Литература

1. К. Ларман

Применение UML 2.0 и шаблонов проектирования.

Введение в объектно-ориентированный анализ, проектирование и итеративную разработку.

Третье издание, Вильямс, 2009 год.

2. Г. Буч

UML.

Второе издание, Питер, 2006 год.

3. Л. Мацяшек

Анализ и проектирование информационных систем с помощью UML 2.0

Третье издание, Вильямс, 2008 год.

4. Г. Буч, А. Якобсон, Дж. Рамбо

UML. Классика CS.

Второе издание, Питер, 2006 год.

5. А.Леоненков

Объектно-ориентированный анализ и проектирование с использованием UML и Rational Rose.

Бином, 2006 год.

Литература (продолжение)

6. Р. Фатрелл, Д. Шафер, Л. Шафер
Управление программными проектами.
Вильямс, 2006 год.
7. Э. Эванс
Предметно-ориентированное проектирование.
Вильямс, 2011 год.
8. Г. Буч и др.
Объектно-ориентированный анализ и проектирование с примерами приложений
Третье издание, Вильямс, 2008 год.
9. С. Макконнелл
Совершенный код, Русская редакция, 2007 год.
10. Сайт университета информационных технологий.
www.intuit.ru
11. Бабич А.В. Введение в UML
www.intuit.ru, 2008

Расходование средств на ИТ

МИР	1997	2001	2003	2007	2010	2011
ПО	16	21	22	25	26	27
АС	48	36	33	30	29	27
У	36	43	45	45	45	46

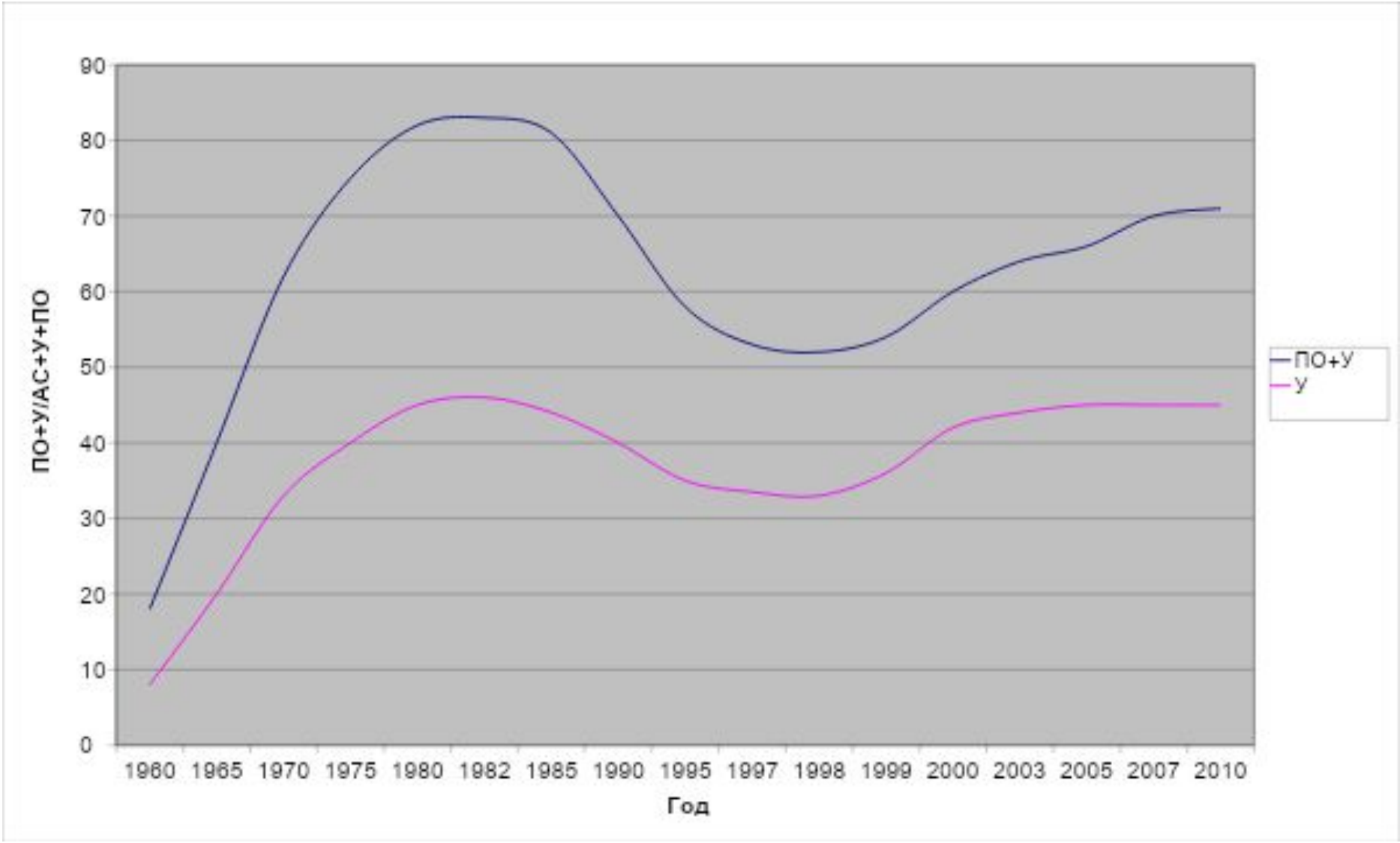
Россия	1999	2003	2005	2008	2010	2011
ПО	9	12	11	14	16	20
АС	77	68	67	66	61	52
У	14	20	22	20	23	28

Программное обеспечение (ПО)
- Системное
- Средства разработки
- Прикладное

Аппаратные средства (АП)
- Компьютеры
- Мониторы
- Периферийное оборудование

Услуги (У)
- Консалтинг
- Системная интеграция
- Установка и сопровождение
- Обучение
- Разработка заказного ПО

Изменение относительной стоимости ПО и У



Российские производители ПО

Типы программных продуктов (распространение):

- Software *коммерческое*
- Shareware *условно-бесплатное*
- Freeware *бесплатное, свободное*
- Open source *с открытым кодом*

Производители:

ABBYY – распознавание текстов, словари

Лаборатория Касперского – антивирусные программы

1С – автоматизация работы предприятий, бухгалтерия, игры

Spirit – микропрограммное ПО

Parallels – виртуализация ПО

Модели оценки качества ПО

- Обобщенные критерии качества
- Элементарные критерии качества

  Каждый элементарный критерий может влиять на несколько обобщенных
- Показатели качества или метрики

  Каждая метрика влияет только на один элементарный критерий

Обобщенные критерии качества

1. Функциональность — Functionality
2. Мобильность — Mobility
3. Надежность — Reliability
4. Эффективность — Performance
5. Модифицируемость — Serviceability
6. Практичность

(понятность и простота использования) — Usability

Элементарные критерии качества

1. Точность
2. Согласованность
3. Структурированность
4. Отсутствие избыточности
5. Универсальность
6. Защищенность

...

Метрики

1. Число строк кода – Lines Of Code (LOC)
2. Число обнаруженных ошибок за месяц работы ПО
3. Число строк документации
4. Число различных операндов
5. Наличие средств проверки входных данных
6. Число внешних вводов
7. Число внешних выводов
8. Число классов
9. Глубина иерархии классов
10. Степень взаимосвязанности классов
11. Число переопределяемых методов
12. Время разработки в человеко-месяцах (характеризует процесс разработки)
13. Стоимость разработки (характеризует процесс разработки)
14. Относительные характеристики: KLOC, число ошибок / KLOC, стоимость / LOC, число строк документации / KLOC

.....

Для чего нужны критерии?

Анализ – оценка уже разработанного ПО с точки зрения значений критериев качества

Синтез – разработка ПО, имеющего определенные значения критериев качества

Прогноз – предсказание значений критериев качества до начала разработки ПО

Критерии выбора языка программирования

- Соответствие языка характеру решаемой задачи
- Надежность
- Возможности управления аппаратными средствами
- Быстрота трансляции
- Эффективность объектного кода
- Сервисные возможности (средства отладки, работа с файлами, встроенная помощь, навигация)
- Интеграция со средствами организации коллективной работы
- Поддержка языка CASE-средствами, в частности, возможность автоматической генерации кода на данном языке

Жизненный цикл ПО

1. Технико-экономическое обоснование разработки
2. Анализ требований (анализ предметной области)
3. Проектирование
4. Программирование
5. Тестирование и отладка
6. Эксплуатация и сопровождение

Технико-экономическое обоснование разработки

- постановка задачи (определение основных функций системы);
- определение экономических возможностей заказчика;
- определение технической базы;
- определение сроков разработки;
- определение требований к безопасности.

В результате должно быть сформировано обобщенное техническое задание (ТЗ) на разработку системы. Разработка ТЗ осуществляется заказчиком.

ТЗ может быть скорректировано заказчиком после консультаций с исполнителем.

Техническое задание

1. НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ
2. ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ
3. ТРЕБОВАНИЯ К СИСТЕМЕ
   3.1 Требования к системе в целом
   3.2 Требования к функциям
   3.3 Требования к интерфейсу
   3.4 Требования к данным
   3.5 Требования к техническим средствам
   3.6 Требования к безопасности
4. КАЛЕНДАРНЫЙ ПЛАН РАБОТ
5. ПОРЯДОК КОНТРОЛЯ И ПРИЁМКИ СИСТЕМЫ
6. ТРЕБОВАНИЯ К ДОКУМЕНТАЦИИ

Стратегии разработки ПО

Функционально-ориентированная стратегия

технологии:

- Нисходящая (водопадная)
- Расширения ядра

Объектно-ориентированная стратегия

технологии:

- Спиральная
- Эволюционная или инкрементная
- Гибкая (agile software development)

Риски при разработке ПО

1. Дефицит необходимых специалистов.
2. Нереалистичные сроки
3. Недостаточный бюджет.
4. Реализация несоответствующей требованиям функциональности.
5. Разработка неудачного пользовательского интерфейса.
6. “Золотая сервировка”, то есть ненужная оптимизация и оттачивание деталей проекта.
7. Непрекращающийся поток изменения требований со стороны заказчика.
8. Недостаточная производительность разрабатываемой системы.
9. Несоблюдение требований безопасности.

Для чего нужны технологии?

1. Повысить качество ПО.
2. Снизить сроки разработки.
3. Уменьшить стоимость.
4. Обеспечить контроль над ходом разработки.

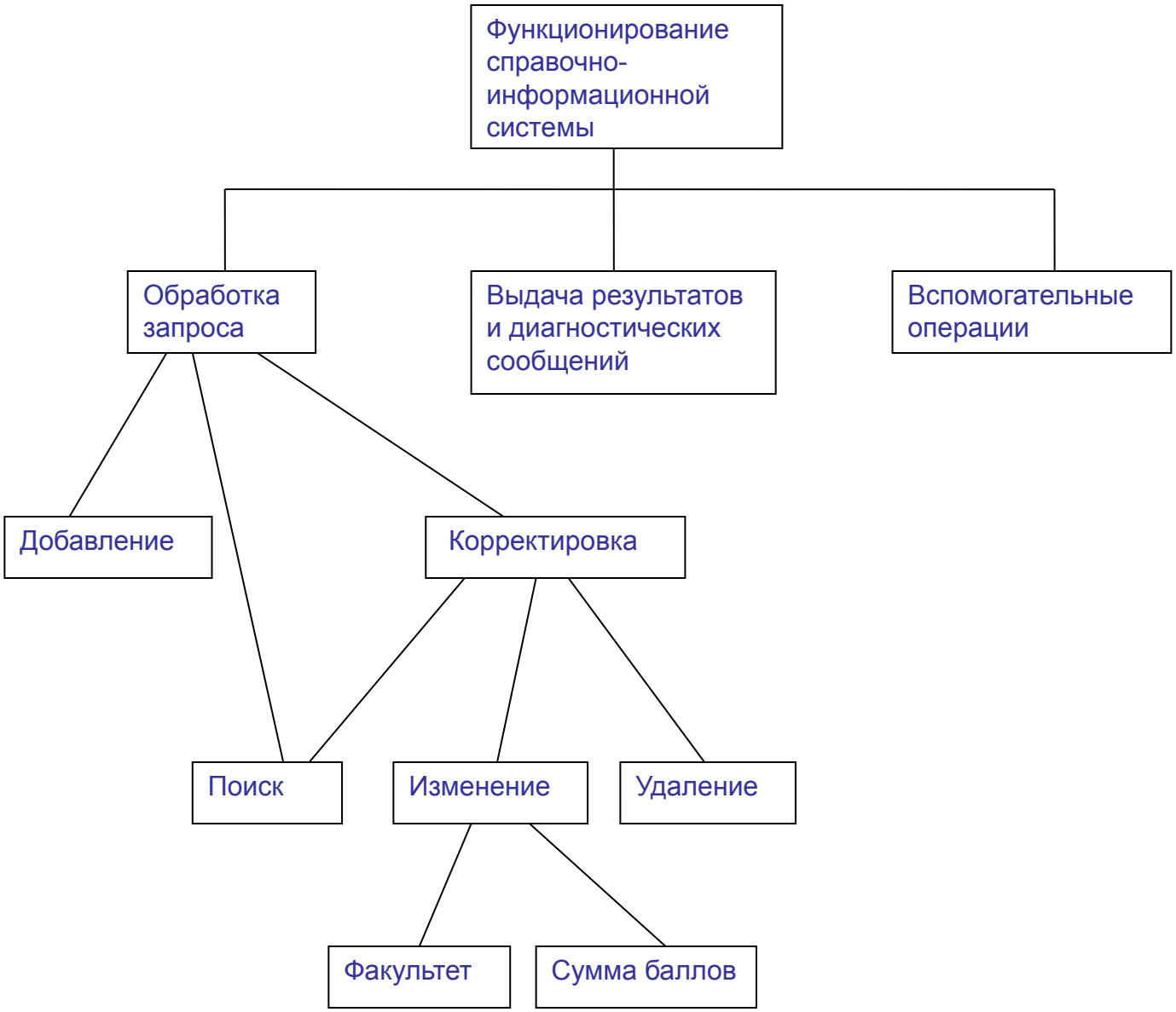
Нисходящая технология

Основная цель этапа проектирования – построение схемы иерархии.

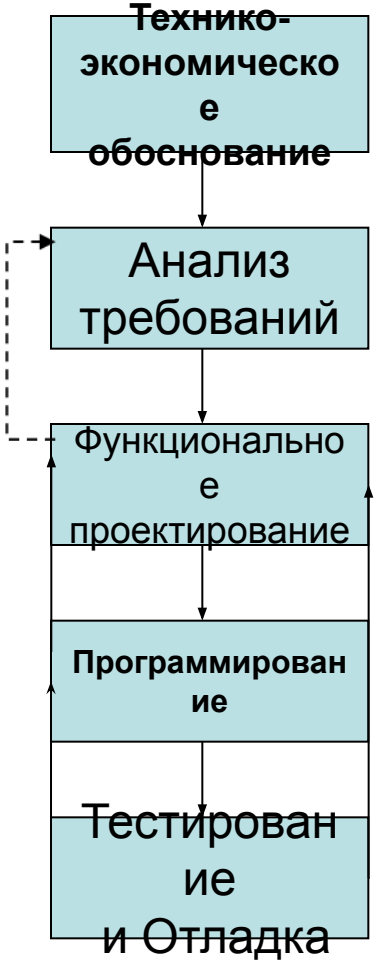
Схема иерархии – функциональная схема, представляющая собой ориентированный граф. Вершины – функции (подпрограммы), ребра – отношения функция-подфункция.

