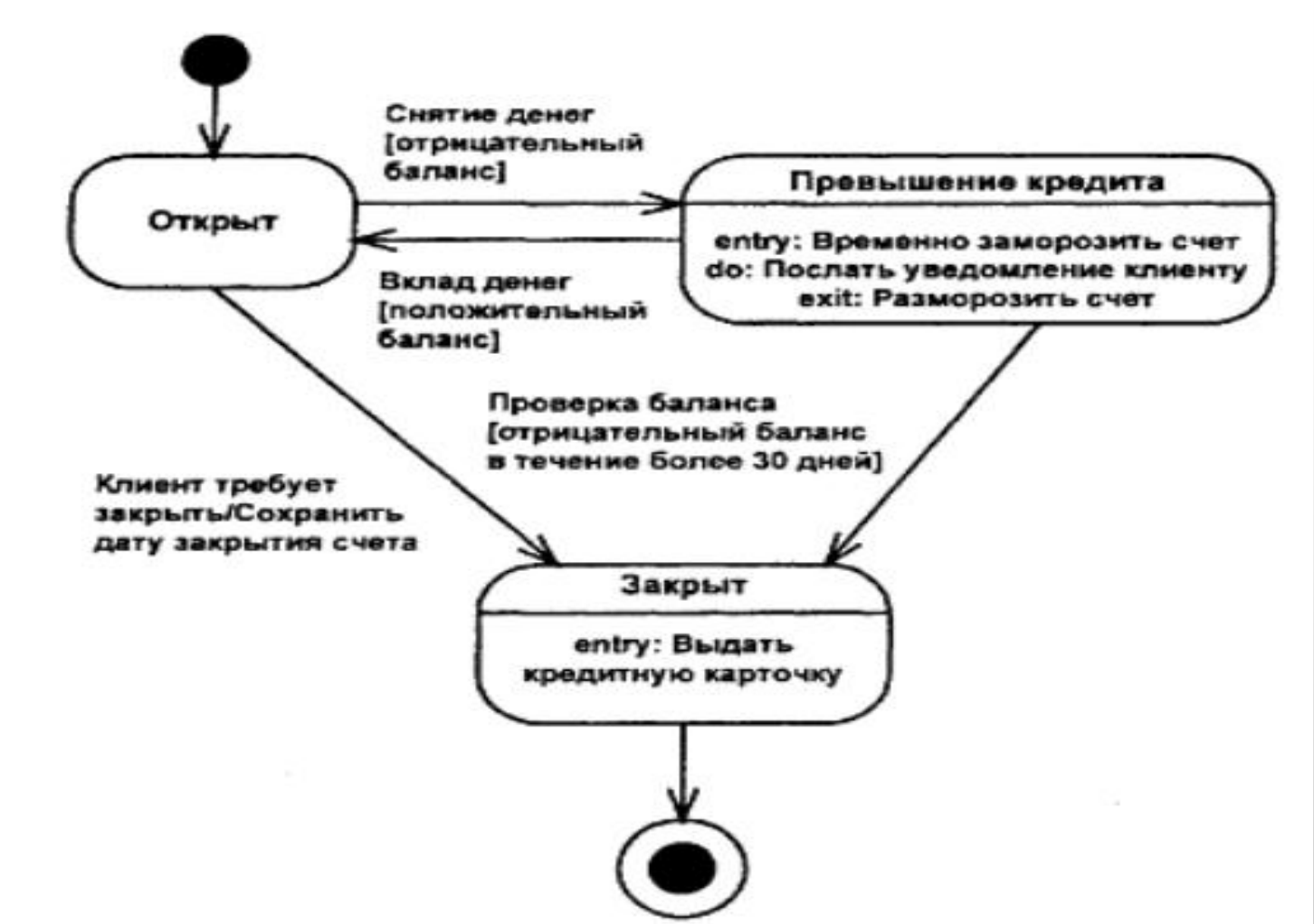
ДИАГРАММА СОСТОЯНИЙ

На рисунке приведен пример диаграммы состояний для банковского счета. Из данной диаграммы видно, в каких состояниях может существовать счет. Можно также видеть процесс перехода счета из одного состояния в другое. Например, если клиент требует закрыть открытый счет, он переходит в состояние «закрыт». **Требование клиента называется событием (event)**, именно такие **события и вызывают переход из одного состояния в другое**.

Если клиент снимает деньги с открытого счета, он может перейти в состояние «Превышение кредита». Это происходит, только если баланс по этому счету меньше нуля, что отражено условием [отрицательный баланс] на диаграмме. При превышении кредита клиенту посылается соответствующее сообщение.

Заключенное **в квадратных скобках ограничивающее условие (guard condition) определяет, когда может или не может произойти переход из одного состояния в другое**.

ДИАГРАММА СОСТОЯНИЙ ДЛЯ КЛАССА ACCOUNT



Начальное состояние выделено черной точкой, оно соответствует состоянию объекта, когда он только что был создан.

Конечное состояние обозначается черной точкой в белом кружке, оно соответствует состоянию объекта непосредственно перед его уничтожением.

ДИАГРАММА СОСТОЯНИЙ

Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы. Они называются **действиями (actions).**

С состоянием можно связывать данные пяти типов:

- деятельность,
- входное действие,
- выходное действие,
- событие и
- история состояния.

Рассмотрим каждый из них в контексте диаграммы состояний для класса Account банковской системы

ДЕЯТЕЛЬНОСТЬ

Деятельность (activity) - это поведение, реализуемое объектом, пока он находится в данном состоянии.

Например, когда счет находится в состоянии «Закрыт», происходит возврат кредитной карточки пользователю. Деятельность — это прерываемое поведение. Оно может выполняться до своего завершения, пока объект находится в данном состоянии, или может быть прервано переходом объекта в другое состояние.

Деятельность изображают внутри самого состояния, ей должно предшествовать слово do (выполнять) и двоеточие.

ВХОДНОЕ ДЕЙСТВИЕ

Входное действие (entry action) — это поведение, которое выполняется, когда объект переходит в данное состояние.

Когда счет в банке переходит в состояние «Превышение кредита», выполняется действие «Временно заморозить счет» независимо оттого, откуда объект перешел в это состояние. Таким образом, данное действие осуществляется не после того, как объект перешел в это состояние, а, скорее, как часть этого перехода. В отличие от деятельности входное действие рассматривается как непрерываемое.

Входное действие также показывают внутри состояния, ему предшествует слово entry (вход) и двоеточие.

ВЫХОДНОЕ ДЕЙСТВИЕ

Выходное действие (exit action) подобно входному, однако оно осуществляется как составная часть процесса выхода из данного состояния.

Так, при выходе объекта Account из состояния «Превышение кредита» действие «Разморозить счет» выполняется независимо оттого, куда он переходит. Оно является частью процесса такого перехода. Как и входное, выходное действие является непрерываемым. Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

Действие может быть не только входным или выходным, но и частью перехода. Например, при переходе счета из открытого в закрытое состояние выполняется действие «Сохранить дату закрытия счета».

Действие изображают вдоль линии перехода после имени события, ему предшествует косая черта.

ПЕРЕХОД

Переходом (transition) называется перемещение объекта из одного состояния в другое. На диаграмме все переходы изображают в виде стрелки, начинающейся на первоначальном состоянии и заканчивающейся на последующем.

Переходы могут быть рефлексивными. Объект может перейти в то же состояние, в котором он в настоящий момент находится. Рефлексивные переходы изображают в виде стрелки, начинающейся и завершающейся на одном и том же состоянии. У перехода существует несколько спецификаций, основными из которых являются события, ограждающие условия и действия.

СОБЫТИЕ (EVENT)

Событие (event) вызывает переход из одного состояния в другое. Событие «Клиент требует закрыть» вызывает переход счета из открытого в закрытое состояние. Событие размещают на диаграмме вдоль линии перехода.

На диаграмме для отображения события можно использовать как имя операции, так и обычную фразу, как в примере. Если нужно использовать операции, то событие «Клиент требует закрыть » можно было бы назвать RequestClosure().

Большинство переходов должны иметь события, так как именно они заставляют переход осуществиться. Тем не менее, бывают и автоматические переходы, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, позволяющей осуществиться входным действиям, деятельности и выходным действиям.

ОГРАНИЧИВАЮЩИЕ УСЛОВИЯ

Ограничивающие условия (guard conditions) определяют, когда переход может, а когда не может осуществиться.

В примере событие «Вклад денег» переведет счет из состояния «Превышение кредита» в состояние «Открыт» только если баланс будет больше нуля. В противном случае переход не осуществится.

Ограничивающие условия изображают на диаграмме вдоль линии перехода после имени события, заключая их в квадратные скобки. Ограничивающие условия задавать необязательно. Однако, если существует несколько автоматических переходов из состояния, необходимо определить для них взаимно исключающие ограждающие условия, чтобы понять, какой путь перехода будет автоматически выбран.

ДИАГРАММА КЛАССОВ

Диаграмма классов определяет **типы классов системы** и различного рода **статические связи, которые существуют между ними**. На диаграммах классов изображаются также **атрибуты классов, операции классов и ограничения, которые накладываются на связи между классам**и. Диаграмма классов для варианта использования «Снять деньги со счета» показана на рисунке.

На этой диаграмме присутствуют четыре класса: Card Reader (устройство для чтения карточек). Account (счет), ATM Screen (экран ATM) и Cash Dispenser (кассовый аппарат). Связывающие классы линии отражают взаимодействие между классами. Так, класс Account связан с классом ATM Screen, потому что они непосредственно сообщаются и взаимодействуют друг с другом. Класс Card Reader не связан с классом Cash Dispenser, поскольку они не сообщаются друг с другом непосредственно

ДИАГРАММА КЛАССОВ

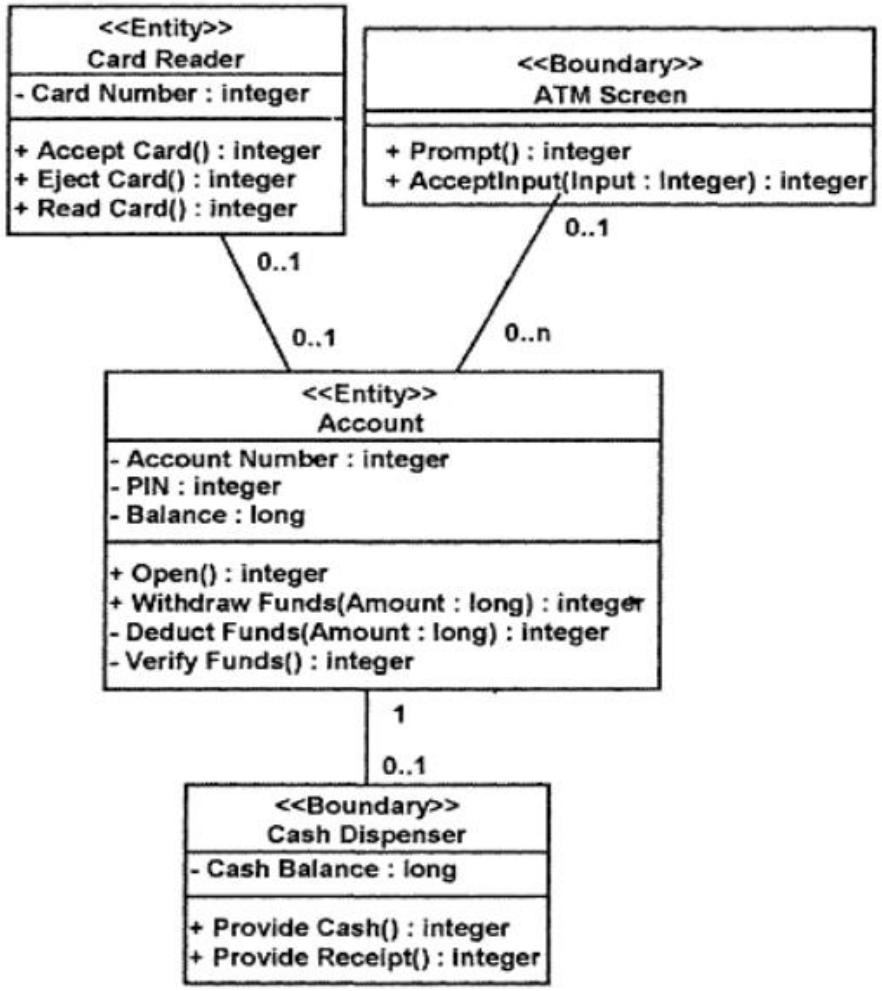


Рис. 2.52. Диаграмма классов для варианта использования «Снять деньги со счета»

Типы классов:
Граничные
Управляющие
Сущности

Boundary (граница).
Entity (сущность
Control (управление)

МЕХАНИЗМЫ РАСШИРЕНИЯ UML

Механизмы расширения UML предназначены для того, **чтобы разработчики могли адаптировать язык моделирования к своим конкретным нуждам, не меняя при этом его метамодель.** Наличие механизмов расширения принципиально отличает UML от таких средств моделирования, как IDEFO, IDEF1X, IDEF3, DFD и ERM.

UML допускает произвольную интерпретацию семантики элементов моделей (в основном за счет стереотипов) и является слабо типизированным языком. К его механизмам расширения относятся:

- **стереотипы;**
- **тегированные (именованные) значения;**
- **ограничения**.

СТЕРЕОТИПЫ

Стереотип — это новый тип элемента модели, который определяется на основе уже существующего элемента. Стереотипы расширяют нотацию модели, могут применяться к любым элементам модели и представляются в виде текстовой метки или пиктограммы (иконки). **Стереотипы классов — это механизм, позволяющий разделять классы на категории**.

Например, основными стереотипами, используемыми в процессе анализа системы, **являются: Boundary (граница). Entity (сущность) и Control (управление).**

Граничными классами (boundary classes) называются такие классы, которые расположены на границе системы и всей окружающей среды. Они включают все формы, отчеты, интерфейсы с аппаратурой (такой, как принтеры или сканеры) и интерфейсы с другими системами.

СТЕРЕОТИПЫ

Классы-сущности (entity classes) отражают основные понятия (абстракции) предметной области и, как правило, содержат хранимую информацию. Обычно для каждого класса-сущности создают таблицу в базе данных.

Управляющие классы (control classes) отвечают за координацию действий других классов. Обычно у каждого варианта использования имеется один управляющий класс, контролирующий потоки событий этого варианта использования. Управляющий класс отвечает за координацию, но сам не несет в себе никакой функциональности — остальные классы не посылают ему большого количества сообщений. Вместо этого он сам посылает множество сообщений. Управляющий класс просто делегирует ответственность другим классам, по этой причине его часто называют классом- менеджером

СТЕРЕОТИПЫ

В системе могут быть и другие управляющие классы, общие для нескольких вариантов использования. Например, класс SecurityManager (менеджер безопасности) может отвечать за контроль событий, связанных с безопасностью.

Класс TransactionManager (менеджер транзакций) занимается координацией сообщений, относящихся к транзакциям с базой данных. Могут быть и другие менеджеры для работы с другими элементами функционирования системы, такими, как разделение ресурсов, распределенная обработка данных или обработка ошибок. Помимо упомянутых стереотипов, разработчики ПО могут создавать собственные наборы стереотипов, формируя тем самым специализированные подмножества UML . Такие подмножества (наборы стереотипов) в стандарте языка UML носят название **профилей** языка.

ИМЕНОВАННОЕ ЗНАЧЕНИЕ. ОГРАНИЧЕНИЕ

Именованное значение — это пара строк «тег = значение^>, или «имя = содержимое», в которых хранится дополнительная информация о каком-либо элементе системы, например, время создания, статус разработки или тестирования, время окончания работы над ним и т.п.

Ограничение — это семантическое ограничение, имеющее вид текстового выражения на естественном или формальном языке (OCL — Object Constraint Language), которое невозможно выразить с помощью нотации UML. Одних графических средств, таких как диаграмма классов или диаграмма состояний, недостаточно для разработки точных и непротиворечивых моделей сложных систем. Существует необходимость задания дополнительных Ограничений, которым должны удовлетворять различные компоненты или объекты модели.

ПАКЕТЫ

Пакет — общий механизм для организации элементов модели в группы. Применяется для группировки классов, обладающих некоторой общностью.

Пакет может включать другие элементы.

Каждый элемент модели может входить только в один пакет.

Пакет является средством организации модели в процессе разработки, повышения ее управляемости и читаемости, а также единицей управления конфигурацией.

ПАКЕТЫ

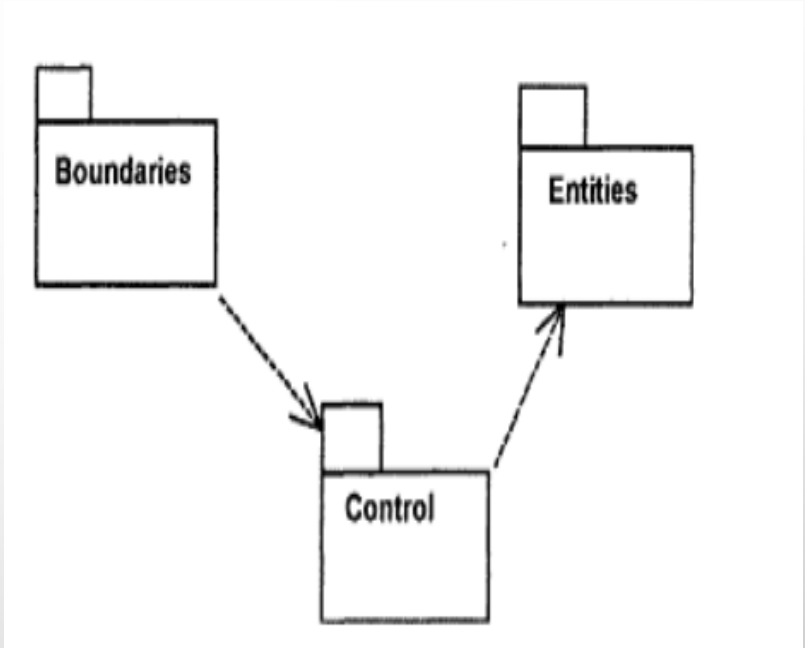
Существуют несколько подходов к группировке классов.

Во первых, **можно группировать их по стереотипу** (типу класса).Например, один пакет содержит классы-сущности предметной области, другой — классы пользовательского интерфейса и т.д. Этот подход может быть полезен с точки зрения размещения системы в среде реализации.

Другой подход заключается в **объединении классов по их функциональности**. Например, в пакете Security (безопасность) содержатся все классы, отвечающие за безопасность приложения. В таком случае другие пакеты могут называться Employee Maintenance (Работа с сотрудниками). Reporting (Подготовка отчетов) и Error Handling (Обработка ошибок). Преимущество этого подхода заключается в возможности повторного использования.

ДИАГРАММА ПАКЕТОВ

Если между любыми двумя классами, находящимися в разных пакетах, существует некоторая зависимость, то имеет место зависимость и между этими двумя пакетами. **Диаграмма пакетов** представляет собой диаграмму, **содержащую пакеты классов и зависимости между ними**. Строго говоря, пакеты и зависимости являются элементами диаграммы классов, т.е. диаграмма пакетов — это форма диаграммы классов. Диаграммы пакетов можно считать основным средством управления общей структурой системы



ПОДСИСТЕМА

Пакеты также используются для представления подсистем (модулей) системы. **Подсистема — это комбинация пакета (поскольку она включает некоторое множество классов) и класса (поскольку она обладает поведением**, т.е. реализует набор операций, которые определены в ее интерфейсах). Связь между подсистемой и интерфейсом называется *связью реализации*. Подсистема используется для представления компонента в процессе проектирования.

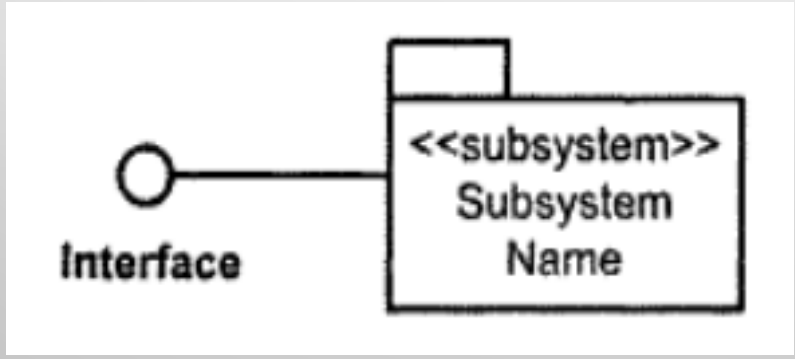


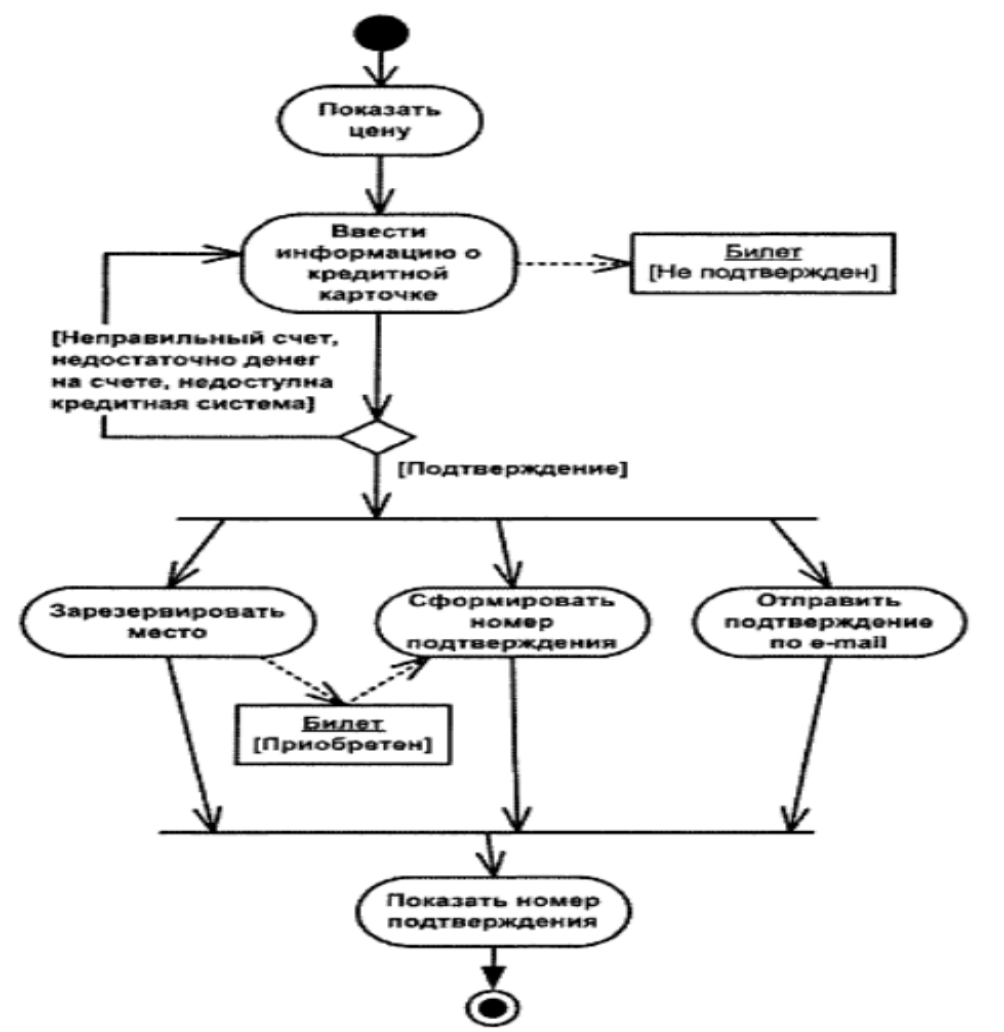
ДИАГРАММА ДЕЯТЕЛЬНОСТИ

Диаграммы деятельности особенно полезны в описании поведения, включающего большое количество параллельных процессов, например, при параллельном программировании. Т.к. можно графически изобразить все ветви и определить, когда их необходимо синхронизировать.

Их можно **применять для описания потоков событий в вариантах использования**. С помощью текстового описания можно подробно рассказать о потоке событий, но в сложных и запутанных потоках с множеством альтернативных ветвей будет трудно понять логику событий.

Диаграммы деятельности предоставляют ту же информацию, что и текстовое описание потока событий, но в наглядной графической форме. На рисунке приведена диаграмма деятельности для потока событий, связанного с системой бронирования авиабилетов..

ДИАГРАММА ДЕЯТЕЛЬНОСТИ ДЛЯ ПОТОКА СОБЫТИЙ, СИСТЕМЫ БРОНИРОВАНИЯ АВИАБИЛЕТОВ



Основным элементом диаграммы является **деятельность (activity)**, это может быть некоторая задача, которую необходимо выполнить вручную или автоматизированным способом, или операция класса. Деятельность изображается в виде **закругленного прямоугольника с текстовым описанием**.

Любая диаграмма деятельности должна иметь **начальную точку, определяющую начало потока событий**. Конечная точка необязательна. На диаграмме может быть **несколько конечных точек, но только одна начальная**.

НОТАЦИЯ ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ

На диаграмме могут присутствовать **объекты и потоки объектов (objectflow**). **Объект может использоваться или изменяться в одной из деятельностей**. Показ объектов и их состояний (в дополнение к диаграммам состояний) помогает понять, когда и как происходит смена состояний объекта.

Объекты связаны с деятельностями через потоки объектов. Поток объектов отмечается пунктирной стрелкой от деятельности к изменяемому объекту или от объекта к деятельности, использующей объект.

На рисунке после ввода пользователем информации о кредитной карточке билет переходит в состояние «не подтвержден». Когда завершится процесс обработки кредитной карточки и будет подтвержден перевод денег, возникает деятельность «зарезервировать место», переводящая билет в состояние «приобретен». и затем он используется в деятельности «формирование номера подтверждения».

НОТАЦИЯ ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ

Переход (стрелка) показывает, **как поток управления переходит от одной деятельности к другой.** Если для перехода определено событие, то переход выполняется только после наступления такого события. Ограничивающие условия определяют, когда переход может, а когда не может осуществиться.

Если необходимо показать, что две или более ветвей потока выполняются параллельно, используются **линейки синхронизации**. В данном примере параллельно выполняются резервирование места, формирование номера подтверждения и отправка почтового сообщения, а после завершения всех трех процессов пользователю выводится номер подтверждения.

ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ

Любая деятельность может быть подвергнута дальнейшей декомпозиции. Описание декомпозированной деятельности может быть представлено в виде другой диаграммы деятельности.

Подобно большинству других средств, моделирующих поведение, диаграммы деятельности отражают только вполне определенные его аспекты, поэтому их лучше всего использовать в сочетании с другими средствами.

ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ

Диаграммы деятельности предпочтительнее использовать для:

• **анализа потоков событий в конкретном варианте использования**. Здесь нас не интересует связь между действиями и объектами, а нужно только понять, какие действия должны иметь место и каковы зависимости в поведении системы. Связывание действий и объектов выполняется позднее с помощью диаграмм взаимодействия;

• **анализа потоков событий в различных вариантах использования**. Когда варианты использования взаимодействуют друг с другом, на диаграмме деятельности удобно представить и проанализировать все их потоки событий (в этом случае диаграмма с помощью вертикальных пунктирных линий разделяется **на зоны — так называемые плавательные дорожки(swimlanes)**. В каждой зоне изображаются потоки событий одного из вариантов использования, а связи между разными потоками — в виде переходов или потоков объектов).

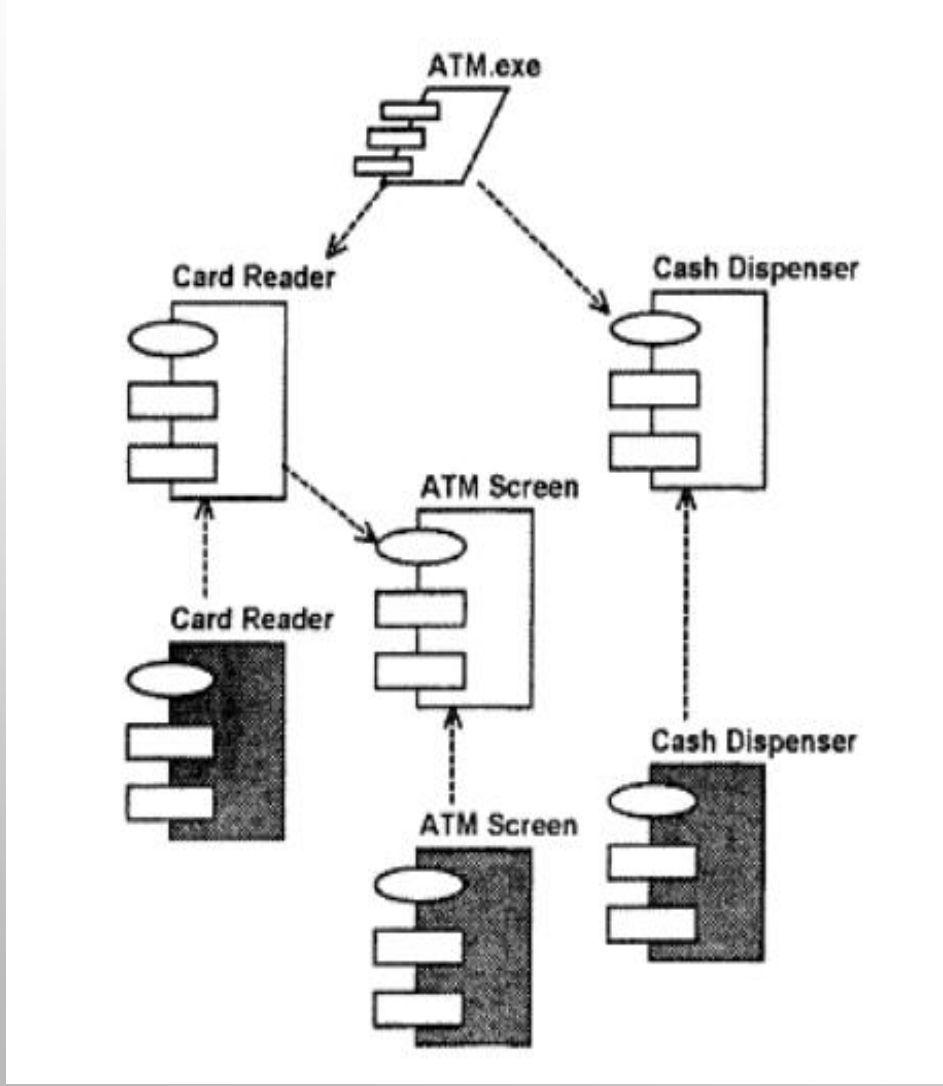
ДИАГРАММЫ КОМПОНЕНТОВ

Диаграммы компонентов моделируют физический уровень системы. На них изображаются компоненты ПО и связи между ними. На такой диаграмме обычно выделяют два типа компонентов: **исполняемые компоненты и библиотеки кода.**

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. Между отдельными компонентами изображают зависимости, соответствующие зависимостям на этапе компиляции или выполнения программы.

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию и сборку системы. Они нужны там, где начинается генерация кода.

ДИАГРАММА КОМПОНЕНТОВ КЛИЕНТСКОЙ ЧАСТИ БАНКОВСКОЙ СИСТЕМЫ



Система разрабатывается на языке C++. У каждого класса имеется свой собственный заголовочный файл (файл с расширением .h) и файл тела класса (файл с расширением .cpp). Например, класс ATM Screen преобразуется в компоненты ATM Screen: тело и заголовок класса.

Выделенный темным компонент называется **спецификацией пакета (package specification)** и соответствует **файлу тела класса ATM Screen**. **Невыделенный компонент** также называется спецификацией пакета, но соответствует **заголовочному файлу класса.**

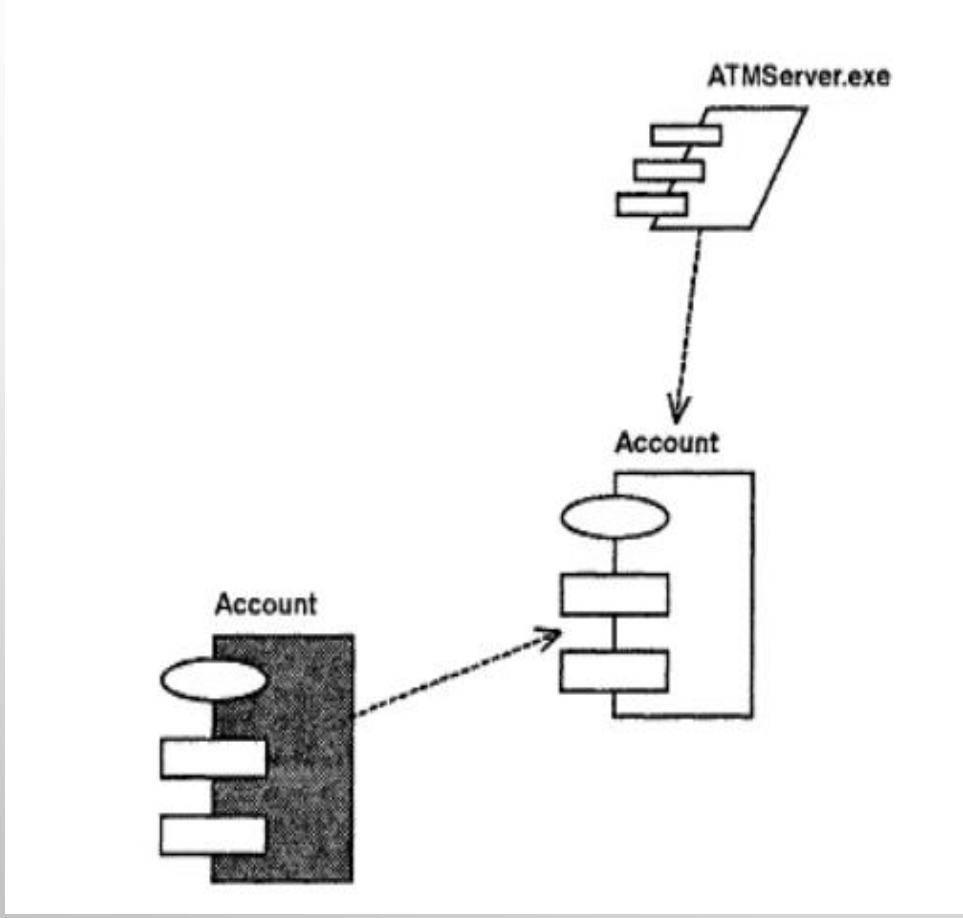
ДИАГРАММА КОМПОНЕНТОВ КЛИЕНТСКОЙ ЧАСТИ БАНКОВСКОЙ СИСТЕМЫ

Компонент ATM.exe называется **спецификацией задачи** и моделирует поток управления (**thread of processing)** — исполняемую программу.

Компоненты соединены зависимостями. Например, класс Card Reader зависит от класса ATM Screen. Это означает, что, для того чтобы класс Card Reader мог быть скомпилирован, класс ATM Screen должен уже существовать. После компиляции всех классов может быть создан исполняемый файл ATMClient.exe.

Банковская система содержит два потока управления и, таким образом, получаются два исполняемых файла. Один из них — это клиентская часть системы, она содержит компоненты Cash Dispenser, Card Reader и ATM Screen. Второй файл — это сервер.

ДИАГРАММА КОМПОНЕНТОВ ДЛЯ СЕРВЕРА



Сервер включает в себя компонент Account.

Как видно из примера, у системы может быть несколько диаграмм компонентов в зависимости от числа подсистем или исполняемых файлов. Каждая подсистема является пакетом компонентов. В общем случае пакеты-это совокупность компонентов.

ДИАГРАММЫ РАЗМЕЩЕНИЯ

На **диаграмме размещения показывается связь между программными и аппаратными компонентами системы**. Она является хорошим средством, чтобы показать размещение объектов и компонентов в распределенной системе.

Каждый **узел** на диаграмме размещения представляет собой некоторый **тип вычислительного устройства** — в большинстве случаев, часть аппаратуры. Эта аппаратура может быть простым устройством или датчиком, а может быть и мэйнфреймом. Диаграмма размещения показывает **физическое расположение сети и местонахождение в ней различных компонентов.**

Диаграмма размещения используется менеджером проекта, пользователями, архитектором системы и эксплуатационным персоналом, чтобы понять физическое размещение системы и расположение ее отдельных подсистем.

ДИАГРАММА РАЗМЕЩЕНИЯ

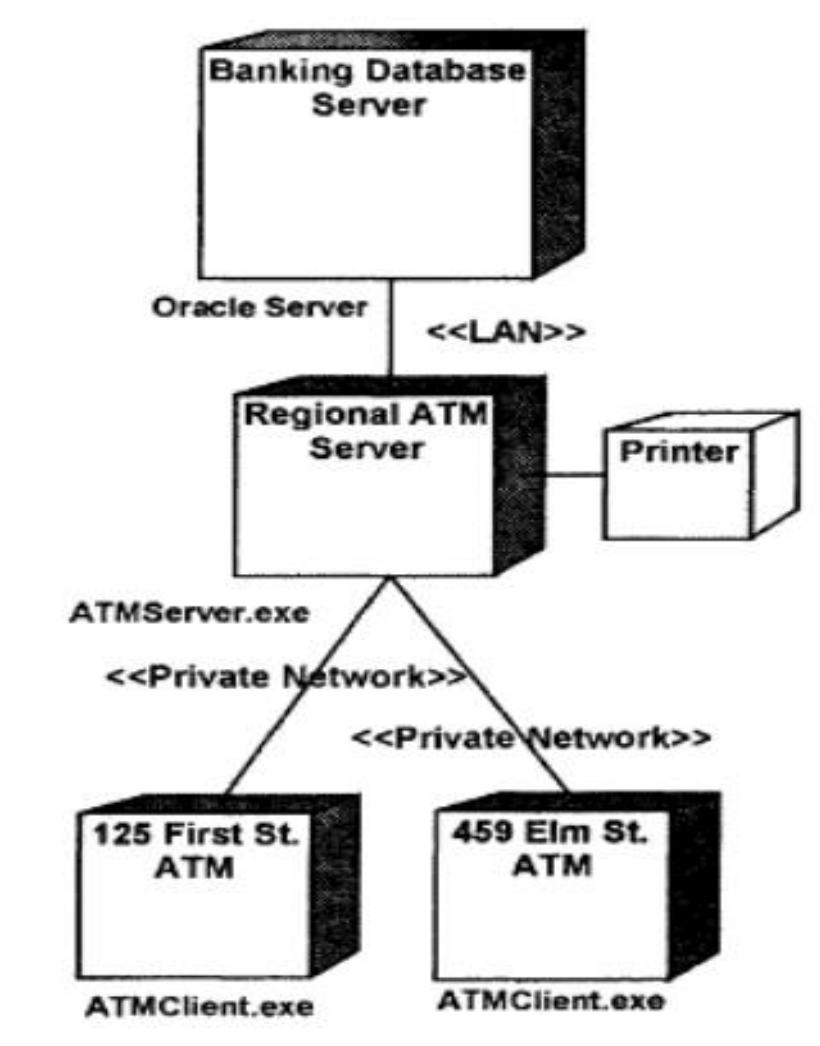


Диаграмма размещения для банковской системы, которая состоит из большого количества подсистем, выполняемых на отдельных физических устройствах или узлах, показана на рисунке.

Основные элементы диаграммы размещения:

- **узел (node)** — вычислительный ресурс — процессор или другое устройство (дисковая память, контроллеры различных устройств и тд.). Для узла можно задать выполняющиеся на нем процессы;
- **соединение (connection)** — канал взаимодействия узлов (сеть).

АНАЛИЗ ДИАГРАММЫ

Из диаграммы размещения для банковской системы можно узнать о физическом размещении системы.

Клиентские программы будут работать в нескольких местах на различных сайтах. Через закрытые сети будет осуществляться их сообщение с региональным сервером системы. На нем будет работать ПО сервера.

В свою очередь, посредством локальной сети региональный сервер будет сообщаться с сервером банковской базы данных, работающим под управлением Oracle.

Принтер соединен с региональным сервером.

ПРИМЕР 1

ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К СИСТЕМЕ

ГЛОССАРИЙ ПРОЕКТА

ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

ПОТОК СОБЫТИЙ

ДИАГРАММА ДЕЯТЕЛЬНОСТИ

ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К СИСТЕМЕ

Для учебного заведения нужно разработать информационную систему для контроля и учета успеваемости студентов.