В процессе построения схемы иерархии каждая новая функция рассматривается как черный ящик. В схеме иерархии не отражены потоки данных и логика работы программной системы.

Схема иерархии



Функционально-ориентированные технологии: этапы разработки



Возвраты на предыдущие этапы нежелательны

Достоинства ФОС разработки

- Возможность планирования всего процесса разработки, поскольку требования к ПС должны быть четко определены с самого начала
- Богатый опыт использования ФОС
- Адекватность примерно 20% предметных областей

Недостатки ФОС разработки

- Неадекватность по отношению к большинству предметных областей
- Требования к ПС должны быть четко определены с самого начала и не должны изменяться
- Последовательное выполнение всех этапов разработки
- Невозможность в большинстве случаев создания прототипа системы
- Невозможность выбора альтернатив
- Сложность внесения изменений в готовую систему
- Повышение трудоемкости к концу разработки
- Недостаточное внимание уделяется данным
- Желательно наличие у разработчиков опыта работы над аналогичными проектами

Анализ требований

- Диаграммы прецедентов
- Словарь терминов

Исполнитель – внешняя по отношению к системе сущность, обладающая поведением.

Прецедент – действие которое система может выполнять.

Диаграмма прецедентов – схема, на которой отображаются отношения между исполнителями и прецедентами, с одной стороны, и между прецедентами, с другой.

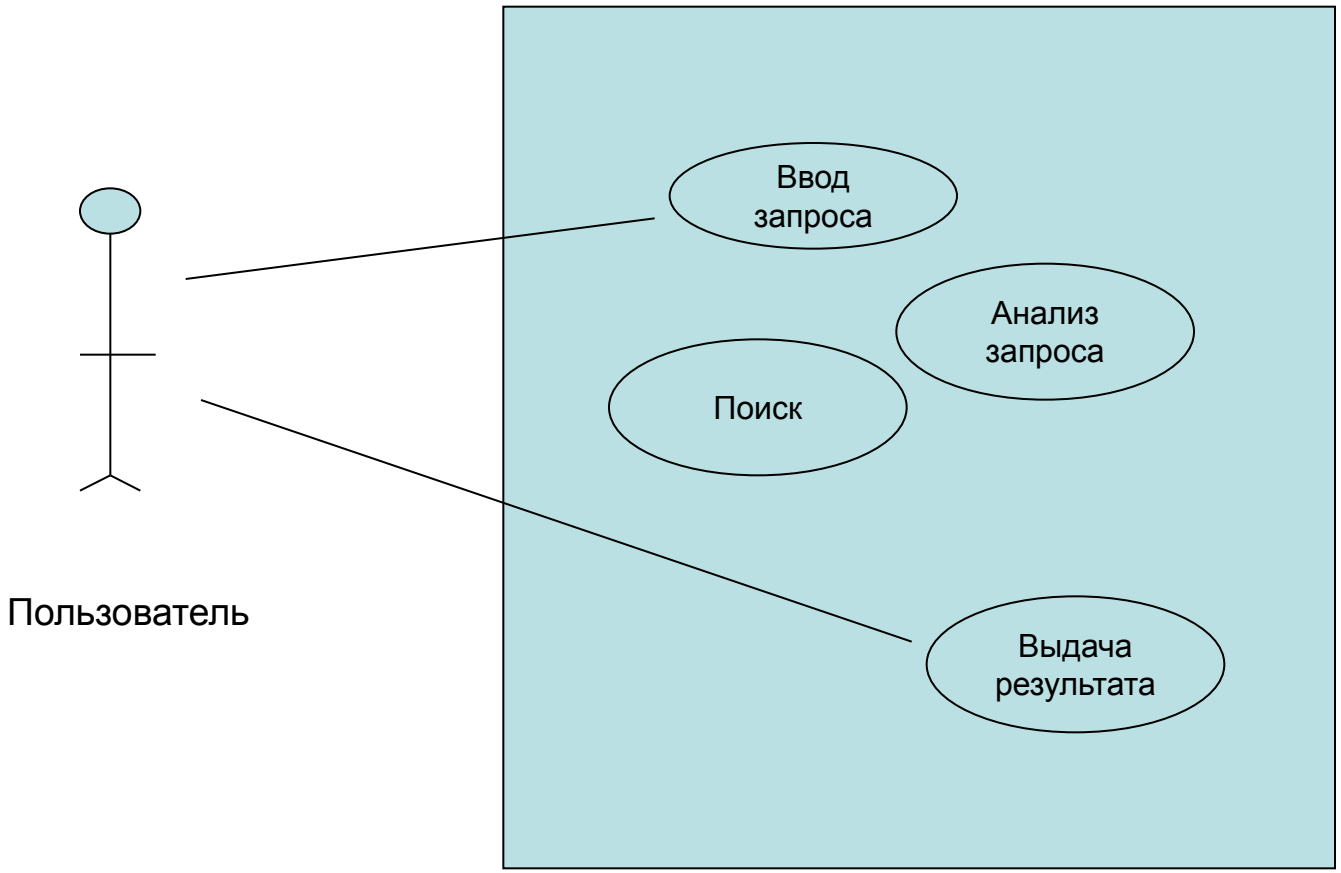
Типы отношений

Исполнитель и прецедент связывает отношение ассоциации, если, либо исполнитель инициирует выполнение прецедента, либо исполнитель анализирует и использует результаты работы прецедента.

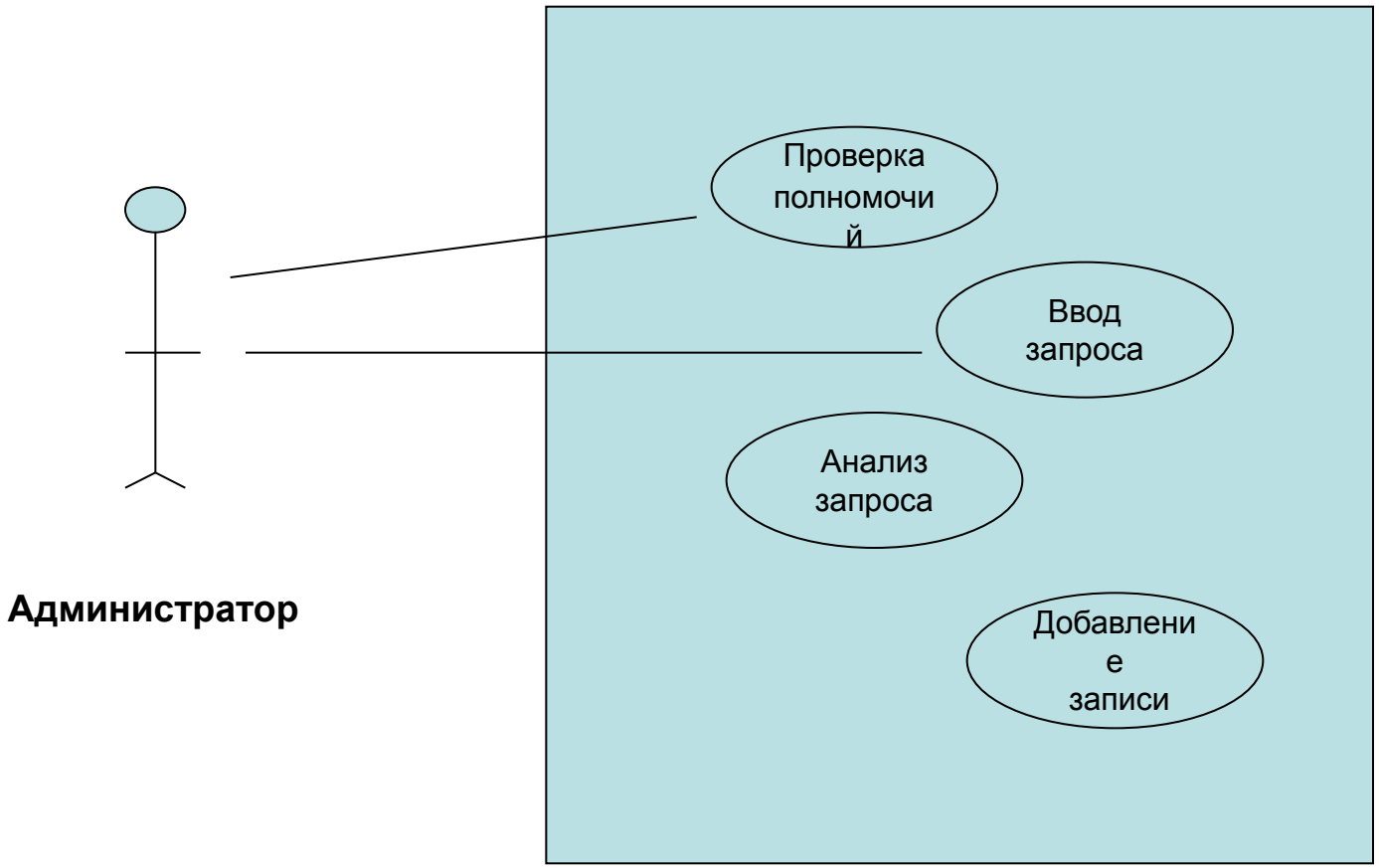
Два прецедента могут связывать отношения обобщения и зависимости.

Обобщение – это отношение наследования.
Зависимость определяет условные и безусловные отношения между прецедентами.

Диаграммы прецедентов

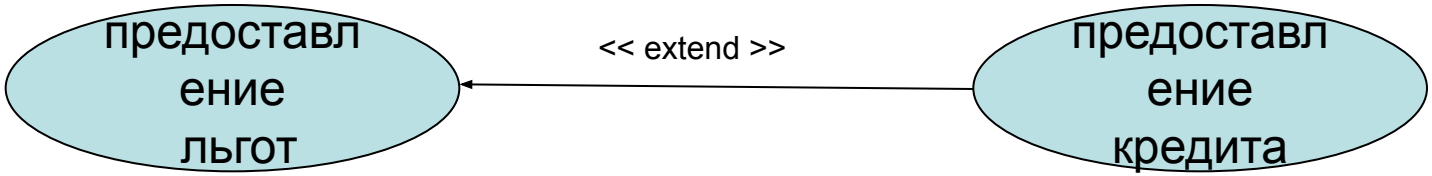


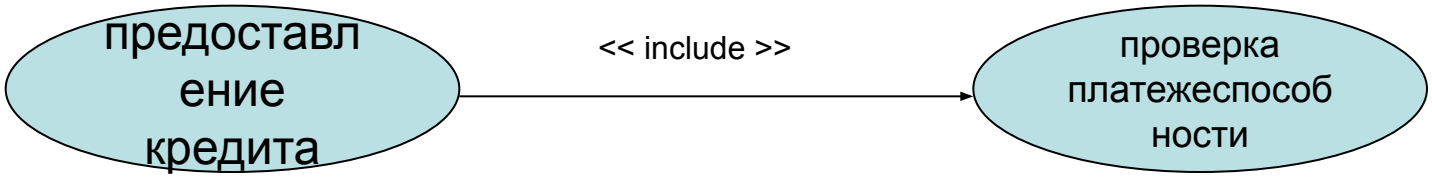
Диаграммы прецедентов



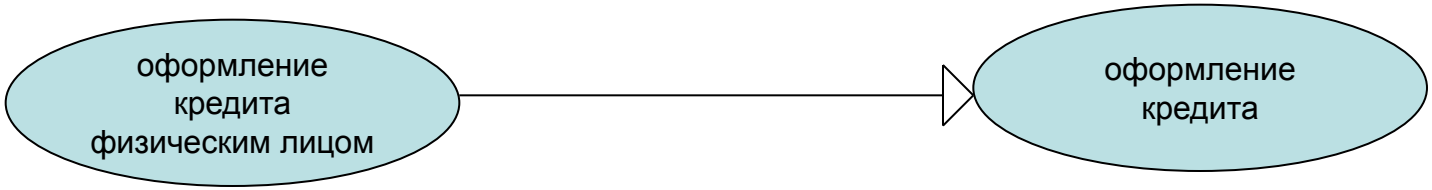
Отношения между прецедентами

Зависимость





Обобщение



Диаграммы прецедентов

Описание сценария

Исполнители	Банкомат
1. Клиент помещает карту в банкомат	2. Проверяет карту
	Исключение1. Карта недействительна
4. Клиент вводит PIN-код	3. Просит ввести PIN-код
	5. Проверяет PIN-код
	Исключение 2. PIN-код неверен
7. Клиент выбирает снятие денег	6. Отображает на экране меню
	8. Делает запрос в банк и выясняет состояние счета
	Исключение 3. Счет пуст
10. Клиент вводит сумму	9. Предлагает клиенту ввести сумму
11. Банк проверяет сумму	
	Исключение 4. Сумма больше допустимой
12. Банк изменяет состояние счета	13. Возвращает кредитную карту, выдает деньги и чек
14. Клиент получает деньги, чек и кредитную карту	

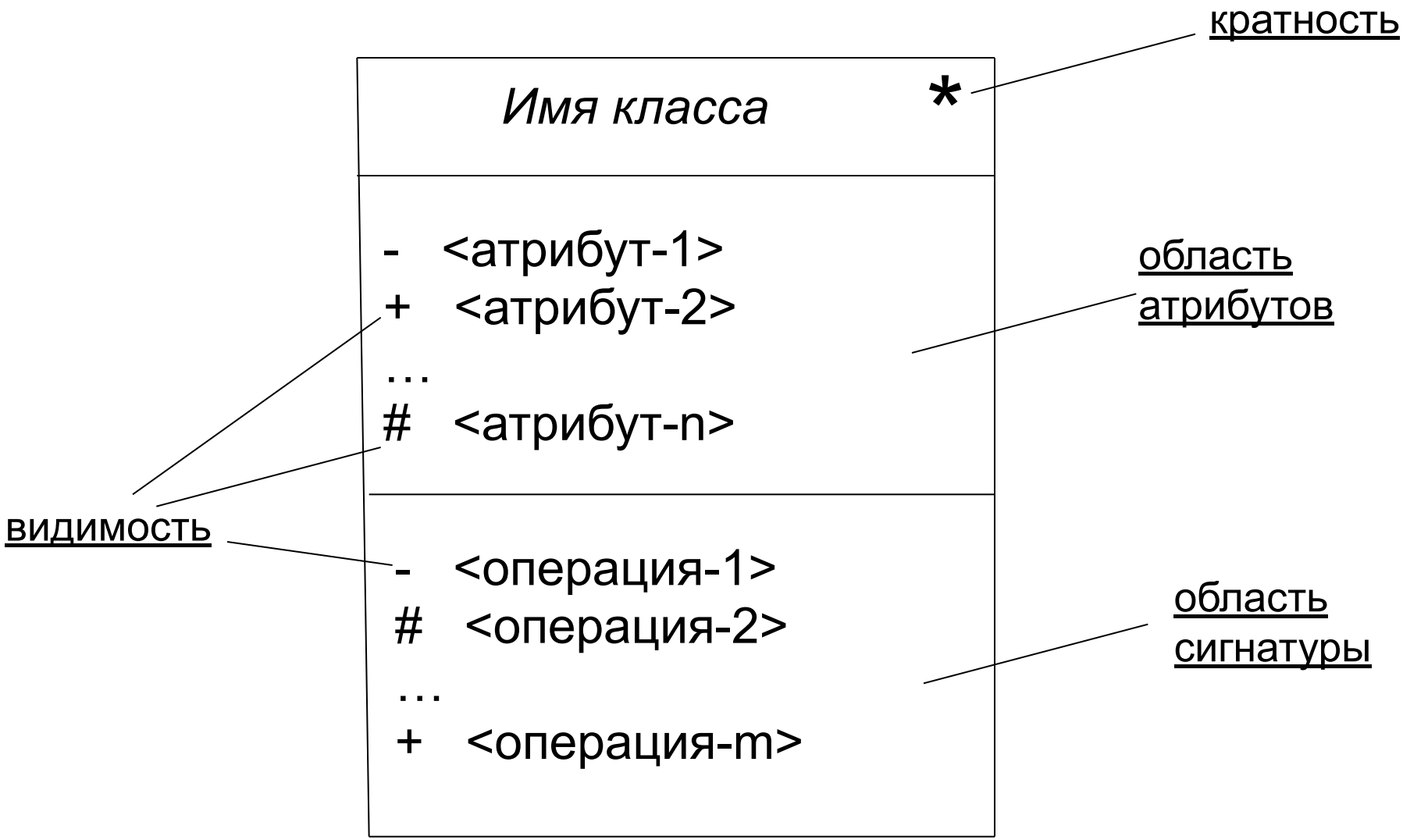
Классы, объекты и методы

- Классы нужны для описания нового типа, содержащего поля и методы
- Объекты – это экземпляры классов
- Методы реализуют функционал класса

<имя объекта>.<имя метода>(<аргументы>)

Работа объектно-ориентированной программы может быть представлена как последовательность действий, состоящая из вызовов методами друг друга.

Шаблоны классов



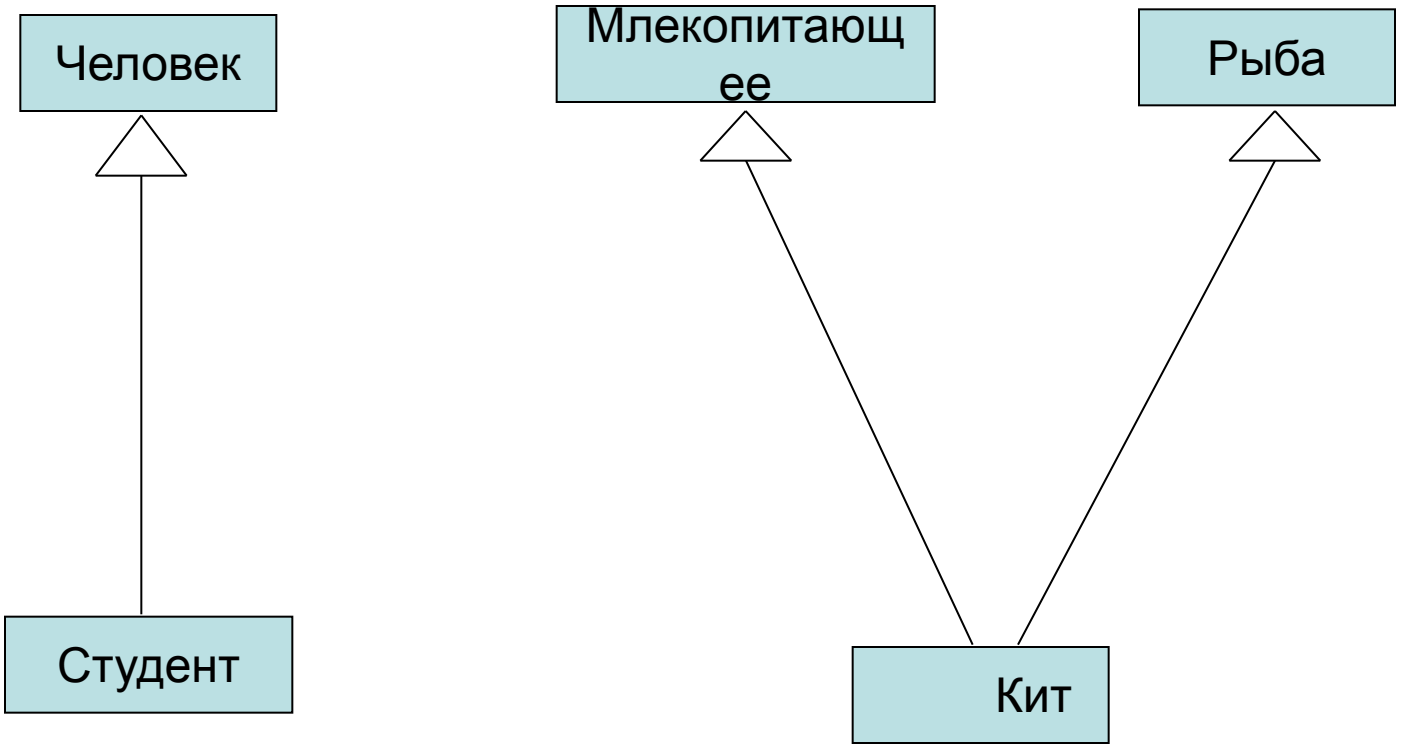
Отношения между классами

- Обобщение
  - простое
  - множественное

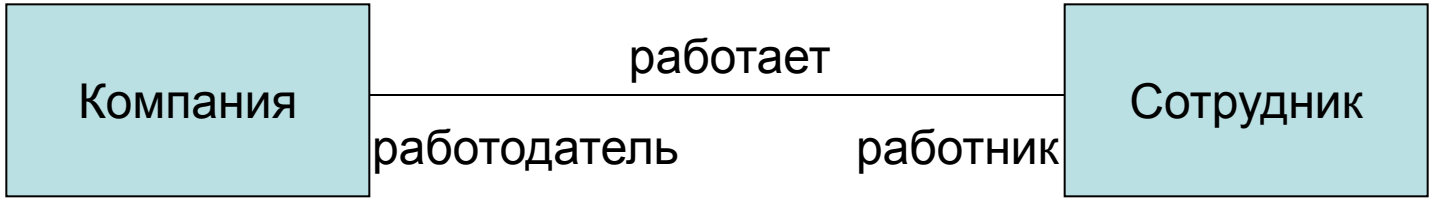
- Ассоциация
  - простая
  - агрегирование
  - композитное агрегирование
  - классы-ассоциации

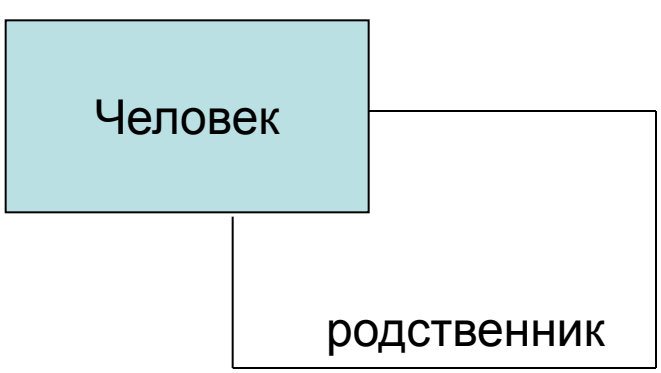
- Зависимость

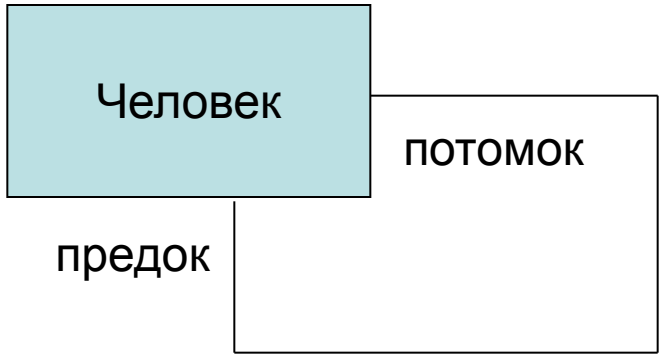
Отношение обобщения



Отношение ассоциации



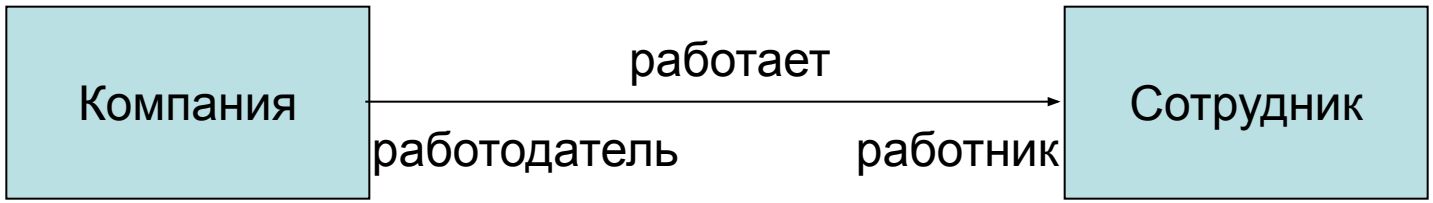




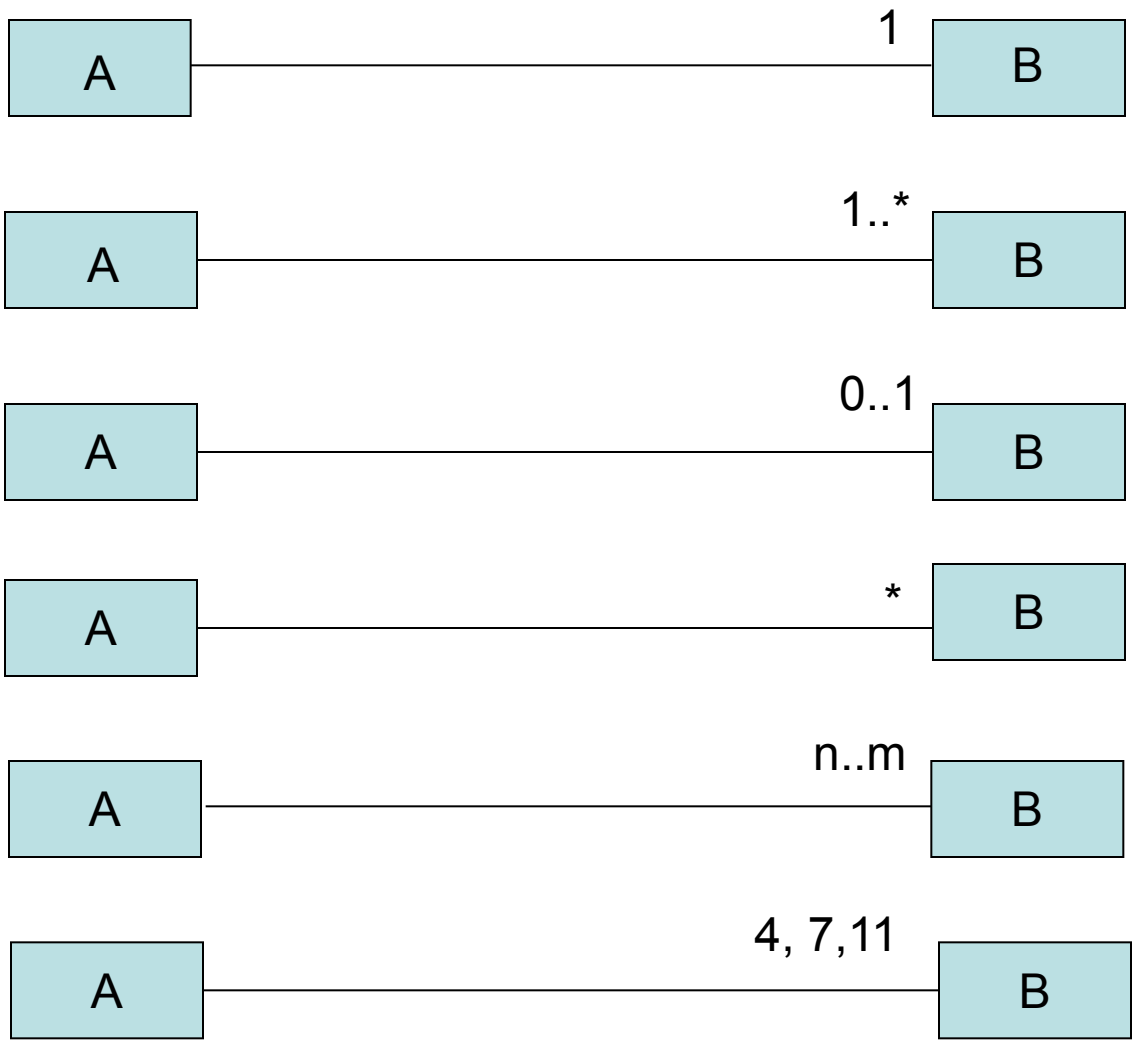
Прямое родство

Навигация

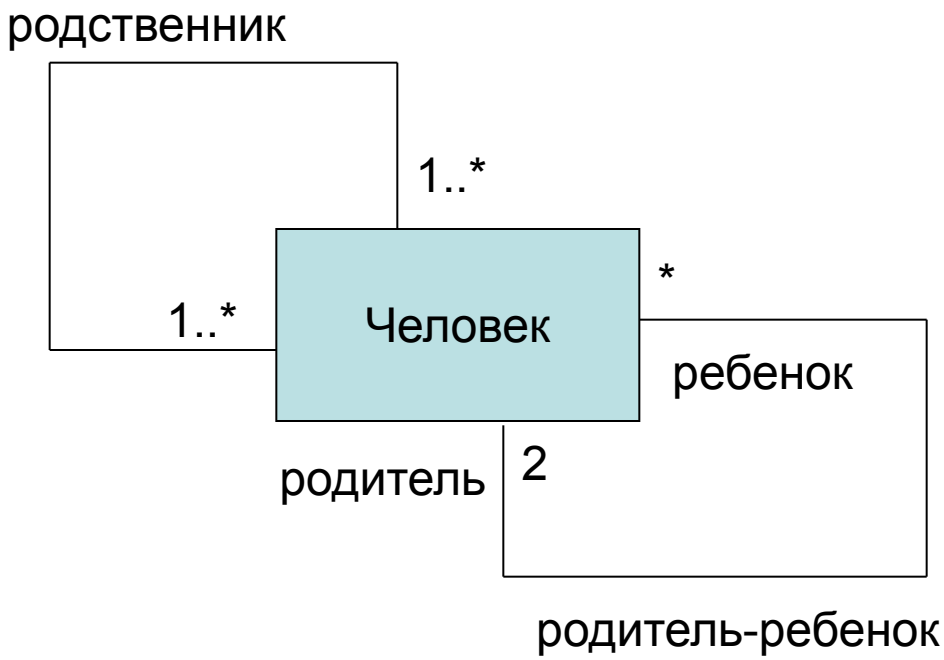
Навигация определяет возможность перехода от объектов одного класса к объектам другого класса, участвующим в ассоциации. Навигация изображается стрелкой. В данном случае навигация позволяет перейти от конкретного объекта Компания ко всем объектам Сотрудник, которые в этой компании работают. Обратный переход от Сотрудника к Компании, в которой этот сотрудник работает, невозможен.