Определим основные функциональные требования к системе.

Система должна позволять выполнять следующие действия:

Администратор может добавлять нового пользователя студента или преподавателя.

Администратор может добавлять в журнал новые учебные дисциплины;.

При регистрации студента ему присваивается номер учебной группы.

Администратор закрепляет за преподавателем учебную группу и учебную дисциплину. Таким образом создаются страницы электронного журнала.

Преподаватель может выставлять оценки по предмету.

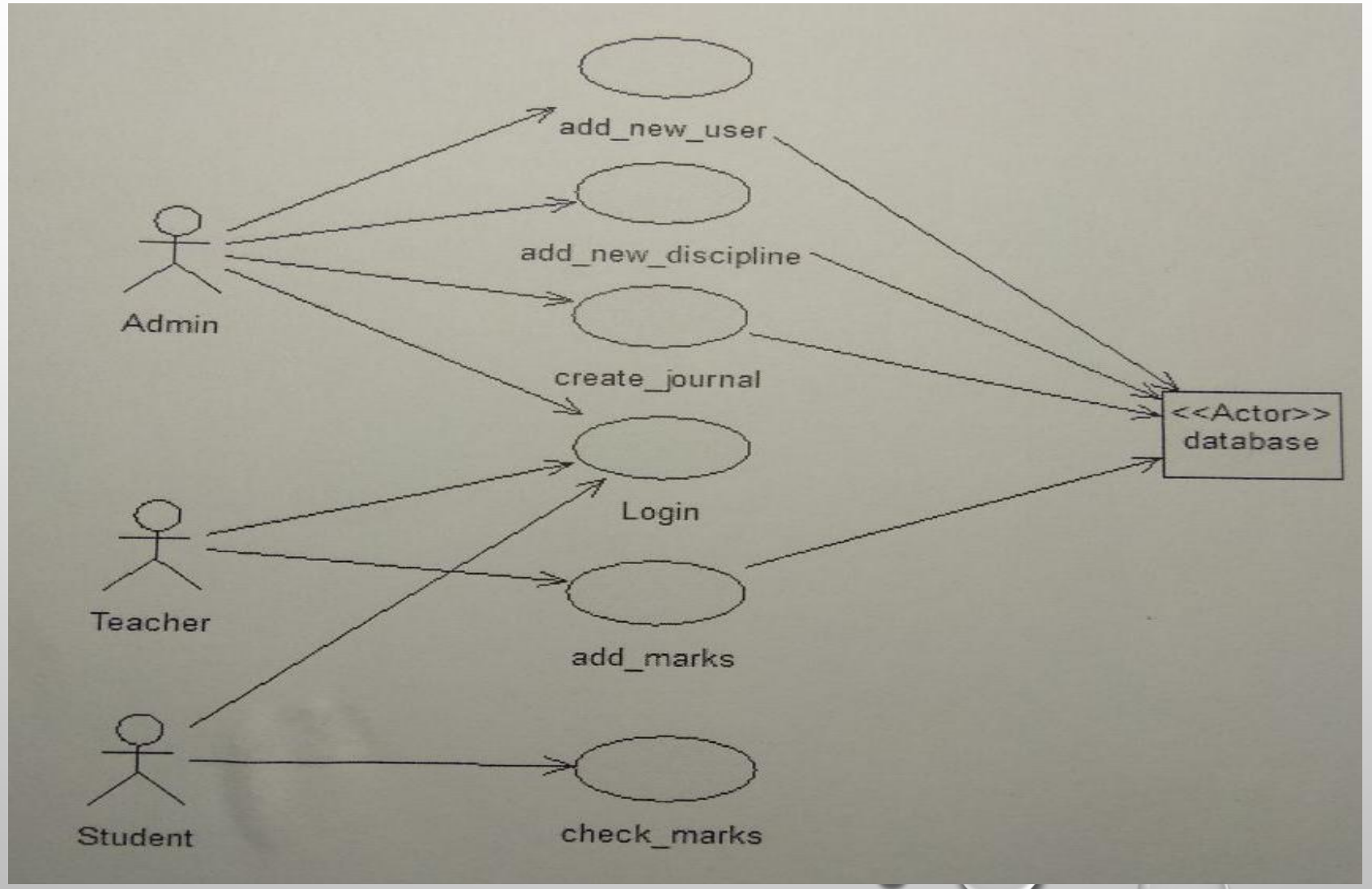
Студент может просматривать свои оценки.

ГЛОССАРИЙ ПРОЕКТА /**PROJECT GLOSSARY**/

Термин	**Значение**
Администратор	Добавляет и удаляет пользователей, учебные дисциплины. Создает страницы электронного журнала
Пользователь	Студент или преподаватель
База данных	Хранит данные пользователя, учебные дисциплины и оценки
LoginForm	Форма авторизации
FddUserForm	Форма регистрации нового пользователя
AddDisForm	Форма добавления новой учебной дисциплины
JournalCreate	Форма создания страниц журнала
JournalForm	Страница журнала готовая для заполнения.

Действующие лица: администратор, пользователь, студент, база данных.

USE CASE DIAGRAM



ПОТОК СОБЫТИЙ /**FLOW OF EVENTS**/ ДЛЯ ВАРИАНТА ИСПОЛЬЗОВАНИЯ LOGIN

Краткое описание

Данный вариант описывает вход пользователя в систему

Основной поток событий

1.Система запрашивает имя пользователя и пароль

2.Пользователь вводит имя и пароль

3.Система проверяет имя и пароль. Если все верно, то открывается доступ к системе.

Альтернативный поток

Если введены неверные имя и пароль, то система выводит сообщение об ошибке. Пользователь может вернуться к началу выполнения основного потока или отказаться от входа в систему.

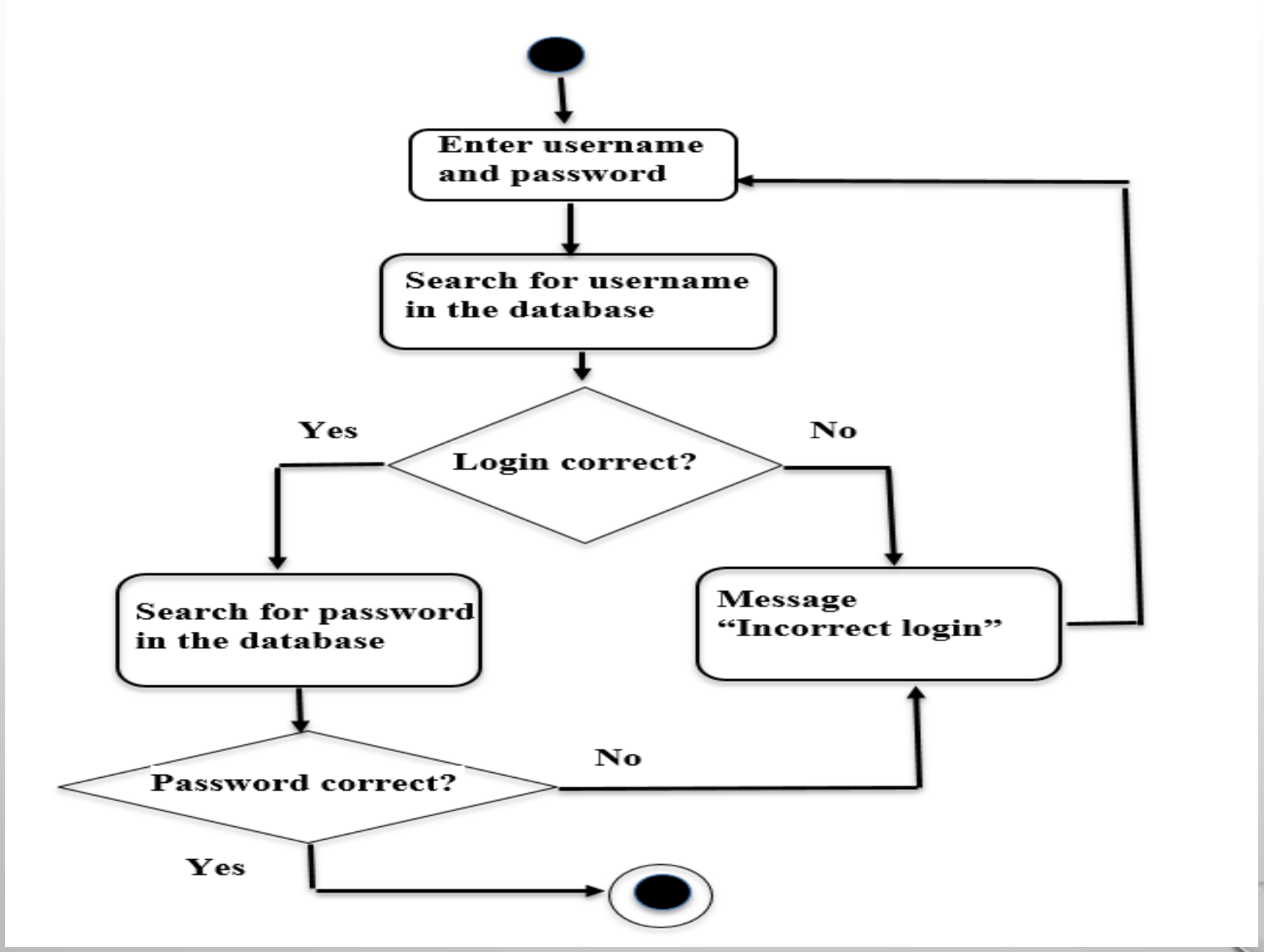
Предусловия

Отсутствуют

Постусловия

Если вариант выполнен успешно, то пользователь входит в систему.

ДИАГРАММА ДЕЯТЕЛЬНОСТИ /**ACTIVITY DIAGRAM**/ ДЛЯ LOGIN



ПРИМЕР 2

Электронная система управления документооборотом в гимназии.
Диаграммы вариантов
Диаграммы деятельности
Диаграммы последовательности
Диаграммы кооперации
Диаграмма классов
Диаграмма состояния

ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Проектируемая информационная система должна быть альтернативой бумажному варианту классных журналов.
Система должна обеспечивать:

- Преподаватель должен иметь возможность ввести и корректировать личные данные учащихся.
- Преподаватели должны иметь возможность проставить и исправить оценки.
- Дети и родители должны иметь возможность ознакомиться с оценками.
- Родители и ученики должны иметь возможность оставлять вопросы и сообщения преподавателям.
- Преподаватель должен иметь возможность прочитать сообщение и ответить на него.

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ

Вариант использования	**Краткое описание**
«Ввод или корректировка личных данных учащихся»	позволяет преподавателям ввести и затем корректировать личные данные учащихся.
«Узнать оценку»	позволяет учащимися и их родителями ознакомиться с отметками дистанционно по сети Интернет.
«Исправить оценку»	позволяет преподавателям исправить выставленную оценку.
«Выставить оценку»	позволяет преподавателям выставить оценку.
«Оставить сообщение»	позволяет всем пользователям оставить сообщение преподавателю
«Прочитать сообщение»	позволяет всем преподавателям прочитать сообщение и ответить на него.