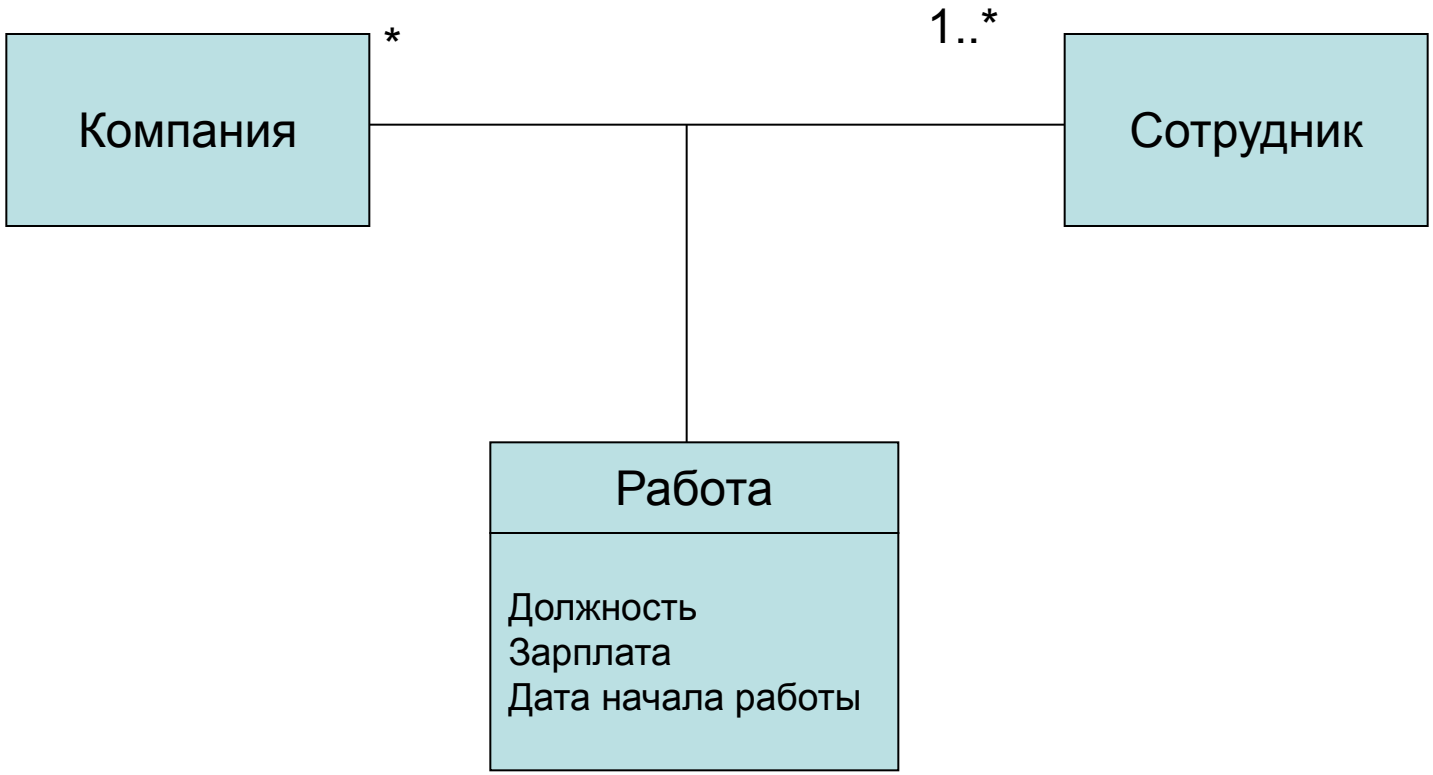
Кратность ассоциаций



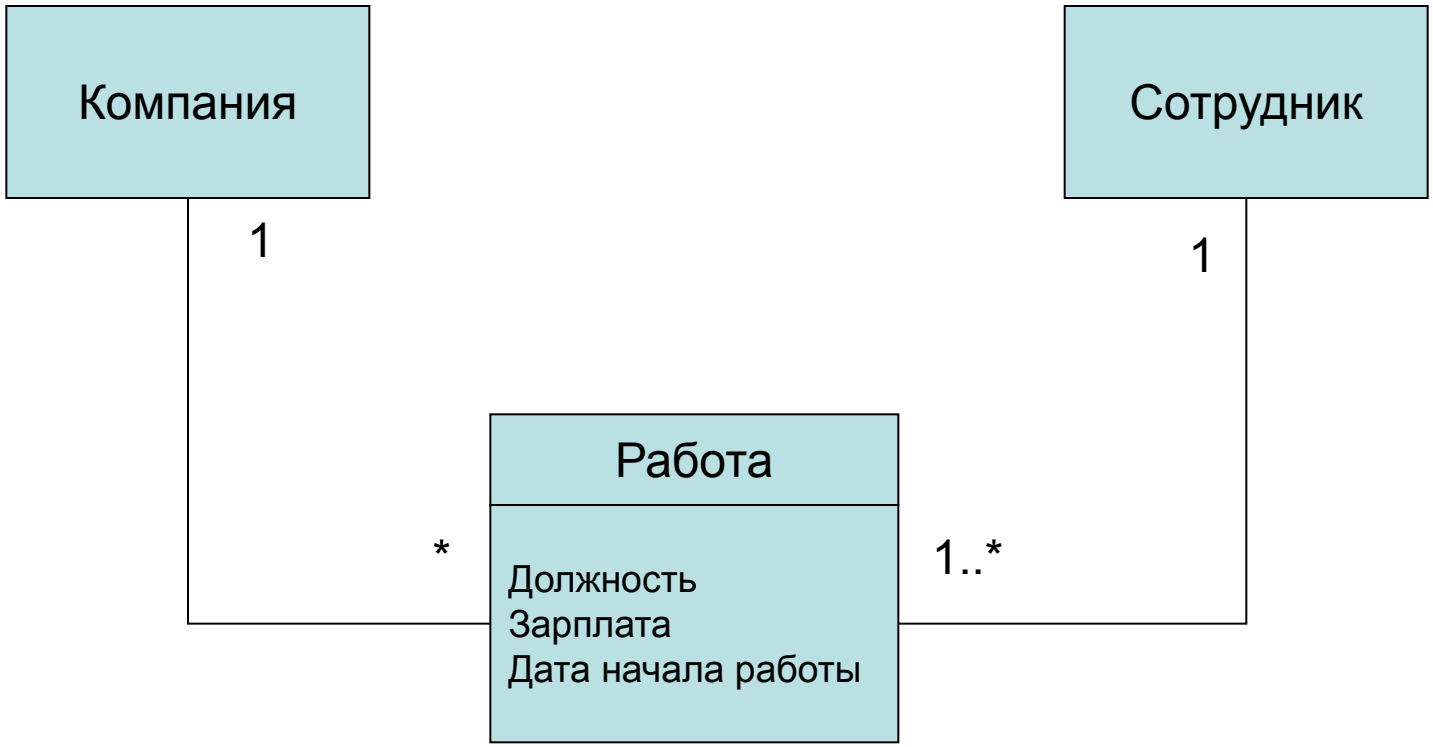
Кратность ассоциаций



Классы-ассоциации



Ассоциация “один-ко-многим”



Агрегирование

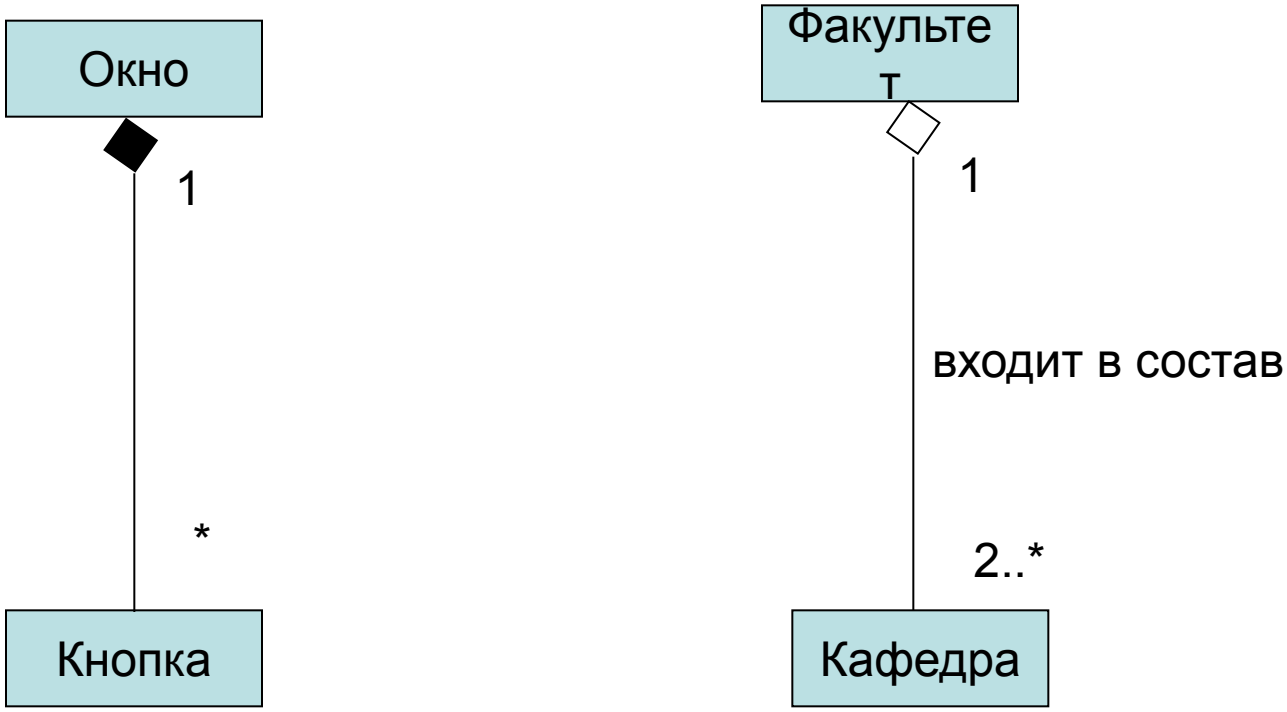
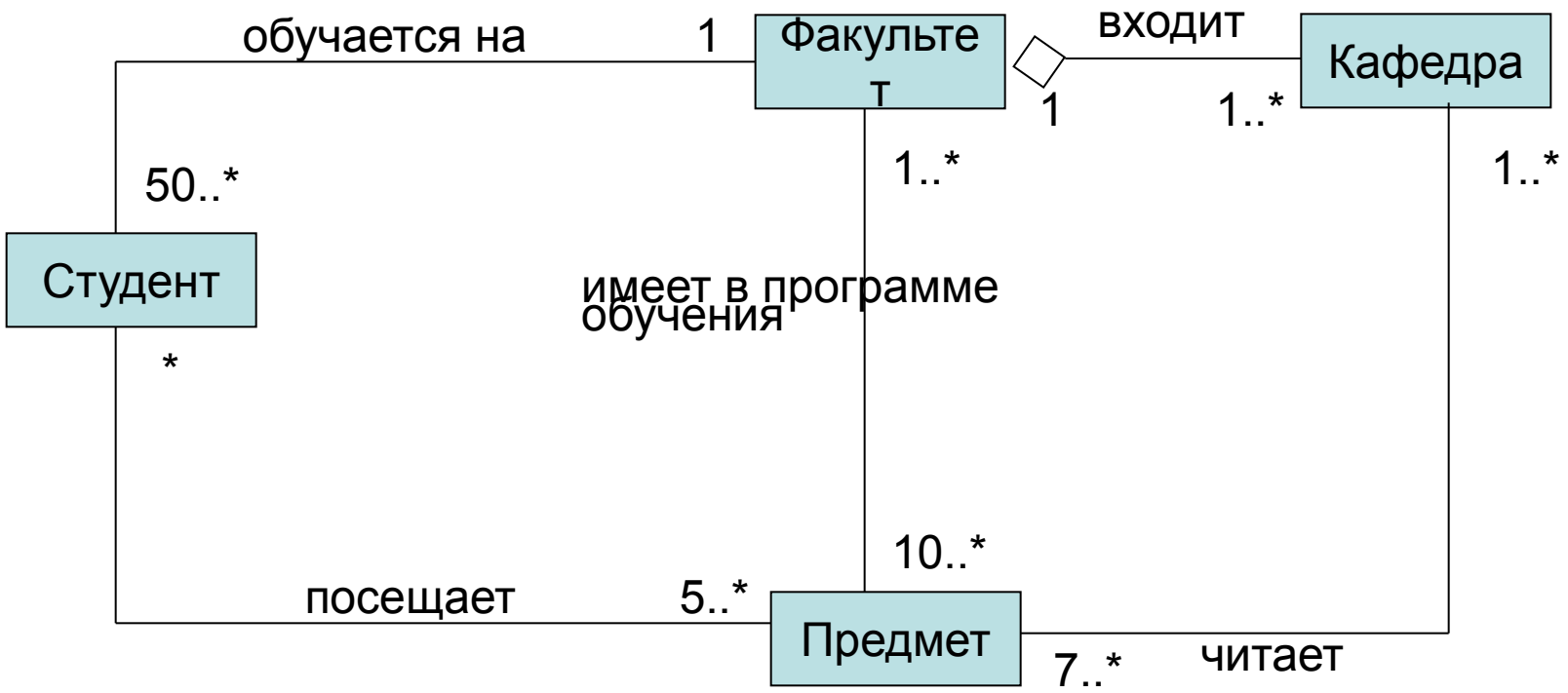


Диаграмма классов



Представление объектов

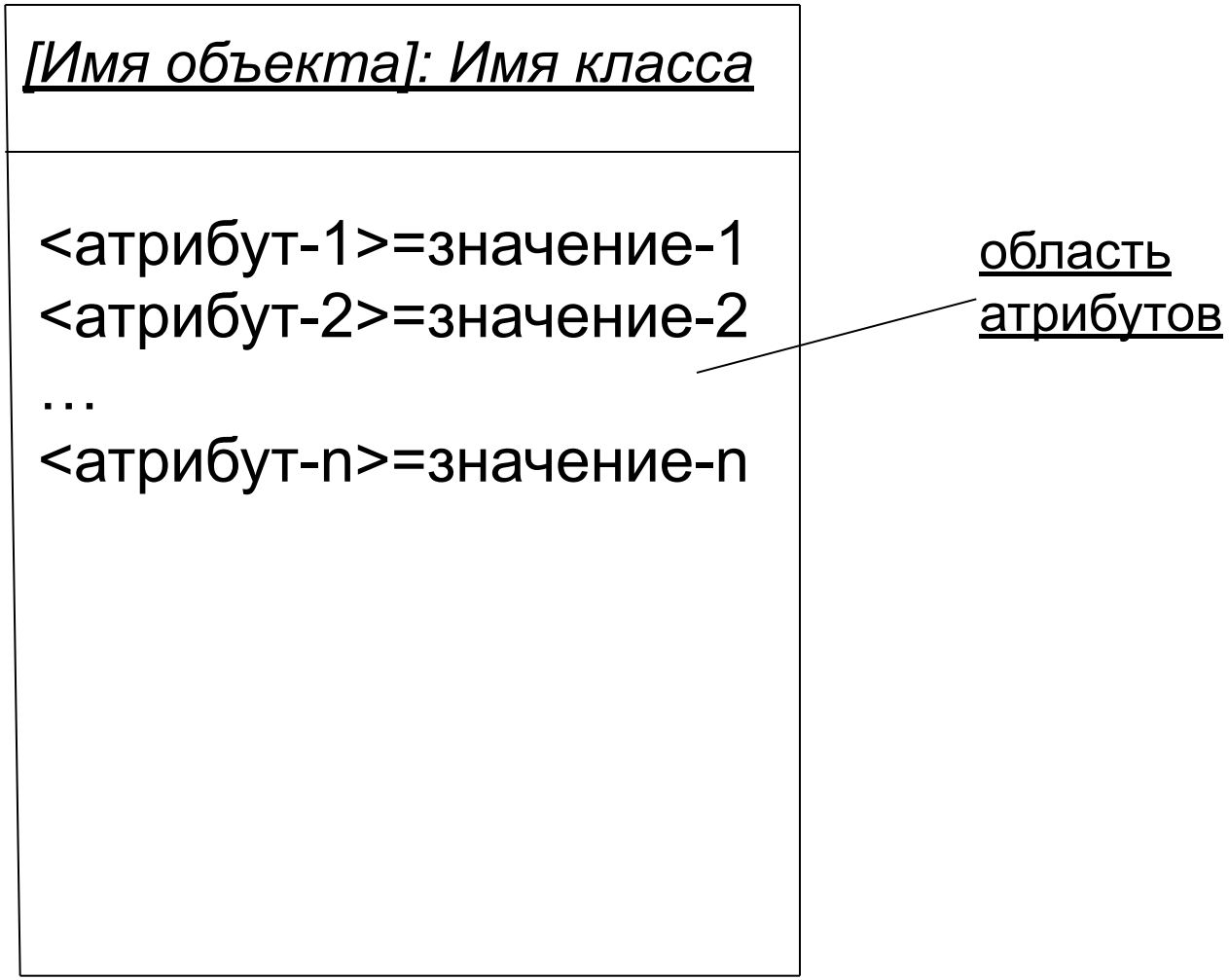
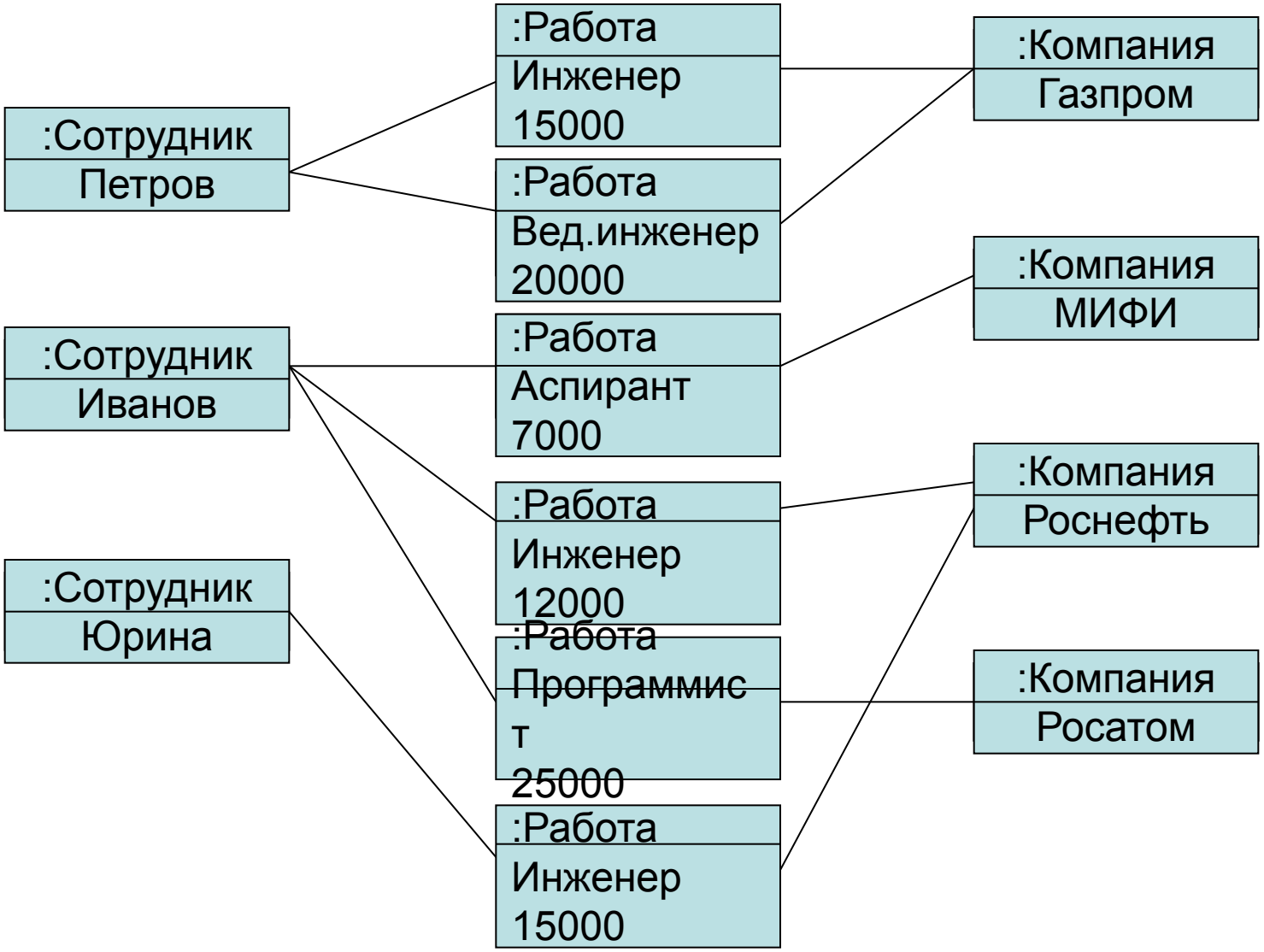


Диаграмма объектов



Объектно-ориентированное проектирование

1. Идентификация классов определенного уровня абстракции, соответствующего данной итерации.
2. Определение атрибутов и сигнатуры классов (шаблоны классов).
3. Определение отношений между классами (диаграммы классов).
4. Определение областей видимости элементов классов (шаблоны классов).
5. Планирование реализации базовых методов (диаграммы последовательностей).

Уровень абстракции – совокупность знаний о предметной области, используемых на данной стадии разработки, т.е. на данной итерации

CASE-средства

CASE (Computer Aided Software Engineering)-средства ориентированы на постоянное использование компьютера в процессе разработки ПО.

В большинстве CASE-средств применяются UML-диаграммы.

Наиболее известные CASE-средства – Rational Software Architect (IBM), Together (Borland), AllFusion (Computer Associates), TAU (Telelogic)

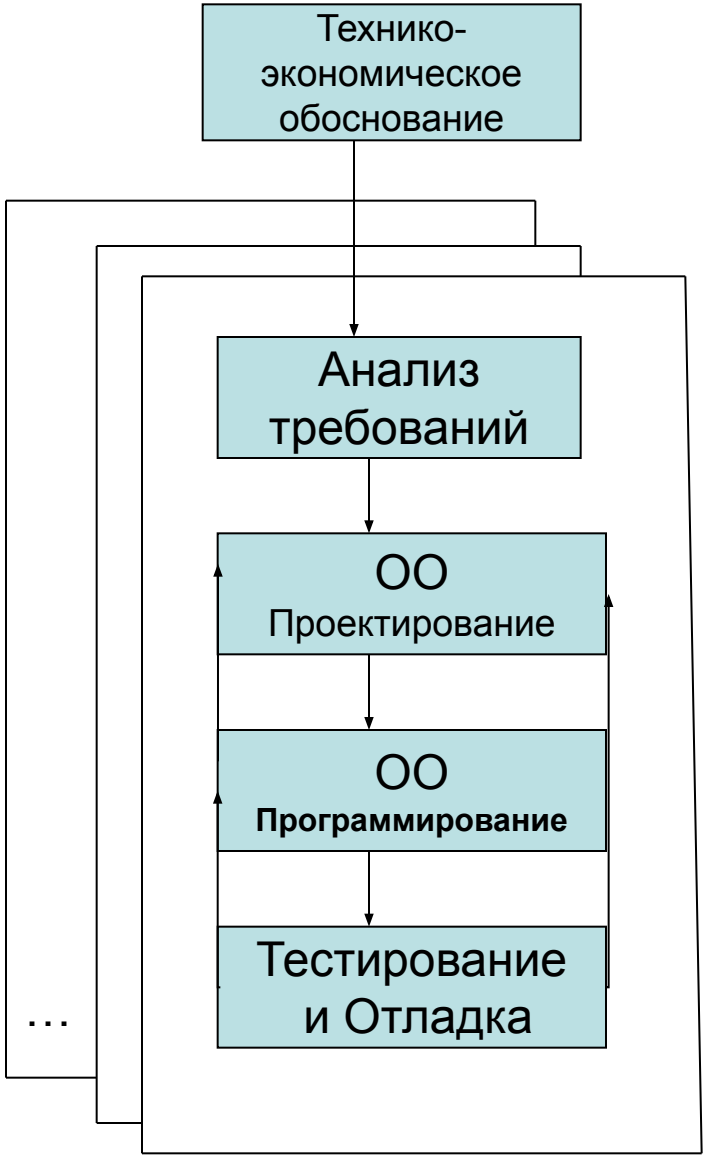
Поддержка UML-диаграмм встроена во многие системы программирования: Visual Studio, Eclipse, Delphi.

http://www.objectsbydesign.com/tools/umltools_byCompany.html

Цели использования CASE-средств:

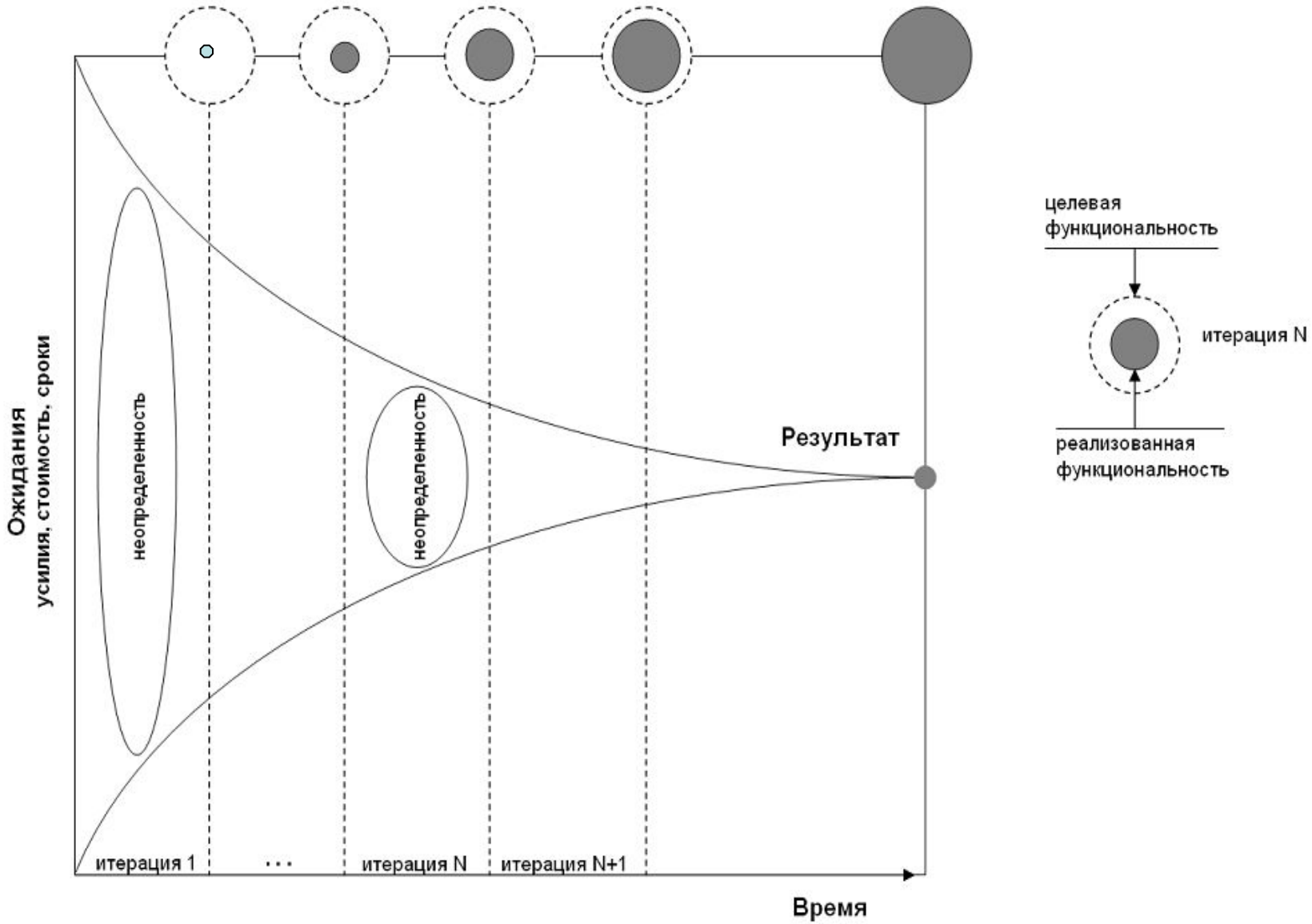
- построение UML-диаграмм;
- генерация кода по UML-диаграммам;
- генерация UML-диаграмм на основе кода.