ДИАГРАММА ДЕЯТЕЛЬНОСТИ

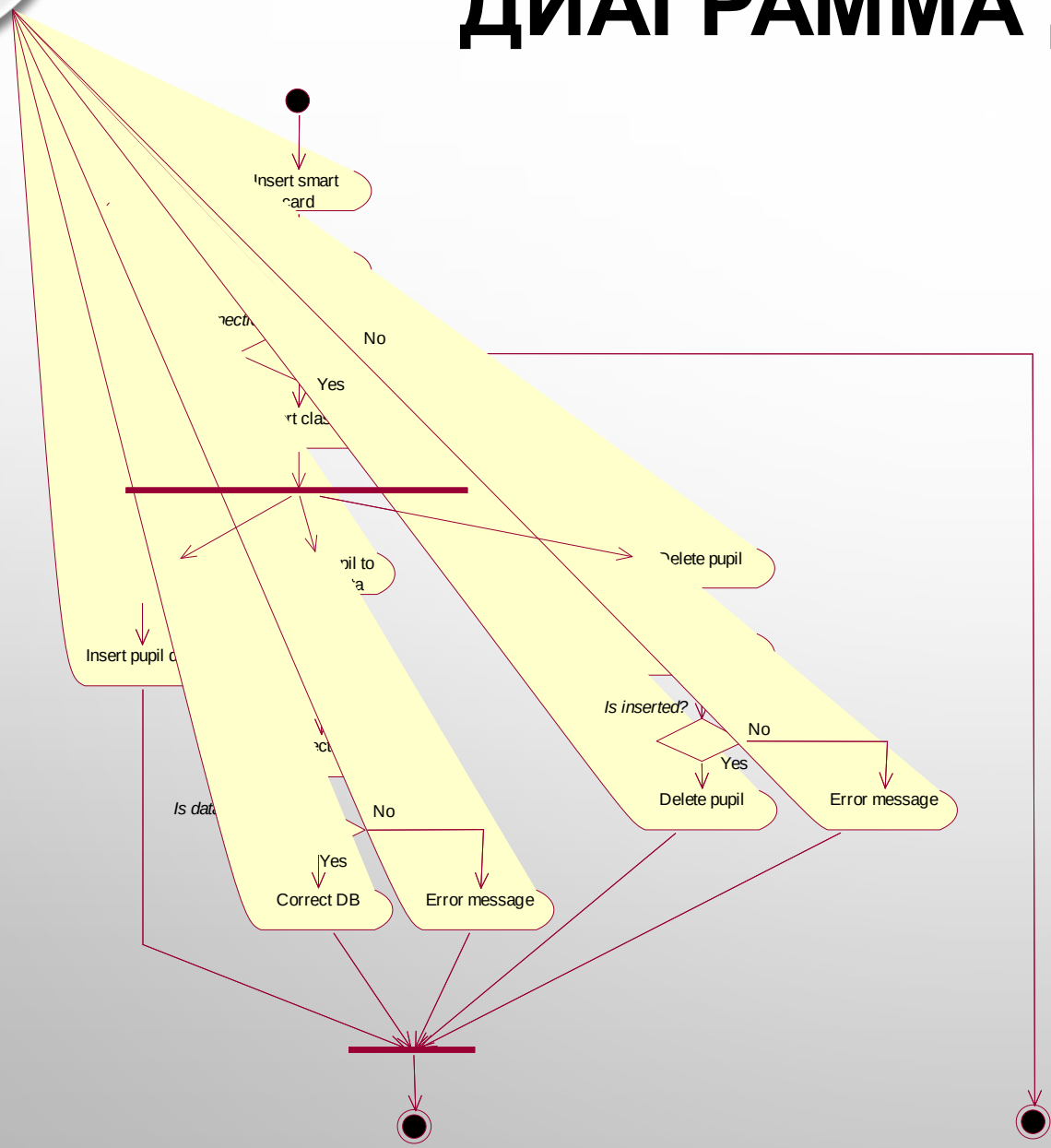
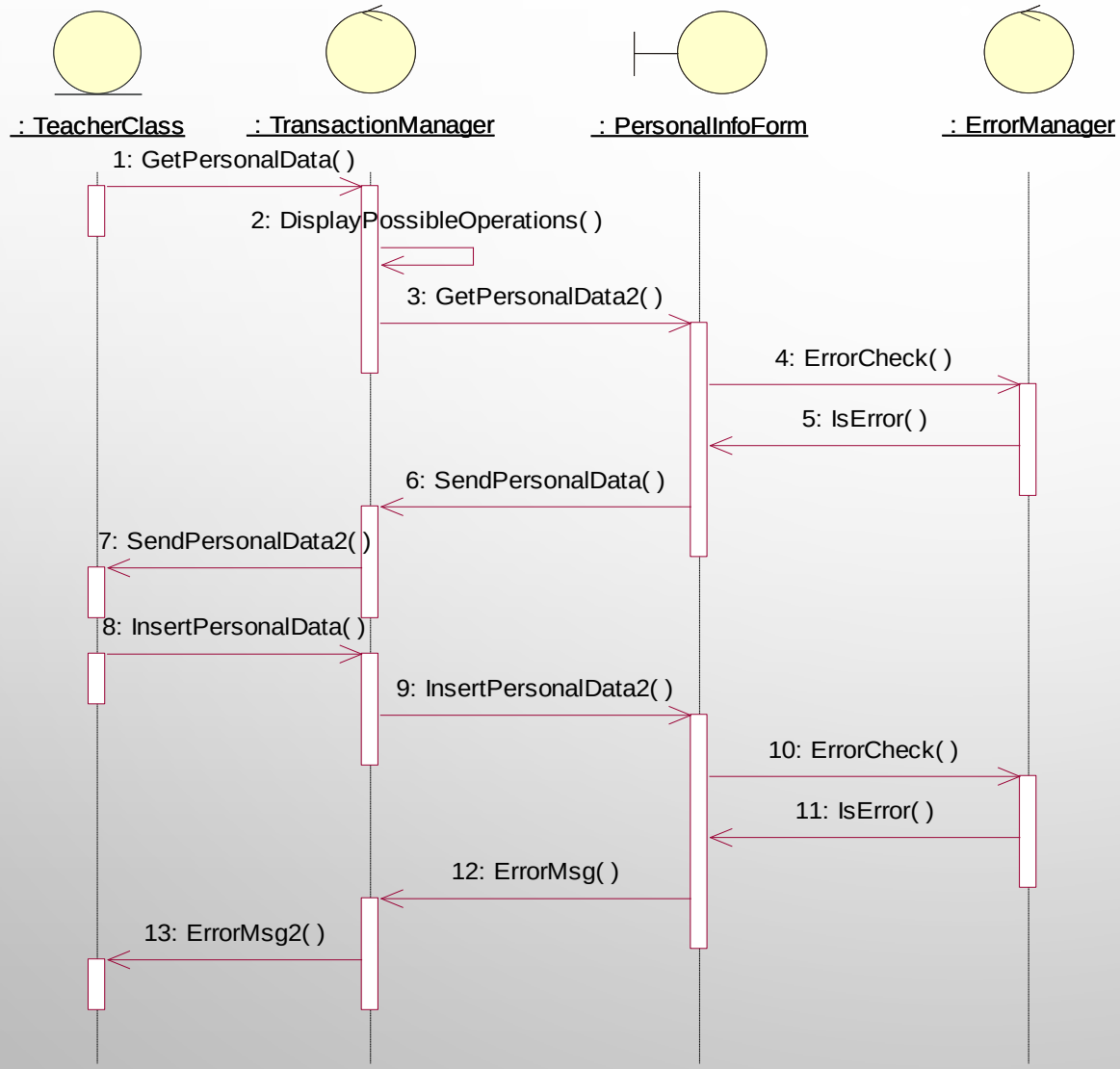


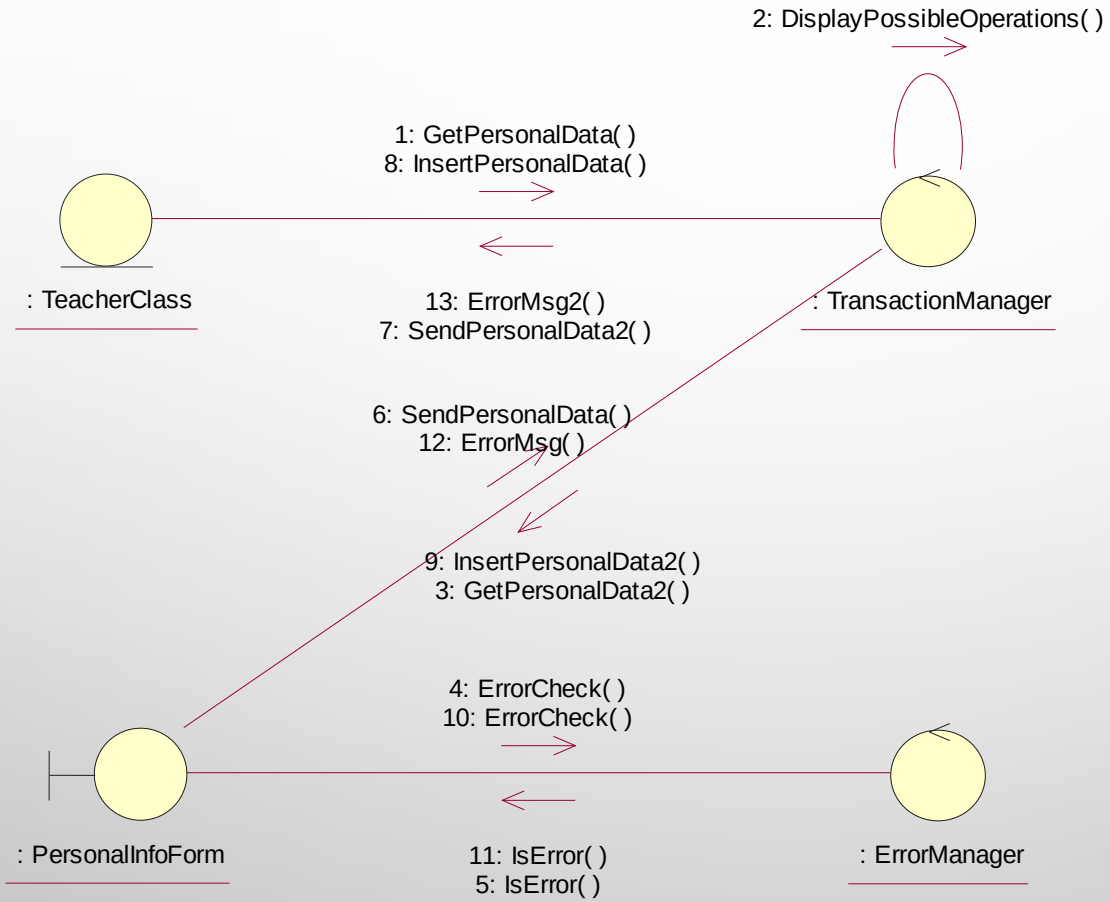
Диаграмма деятельности для варианта использования «Ввод или корректировка личных данных учащихся» (Insert or Correct Personal Data

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ



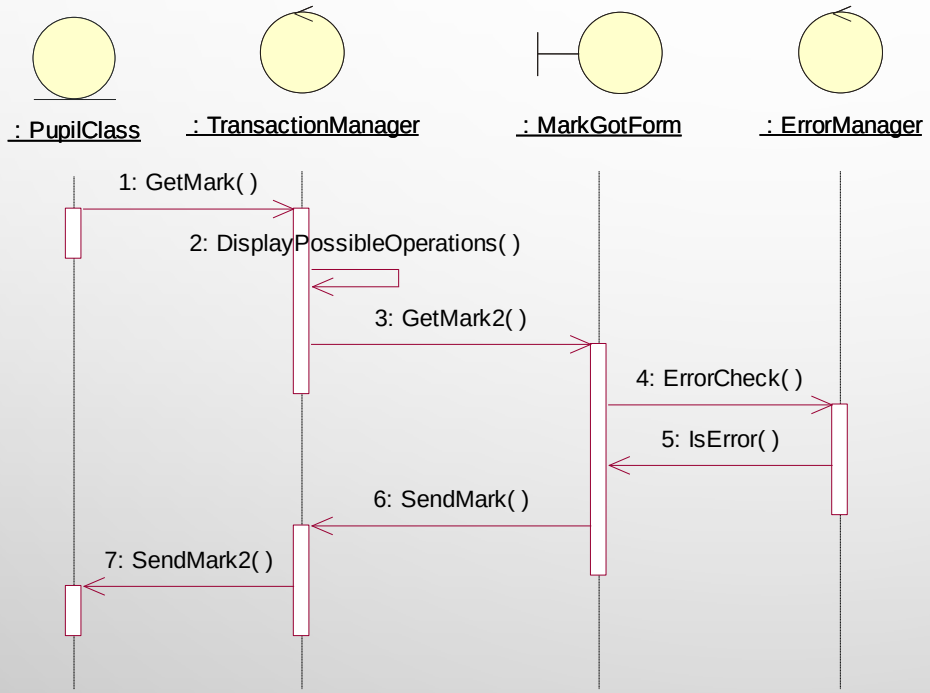
для варианта использования «Ввод или корректировка личных данных учащихся» (InsertPersonalData)

КООПЕРАТИВНАЯ ДИАГРАММА



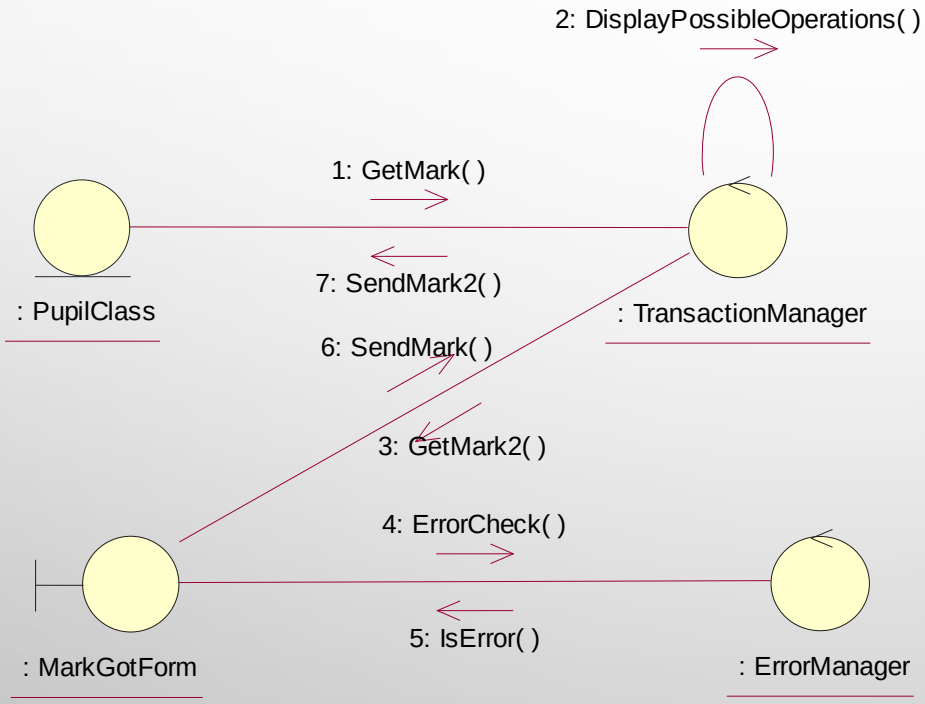
для варианта использования «Ввод или корректировка личных данных учащихся» (InsertPersonalData)