ОО-технологии: этапы разработки

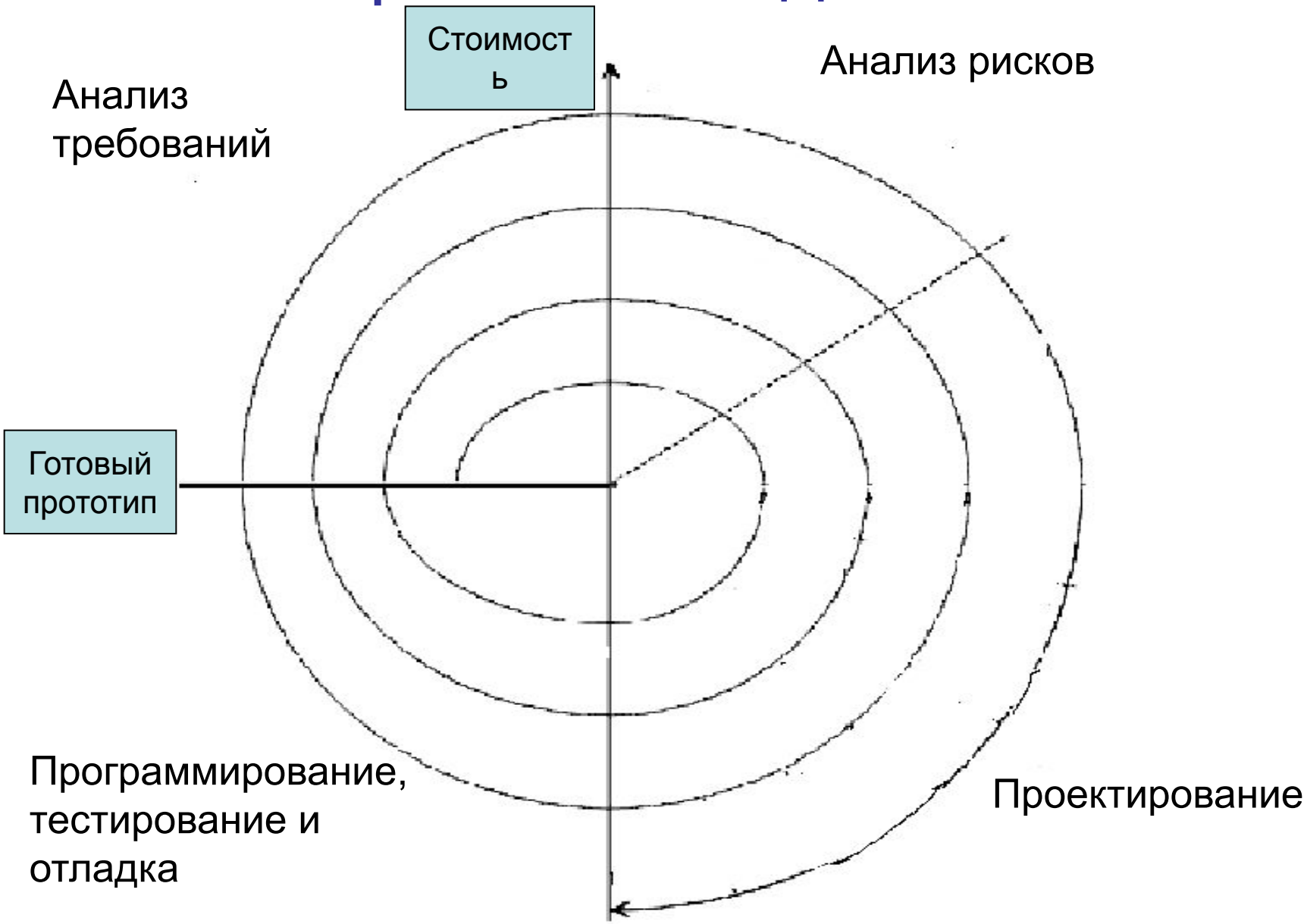


Эволюция начинается
со второй итерации

Итеративная разработка ПО



Спиральная модель Боэма



Достоинства ОО-технологий разработки ПО

1. Тесная связь с заказчиком в процессе разработки.
2. Возможность изменения требований к ПО.
3. Получение работающих версий до завершения разработки.
4. Повышенное внимание к объектам и структурам данных.
5. Возможность принятия альтернативных решений.
6. Детальная отработка элементов интерфейса.
7. Равномерное распределение разных видов работ в процессе создания программной системы.

Выявление классов

1. Формулирование предназначения класса в программной системе.

2. Класс – шаблон описания множества однотипных объектов. Классы, предусматривающие существование одного объекта, встречаются редко. Это, как правило, управляющие вспомогательные классы.

3. Класс должен содержать набор атрибутов.

Часто существуют один или несколько атрибутов, идентифицирующих объекты класса.

4. Класс должен отличаться от атрибута.

5. Класс должен содержать набор операций.

Спецификация обобщения

Отношение обобщения соединяет базовый класс с более специализированными классами. У специализированного класса – класса-потомка – имеются по отношению к базовому классу дополнительные атрибуты и/или операции.

Обобщение делает возможным повторное использование элементов базового класса.

Наличие абстрактных классов предопределяет использование отношения обобщения. Обычно абстрактные классы в качестве базовых специфицируются в процессе разработки позднее их потомков.

При поиске отношений обобщения ключевой вопрос формулируется так:

является один класс разновидностью второго или нет?

Спецификация ассоциаций

Ассоциация существует, когда объекты одного класса устойчиво связаны с объектами другого или других классов.

Спецификация ассоциаций включает:

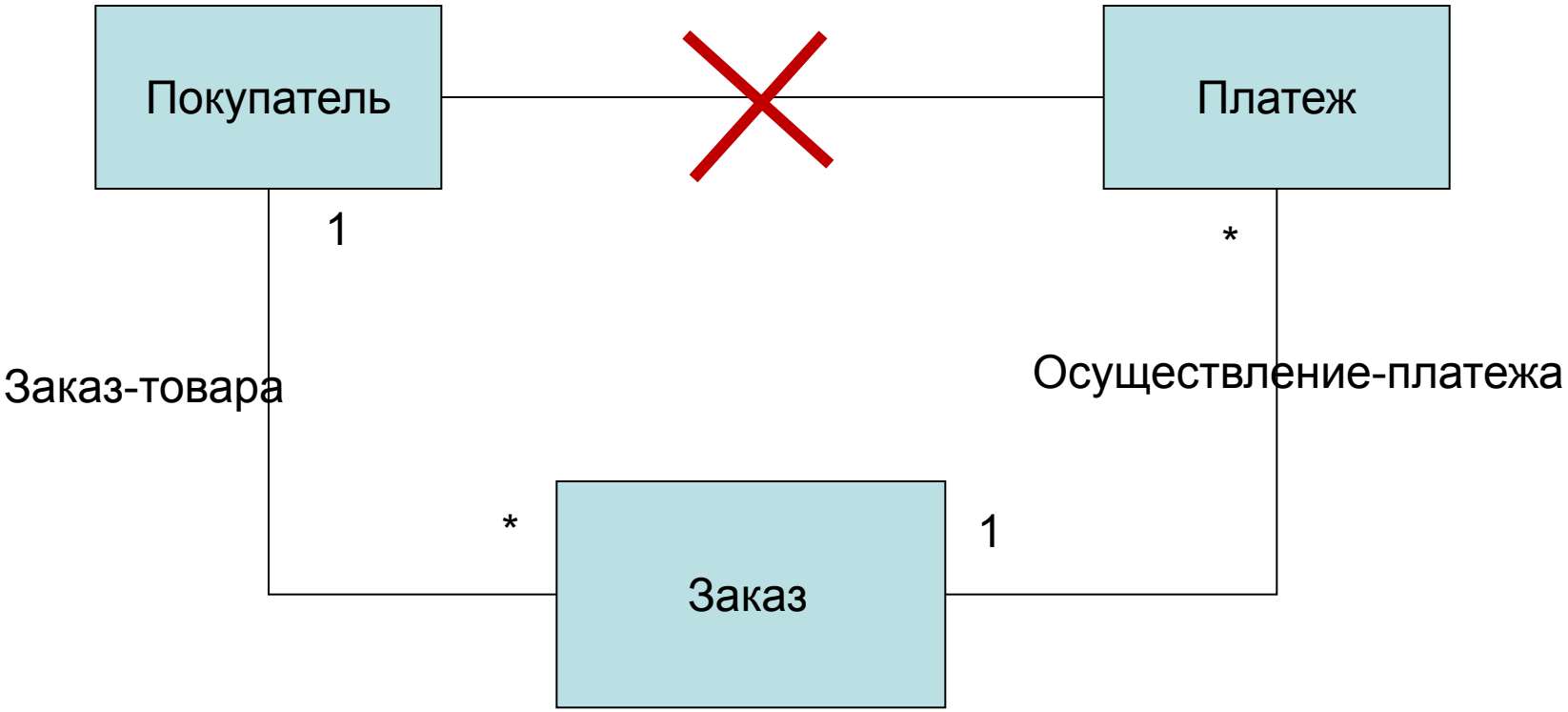
- задание имени ассоциации
- задание имен ролей ассоциации
- установление кратности ассоциации.

Кратности ассоциаций могут уточняться на начальных итерациях разработки.

Отметим, что CASE-средства часто автоматически генерируют имена ролей ассоциаций. Имена ролей бывают важны для ассоциаций, связывающих объекты одного класса.

Важно уметь выявлять избыточные ассоциации. Для этого надо анализировать циклы ассоциаций и отбрасывать лишние ассоциации.

Избыточные ассоциации



Спецификация агрегирования

Агрегирование – это отношение «часть - целое». Агрегирование – это особый случай ассоциации, обладающий дополнительной семантикой.

Так агрегирование обладает свойствами транзитивности и асимметричности.

Транзитивность: если объект класса A является частью объекта класса B, а объект класса B - частью объекта класса C, то объект класса A является частью объекта класса C.

Асимметричность: если объект класса A является частью объекта класса B, то объект класса B не является частью объекта класса A.

Композитное агрегирование обладает дополнительным свойством – зависимостью по существованию.

Агрегирование степени выше 2 бессмысленно.

Эволюция системы

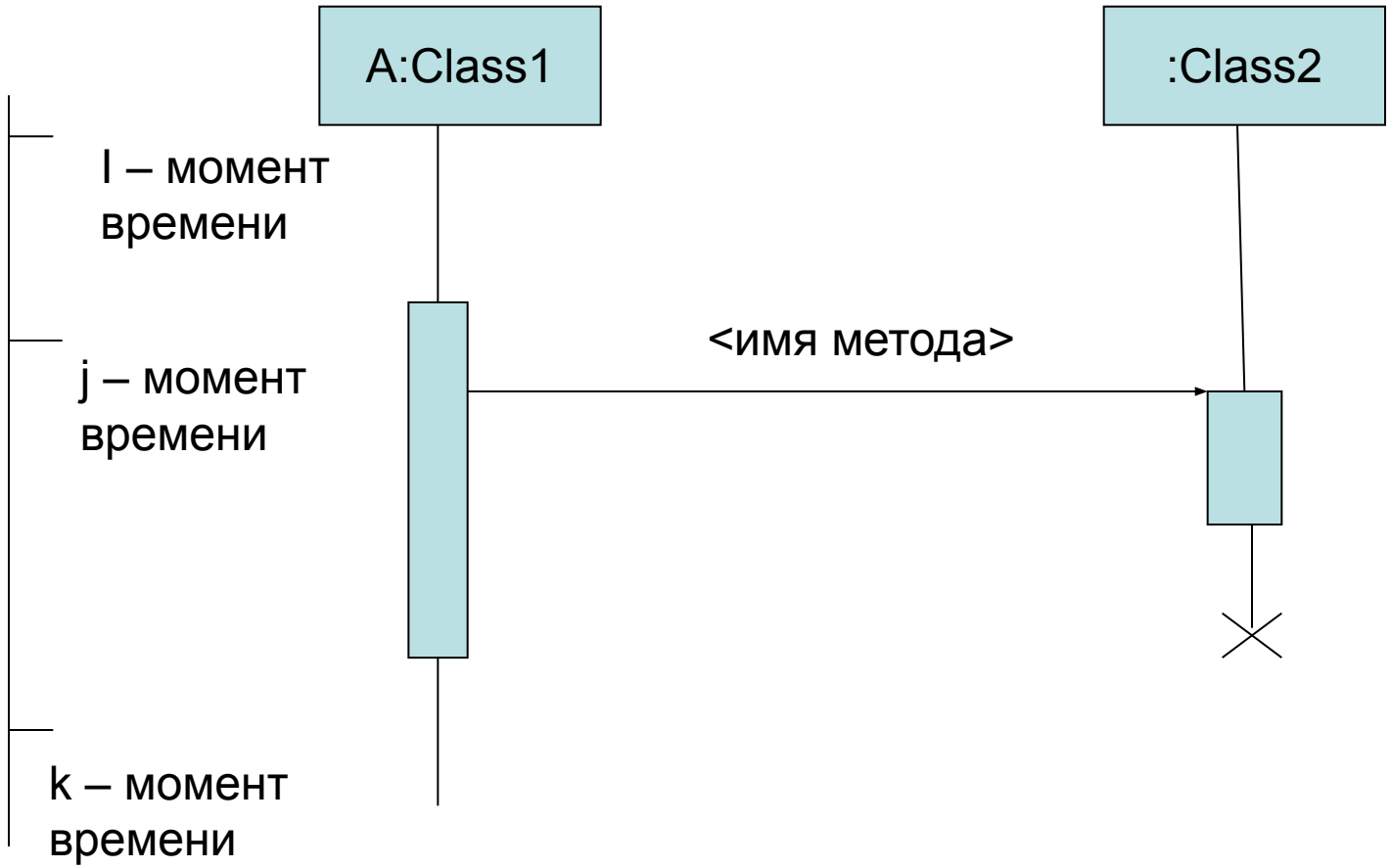
- добавление новых классов
- введение абстрактных классов
- разделение одного класса на ряд других
- изменение интерфейсов классов
- корректировка отношений между классами
- введение новых отношений между классами
- изменение логики работы базовых методов классов

Диаграммы взаимодействий

- Диаграммы последовательностей
- Диаграммы кооперации

Служат для описания динамики работы программной системы с точки зрения взаимодействия объектов в процессе работы системы.

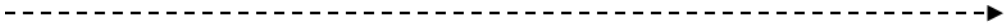
Диаграммы последовательностей



Условные обозначения

<имя метода>

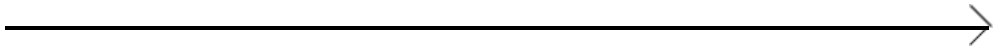
Синхронная передача управления



Возврат управления

Линия жизни

<имя метода>



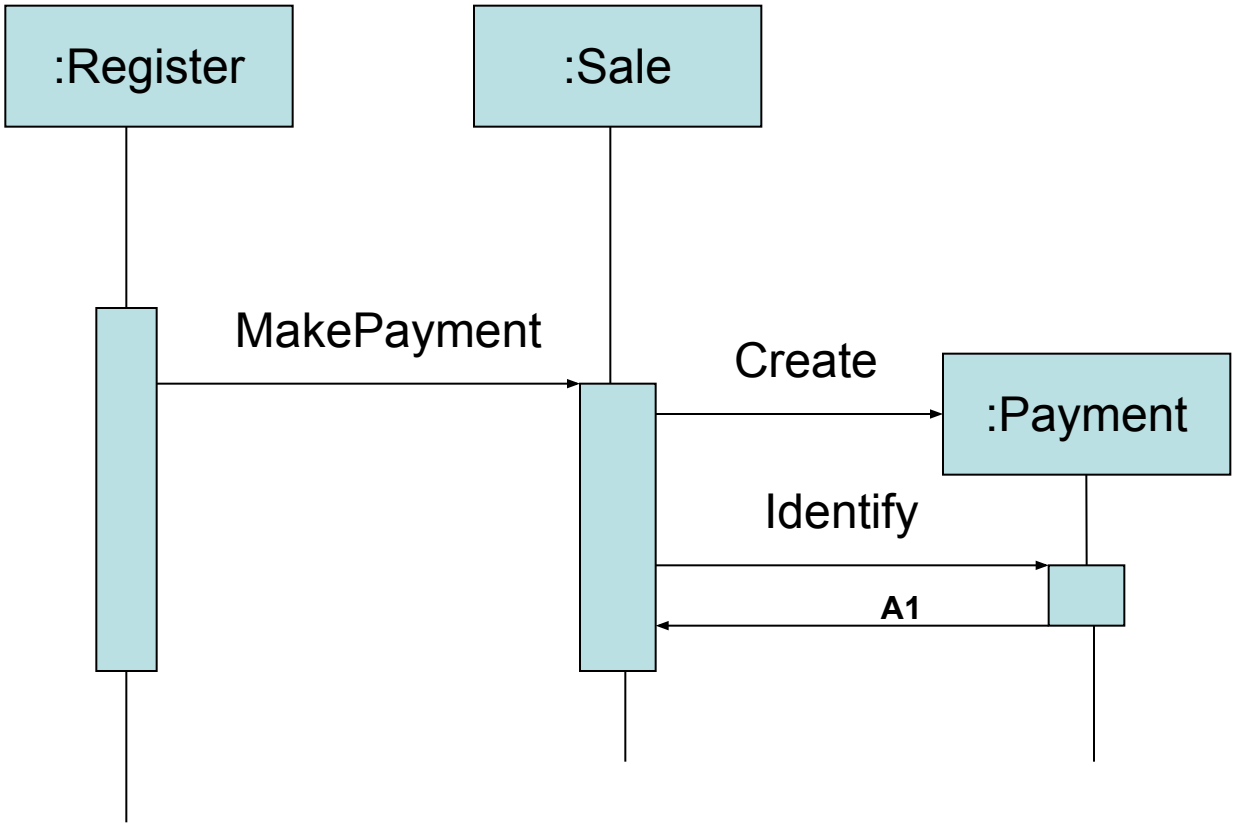
Асинхронная передача управления

Имя:Класс

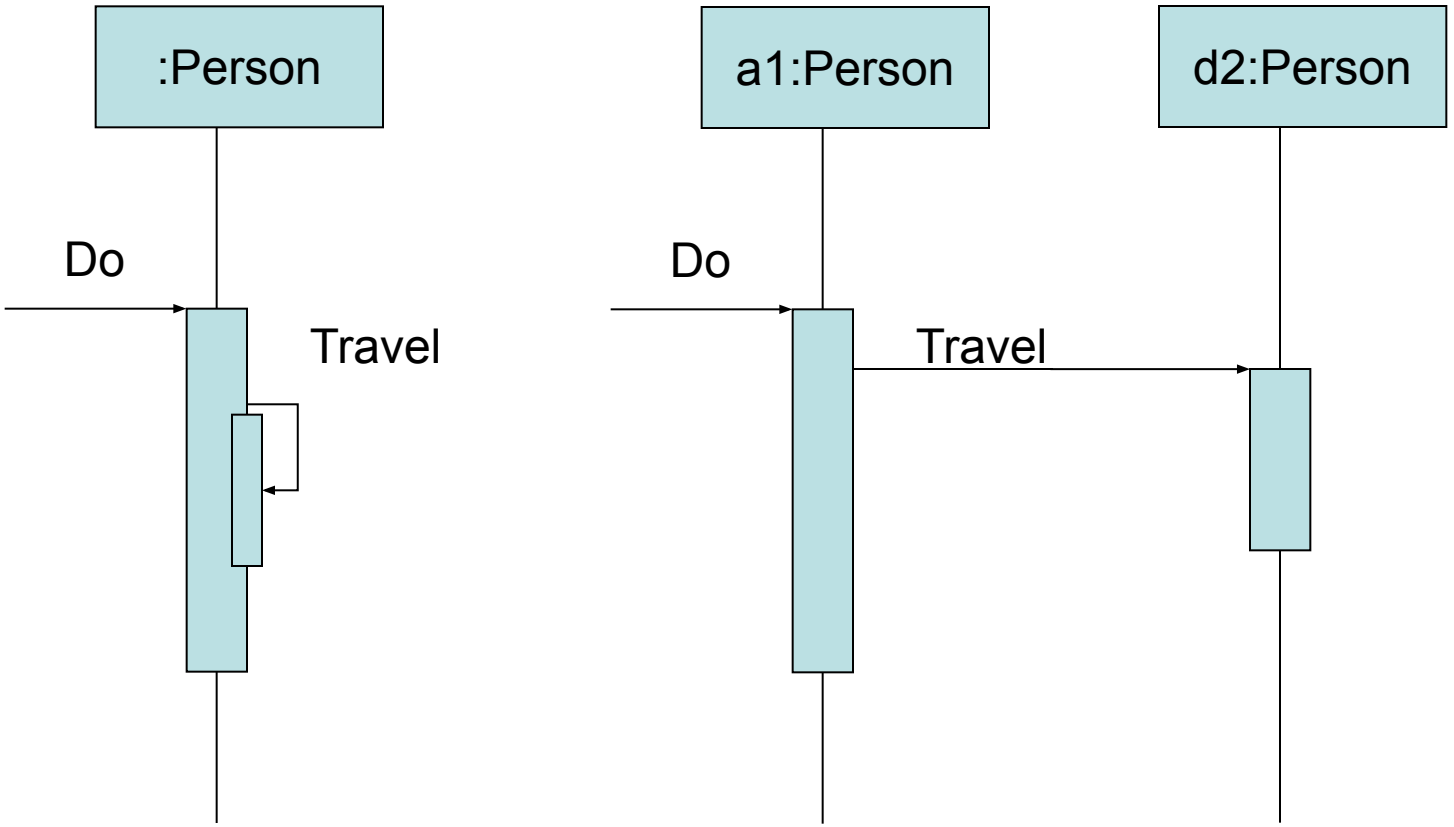
Объект

Фокус управления

Диаграммы последовательностей



Диаграммы последовательностей (рефлексивный вызов)



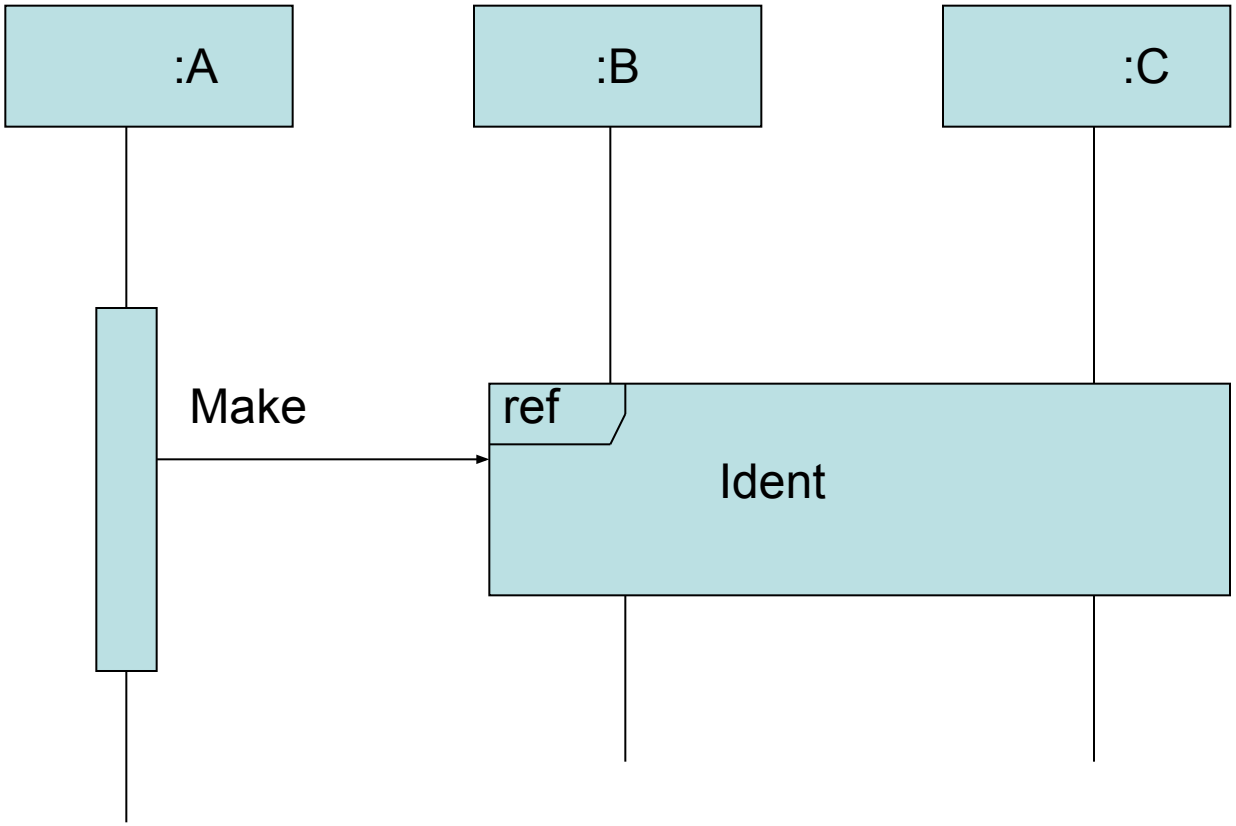
Фреймы

Фреймы используются для:

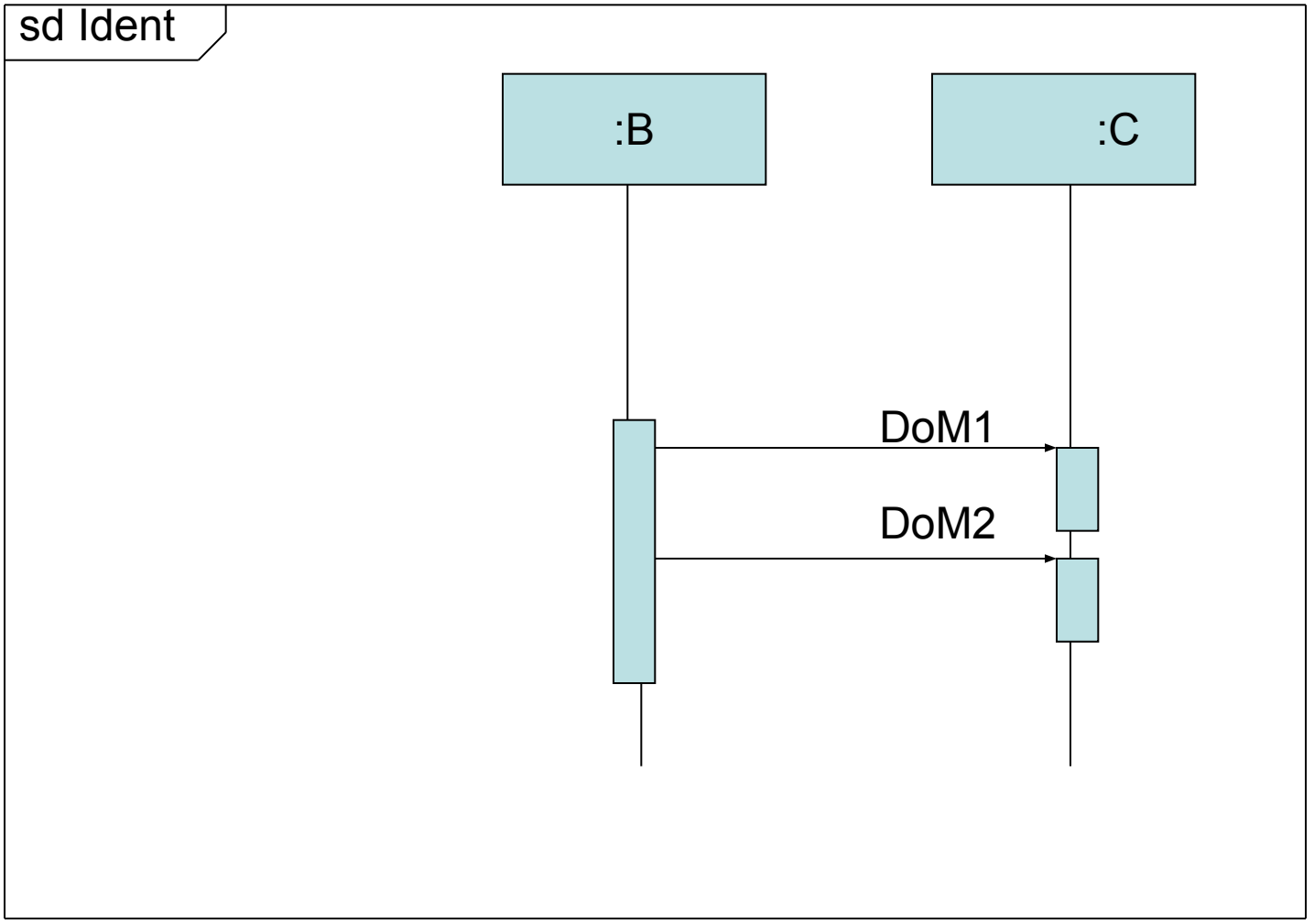
- Реализации вложенных диаграмм
- Представления ветвлений и условий
- Реализации циклов

Фреймы изображаются в виде прямоугольников с ярлычками (метками) в левом верхнем углу