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ



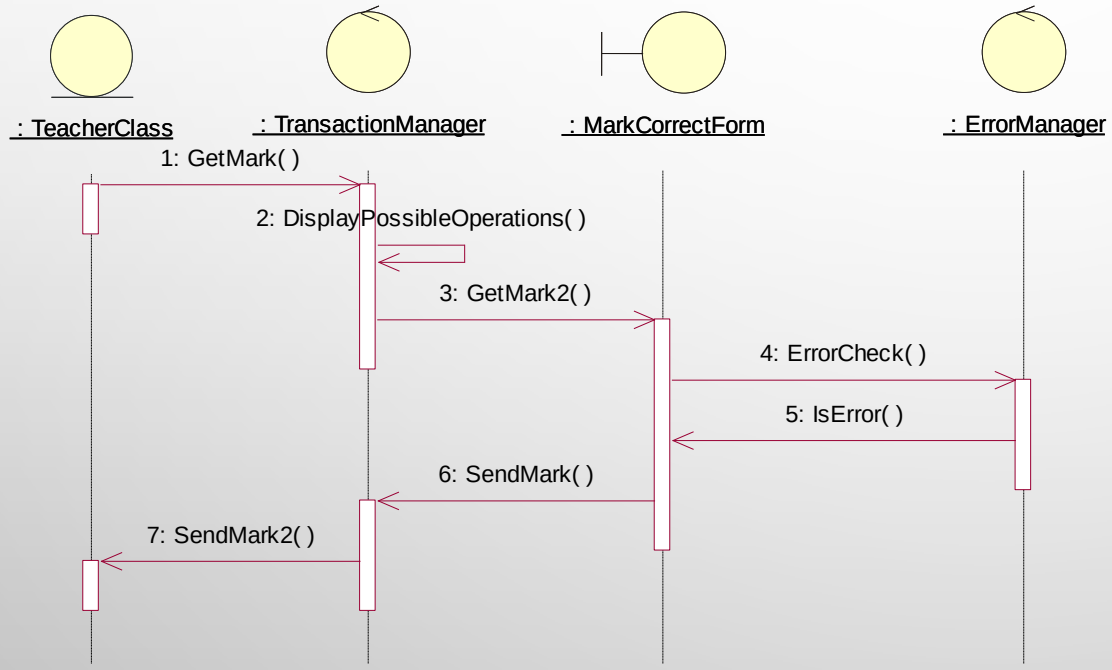
для варианта использования «Узнать оценку» (GotToKnowMark)

КООПЕРАТИВНАЯ ДИАГРАММА



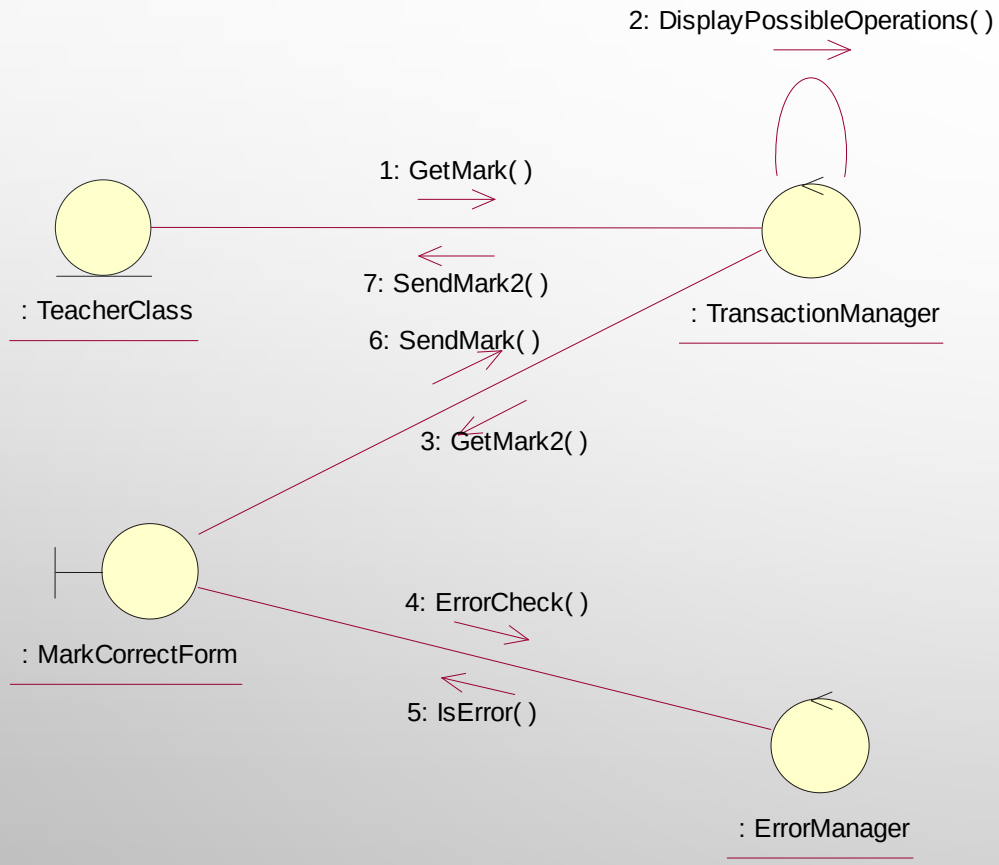
для варианта использования «Узнать оценку» (GotToKnowMark)

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ



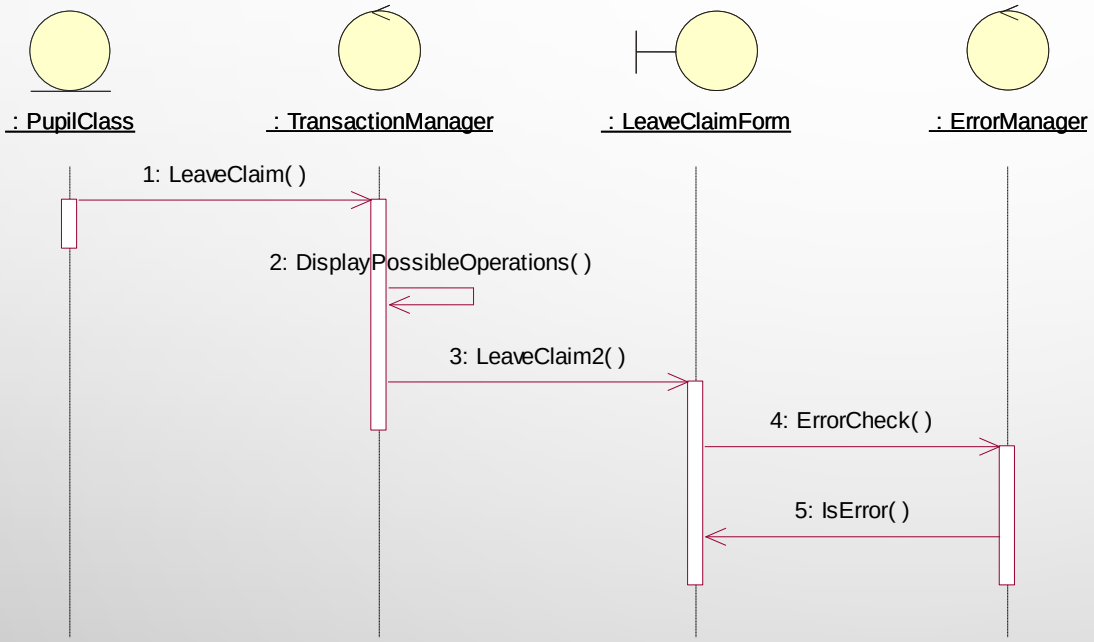
для варианта использования «Исправить оценку» (CorrectMark)

КООПЕРАТИВНАЯ ДИАГРАММА



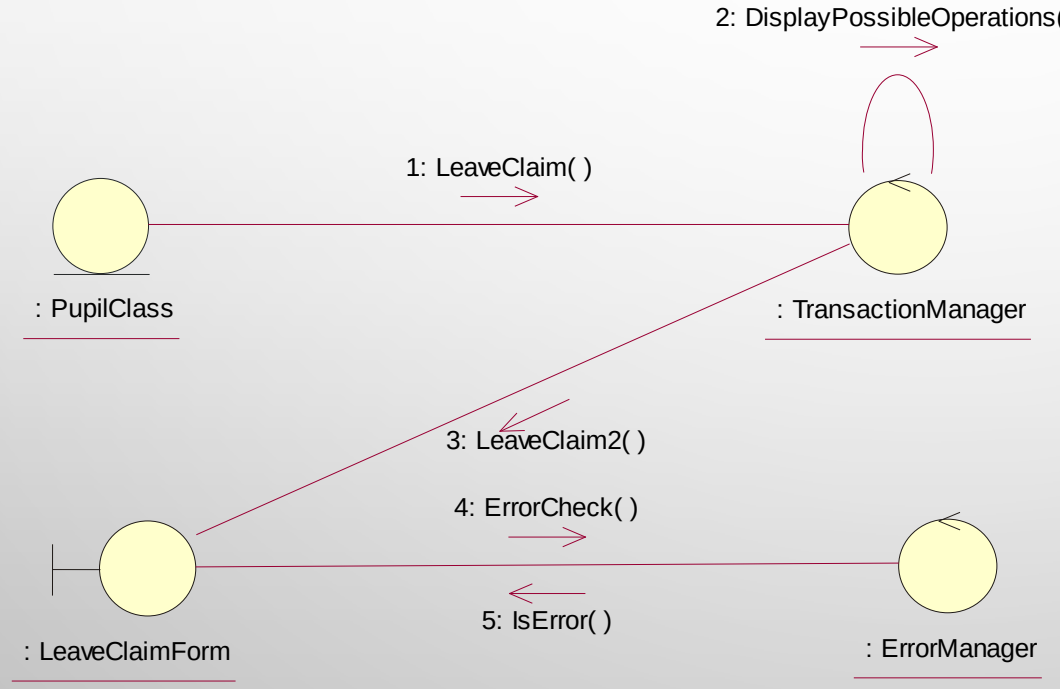
для варианта использования «Исправить оценку» (CorrectMark)

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ



для варианта использования «Оставить сообщение»

КООПЕРАТИВНАЯ ДИАГРАММА



для варианта использования «Оставить жалобу» (LeaveClaim)

ДИАГРАММА КЛАССОВ

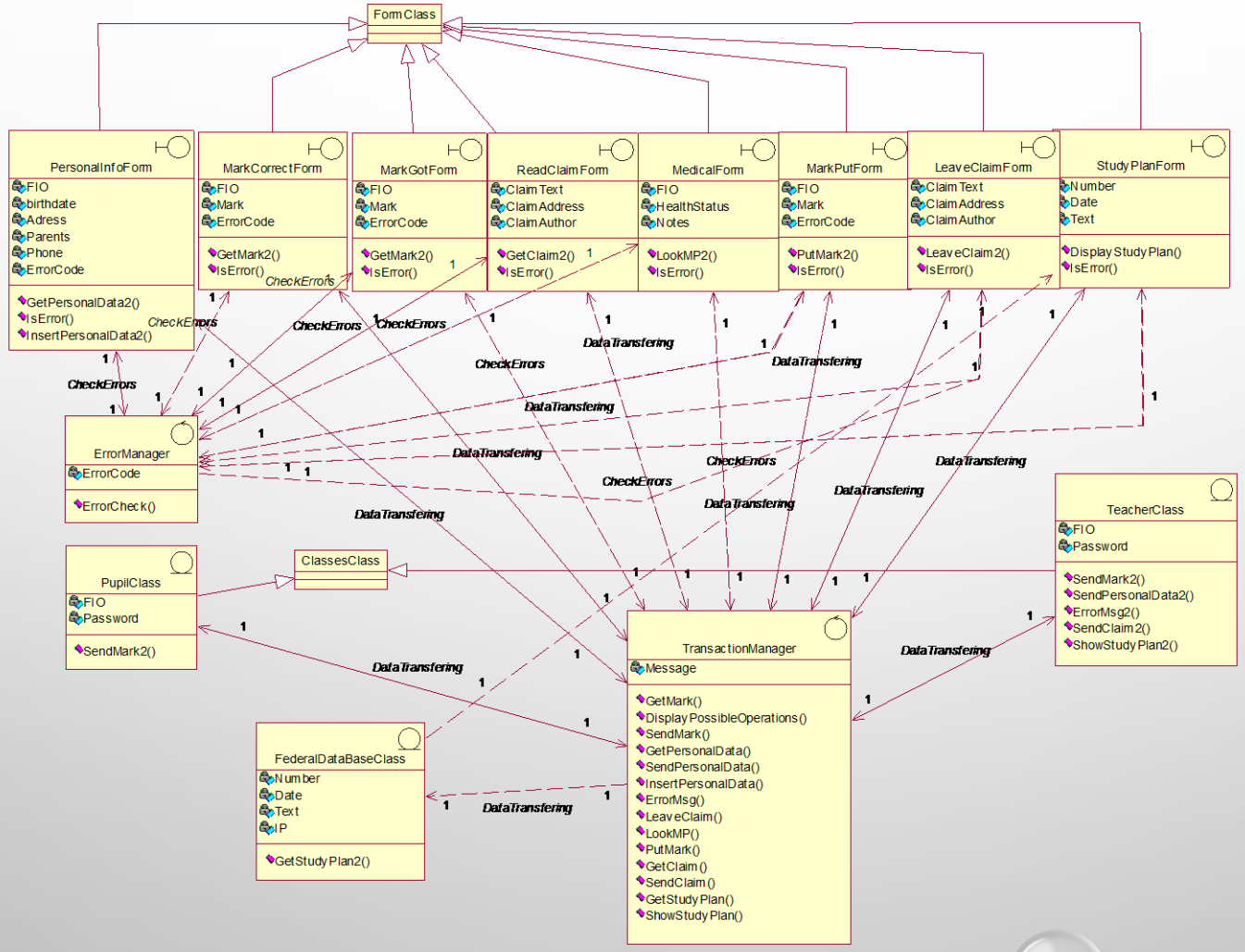


ДИАГРАММА СОСТОЯНИЙ

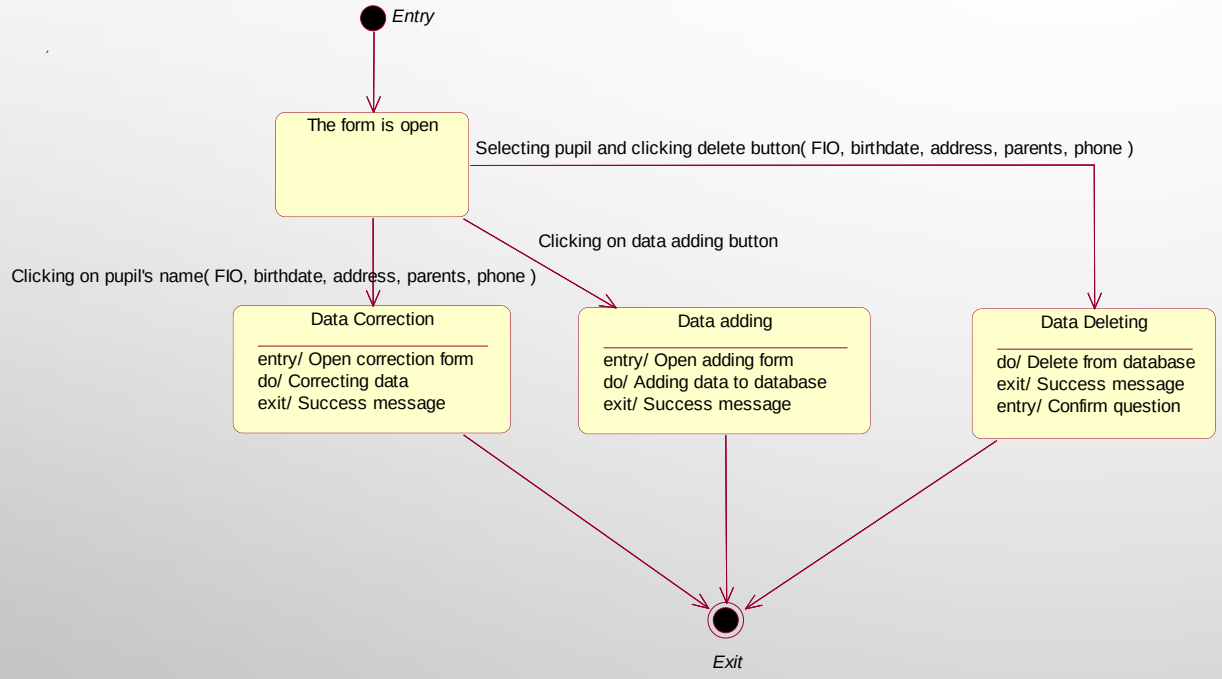


Диаграмма состояний для класса «Форма с персональными данными» (PersonalInfoForm

ПРИМЕР 3

ИНФОРМАЦИОННАЯ СИСТЕМА ДЛЯ МЕДИЦИНСКОГО ЦЕНТРА

ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Основные функции системы.

- вход и авторизация сотрудников в системе;
- регистрация пациентов в системе, назначение даты и времени осмотра;
- проведение осмотра, ввод информации в БД;
- печать результатов осмотра;
- импорт данных в центр аналитики.

ДЕЙСТВУЮЩИЕ ЛИЦА

a) *Пациент* – обращается в медицинский центр, регистрируется в системе, проходит осмотры, получает результаты, получает препараты после оплаты соответствующего счета;
b) *Оператор* – регистрирует пациентов в системе, передает необходимые данные в центр аналитики;
c) *Врач* – проводит осмотр, заносит информацию в БД, распечатывает необходимую документацию;

ДИАГРАММА КЛАССОВ

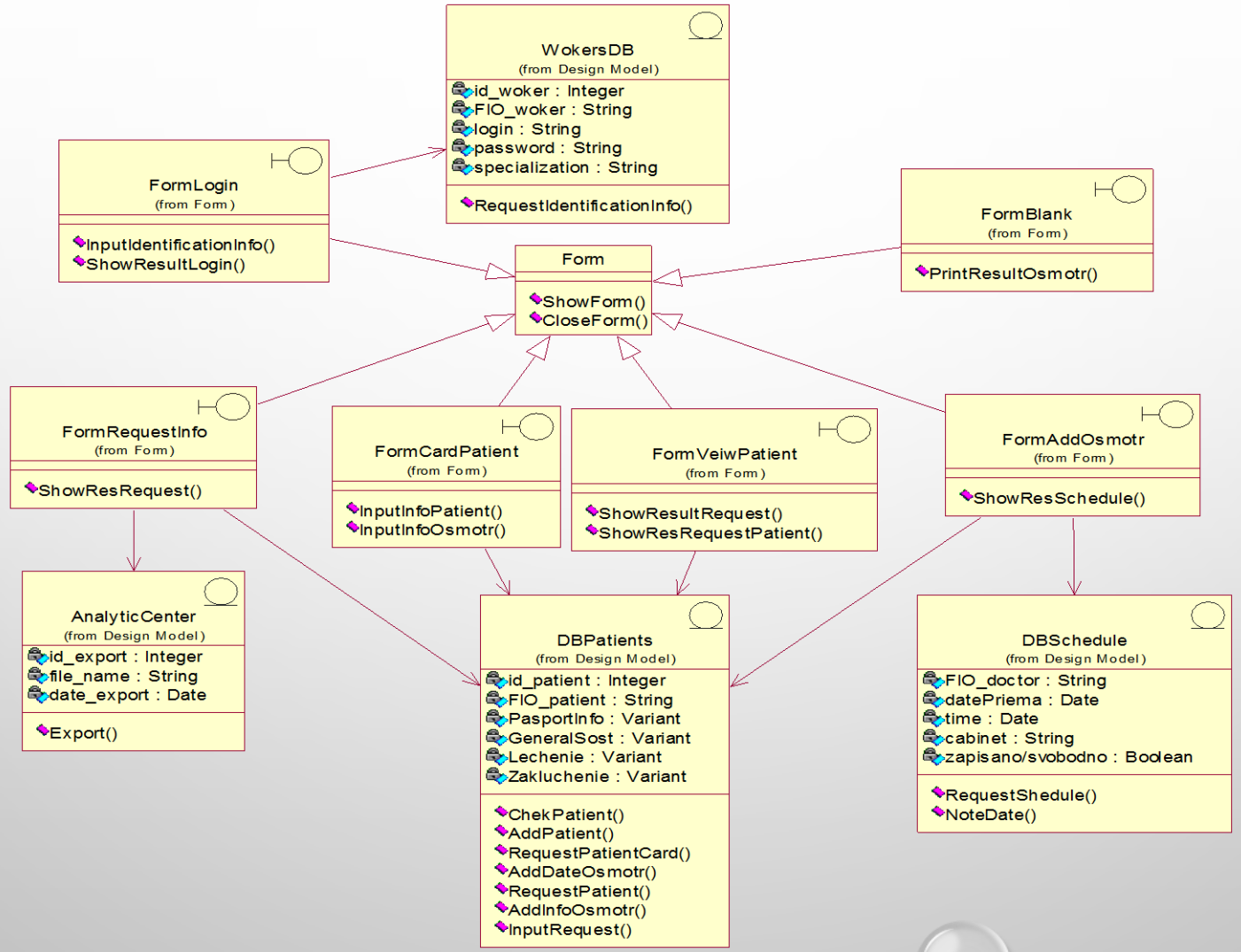


ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

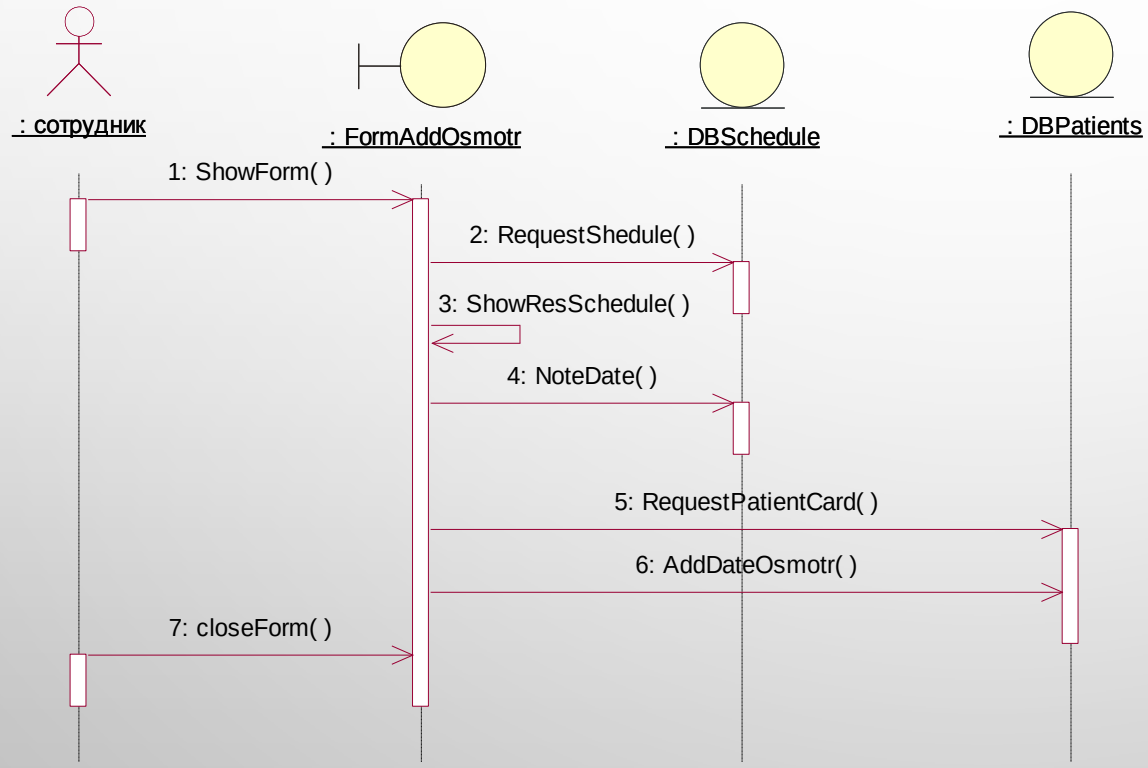


Диаграмма последовательности для варианта использования *«назначение осмотра»*

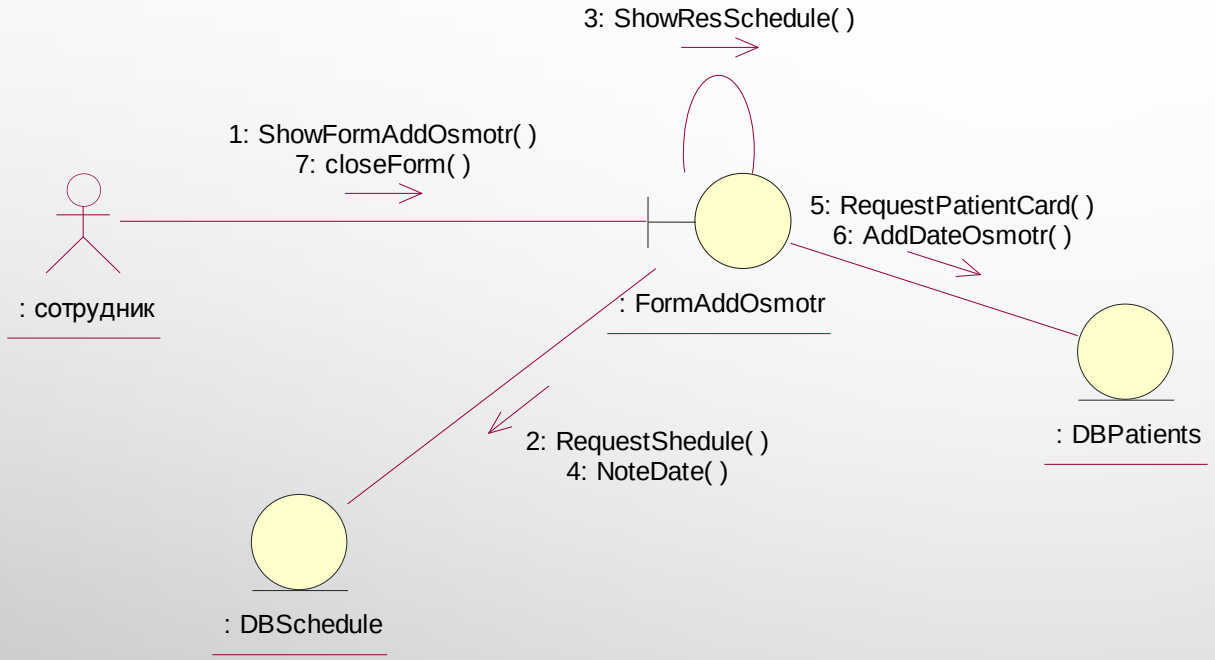


Диаграмма кооперации для варианта использования *«назначение осмотра»*

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

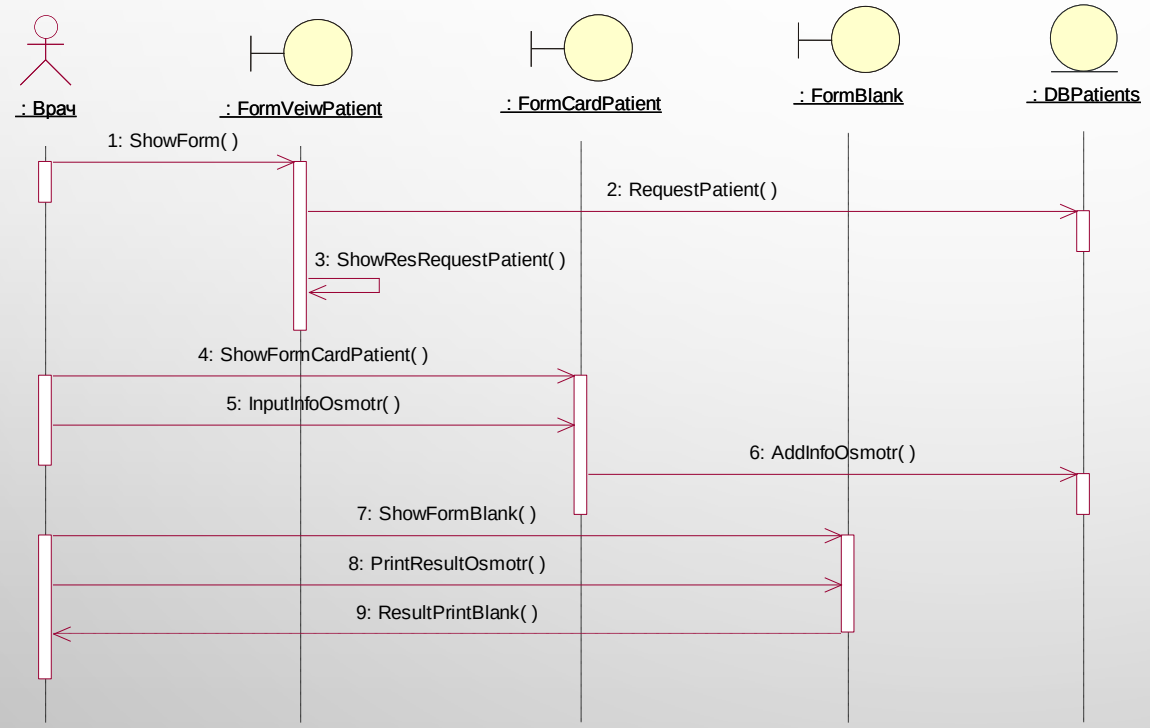


Диаграмма последовательности для варианта использования «проведение осмотра»

КООПЕРАТИВНАЯ ДИАГРАММА

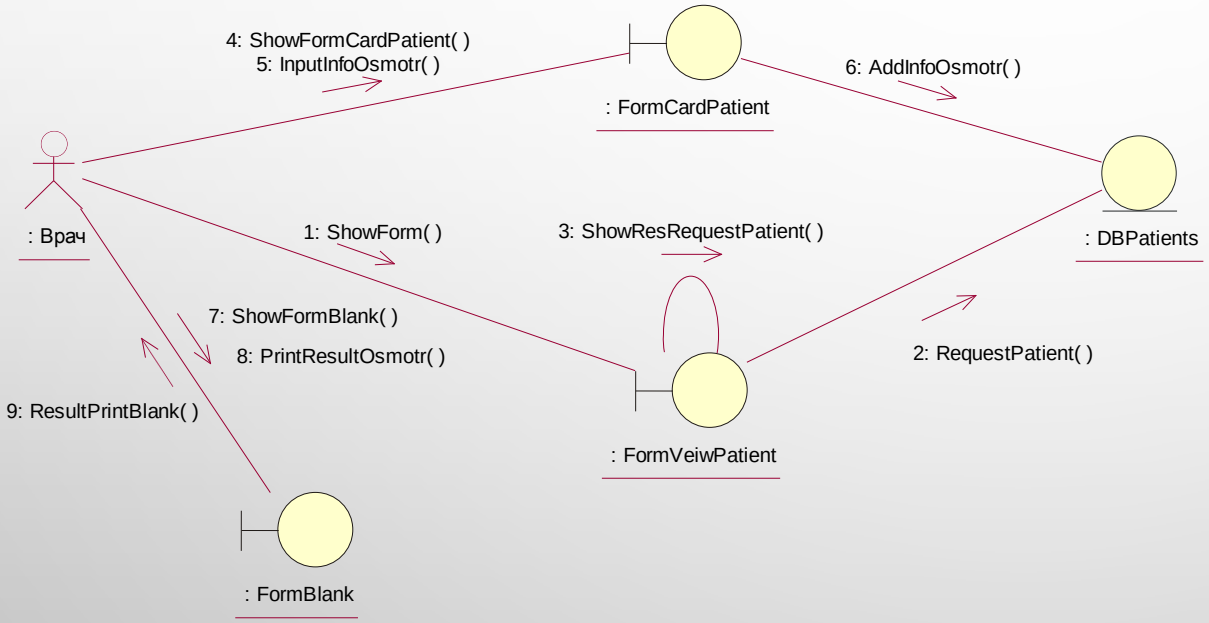


Диаграмма кооперации для варианта использования *«проведение осмотра»*

ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

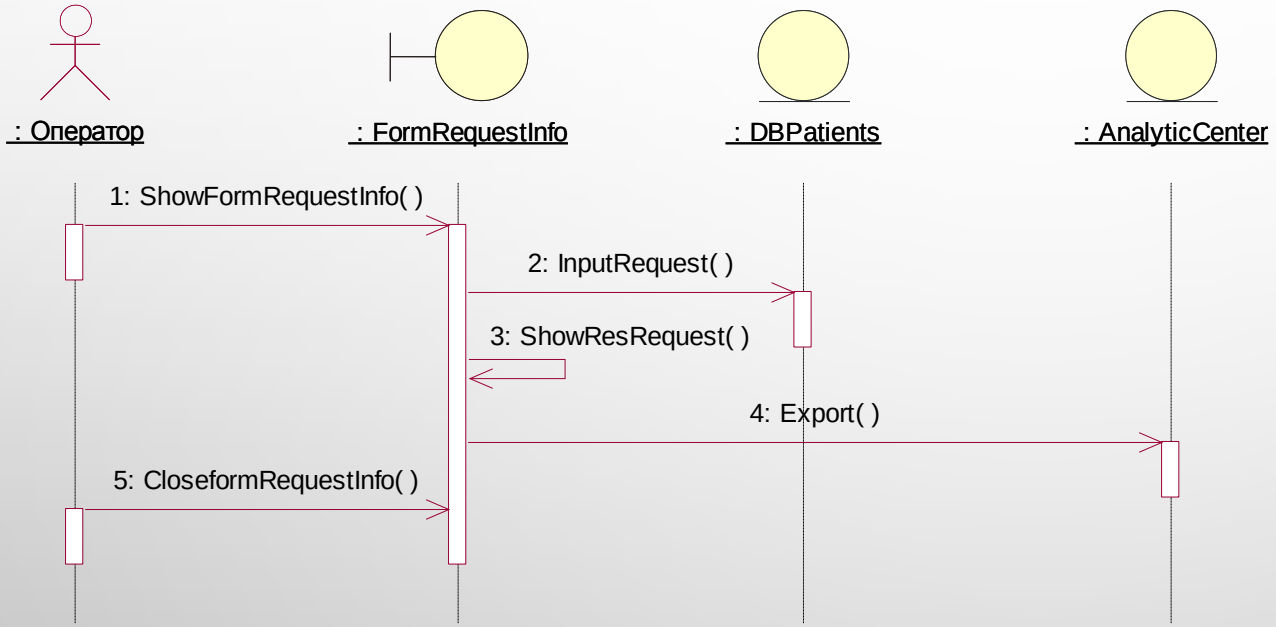


Диаграмма последовательности для варианта использования *«импорт данных»*

КООПЕРАТИВНАЯ ДИАГРАММА

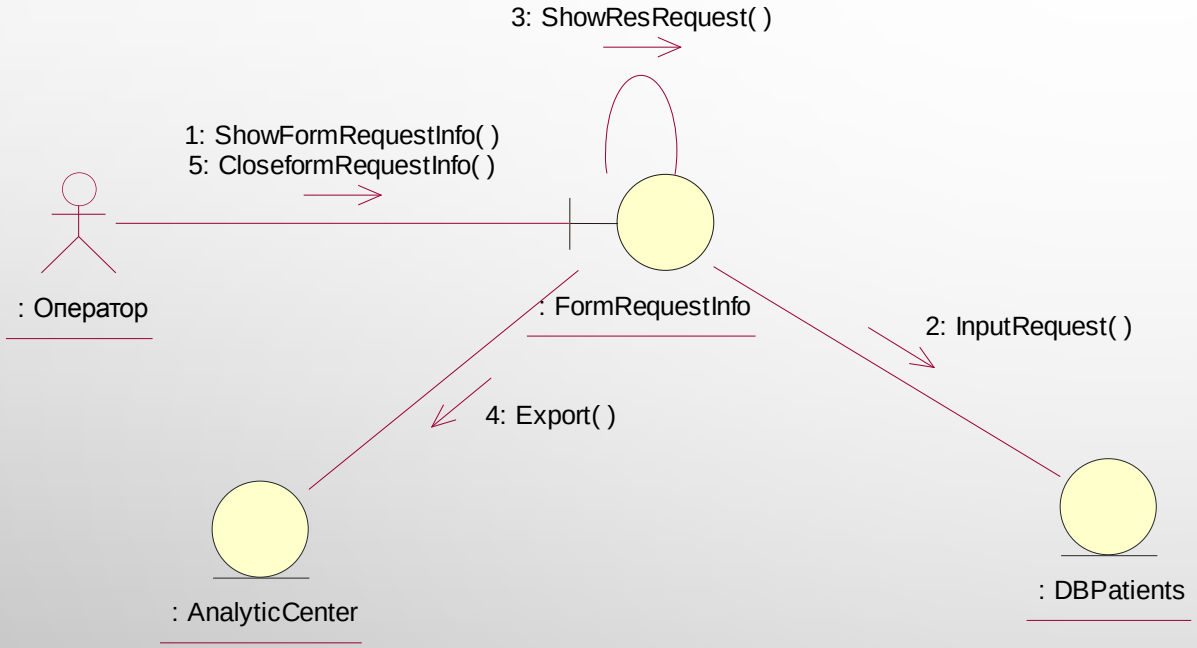
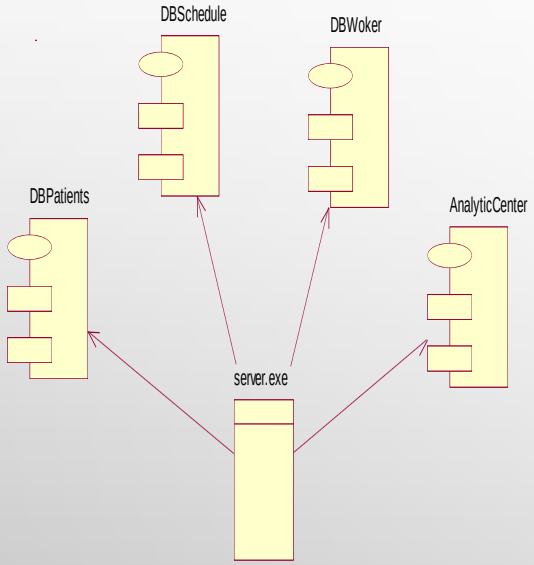


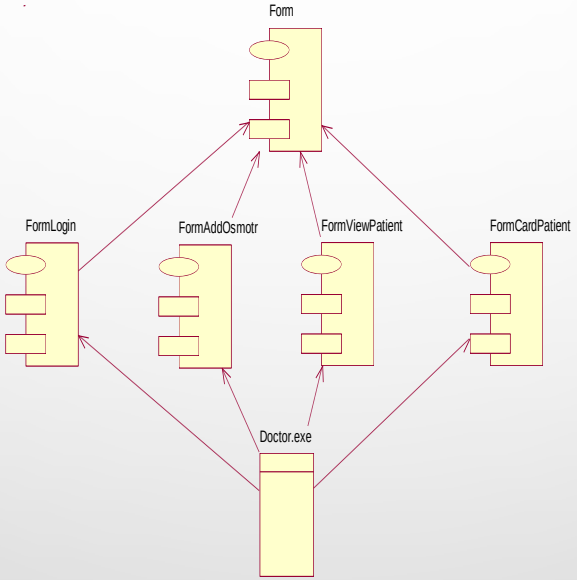
Диаграмма кооперации для варианта использования *«импорт данных»*

ДИАГРАММА КОМПОНЕНТОВ

для сервера данных



для *врача*



для оператора

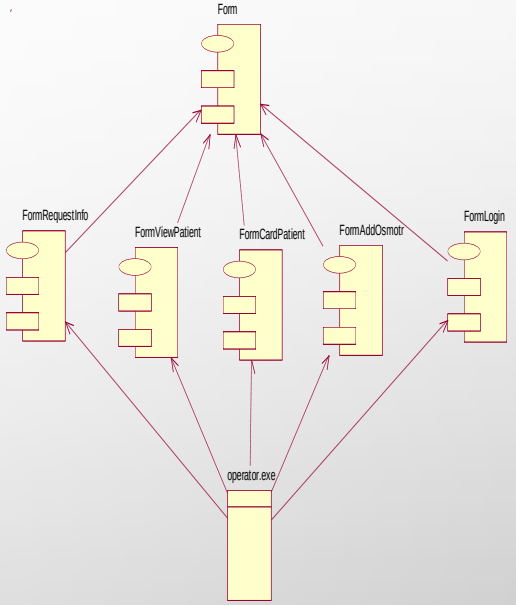


ДИАГРАММА РАЗМЕЩЕНИЯ