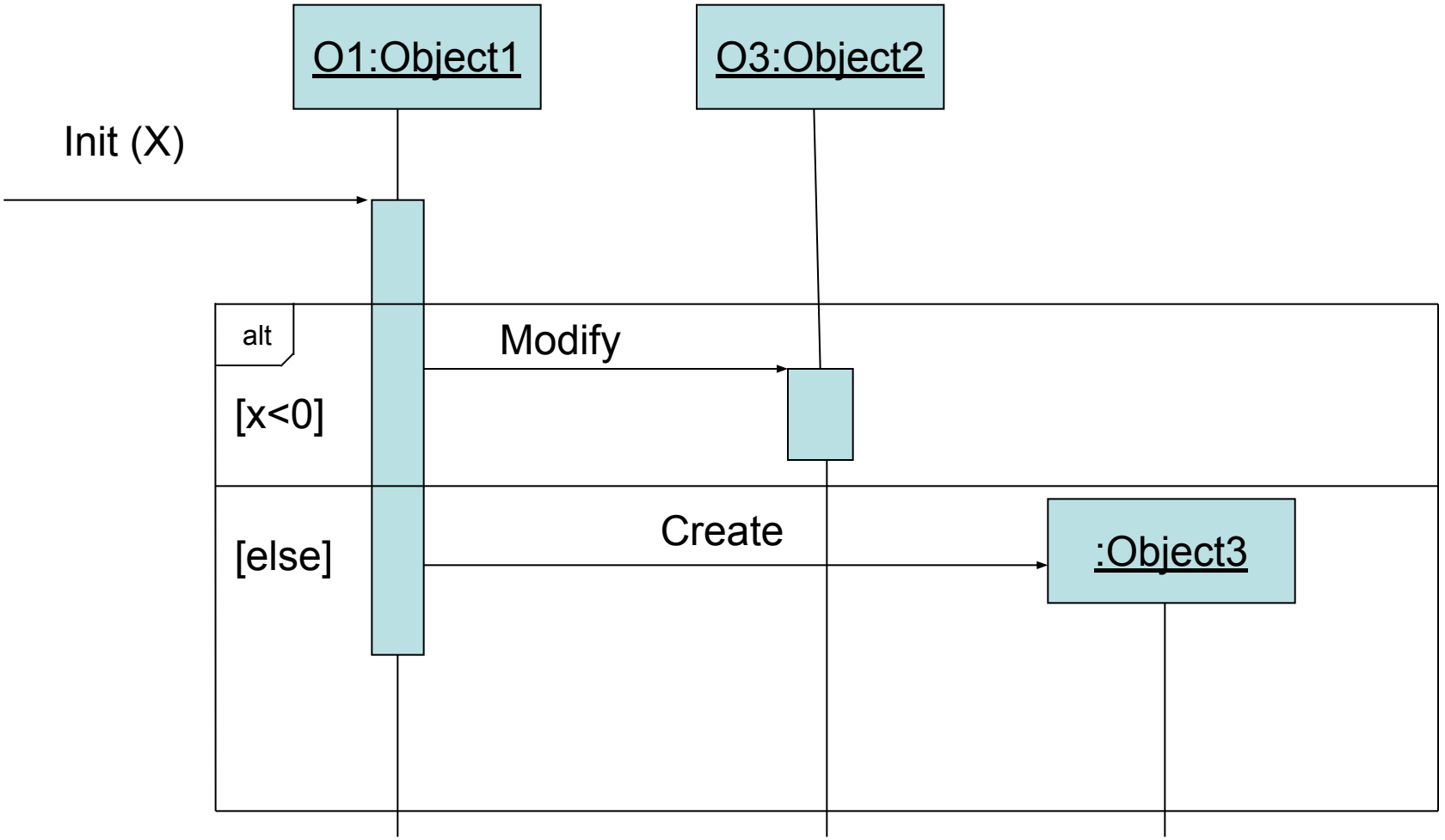
Диаграммы последовательностей (фреймы – связывание)



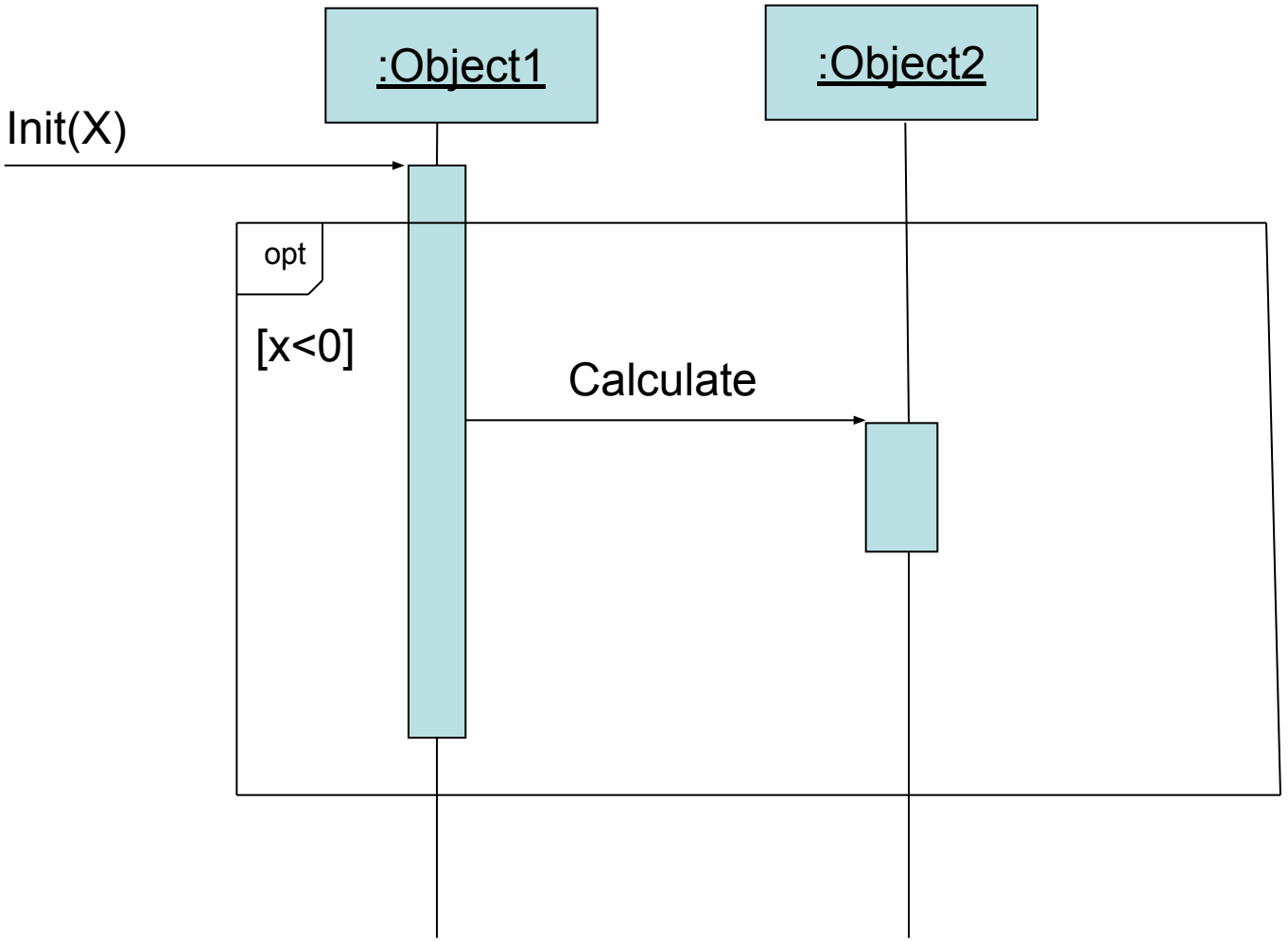
Диаграммы последовательностей (фреймы – связывание)



Диаграммы последовательностей (ветвление)



Диаграммы последовательностей (условие)



Диаграммы последовательностей (циклы)

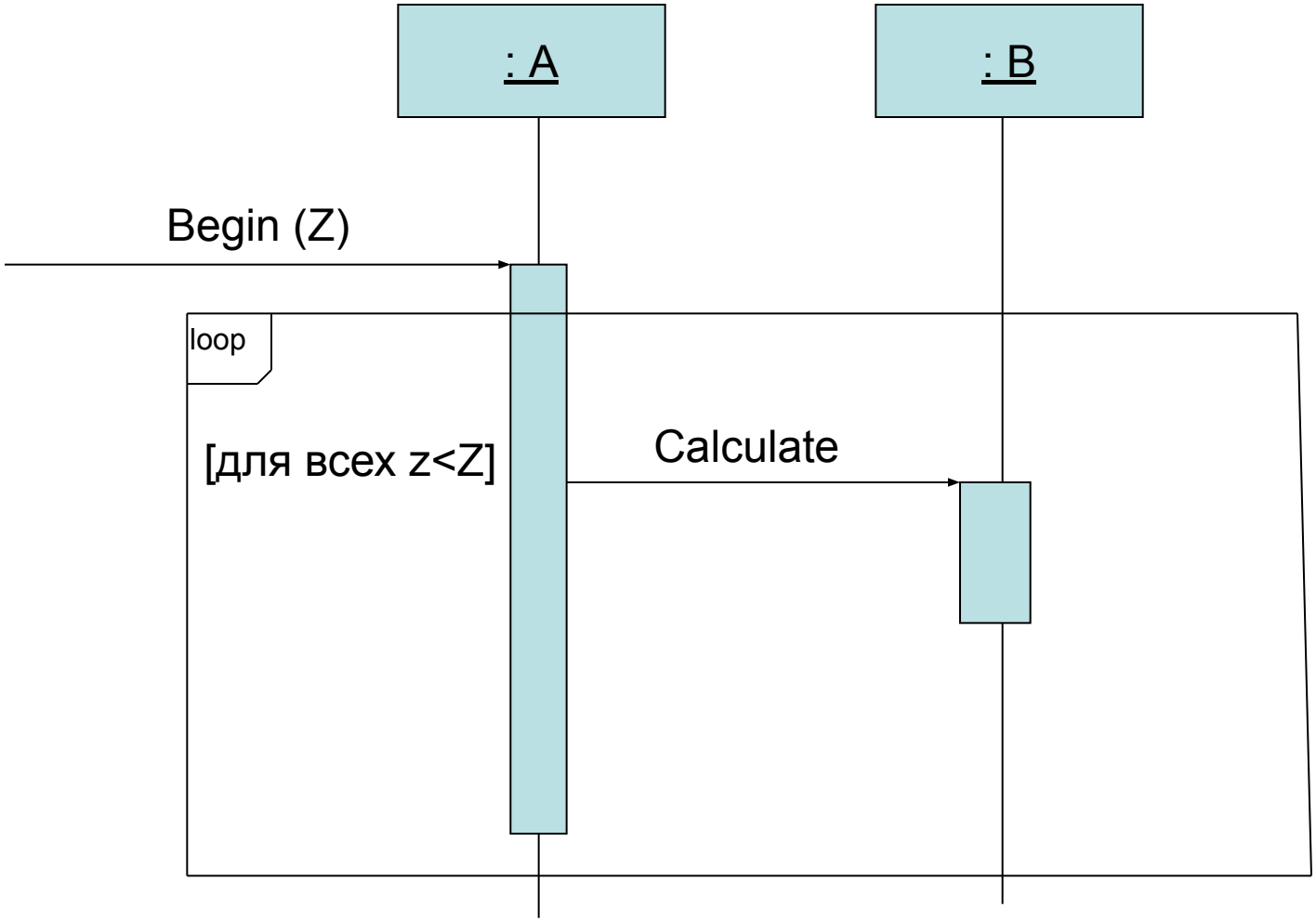


Диаграмма последовательностей с исполнителем

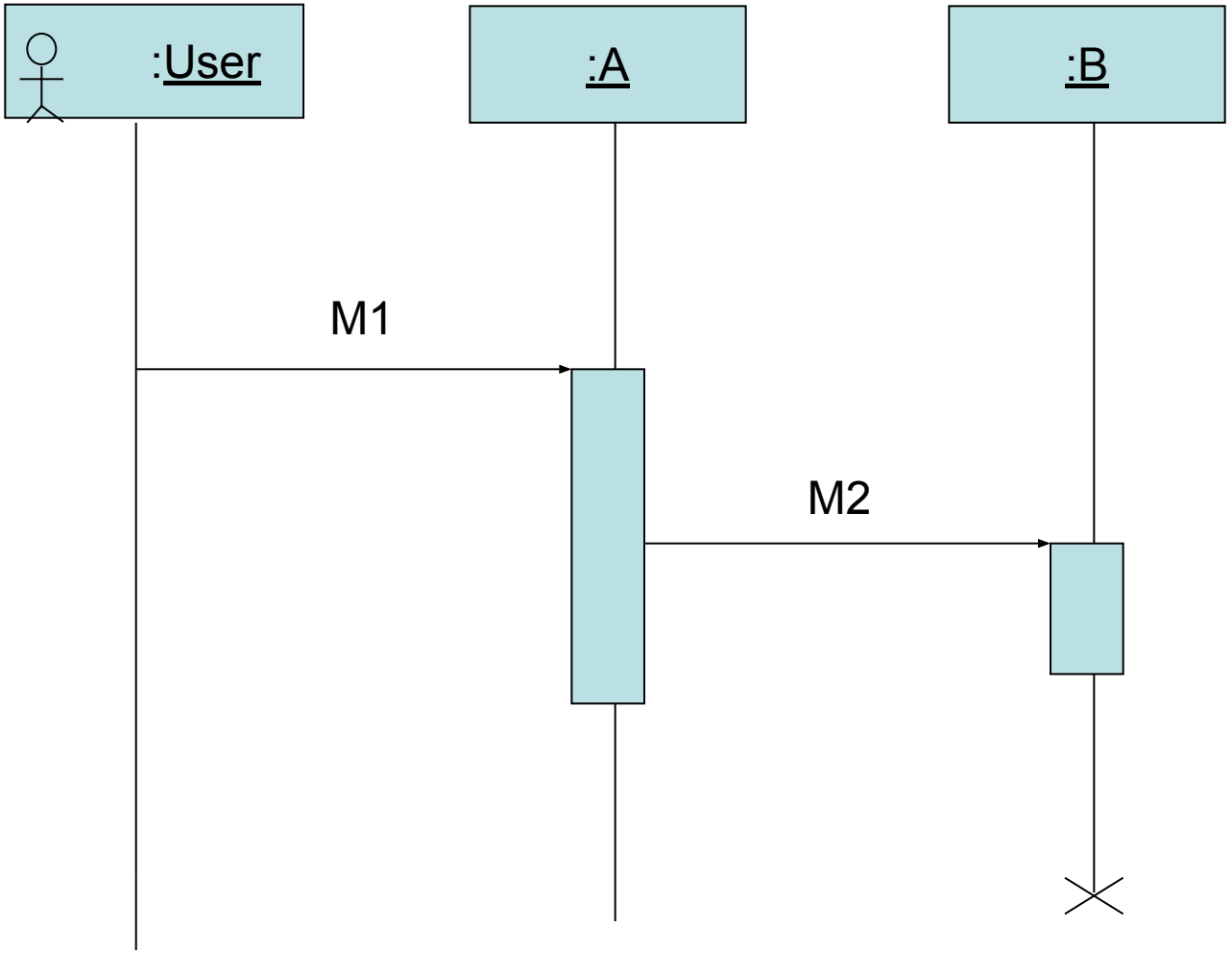


Диаграмма последовательностей “зачисление студента на курсы”

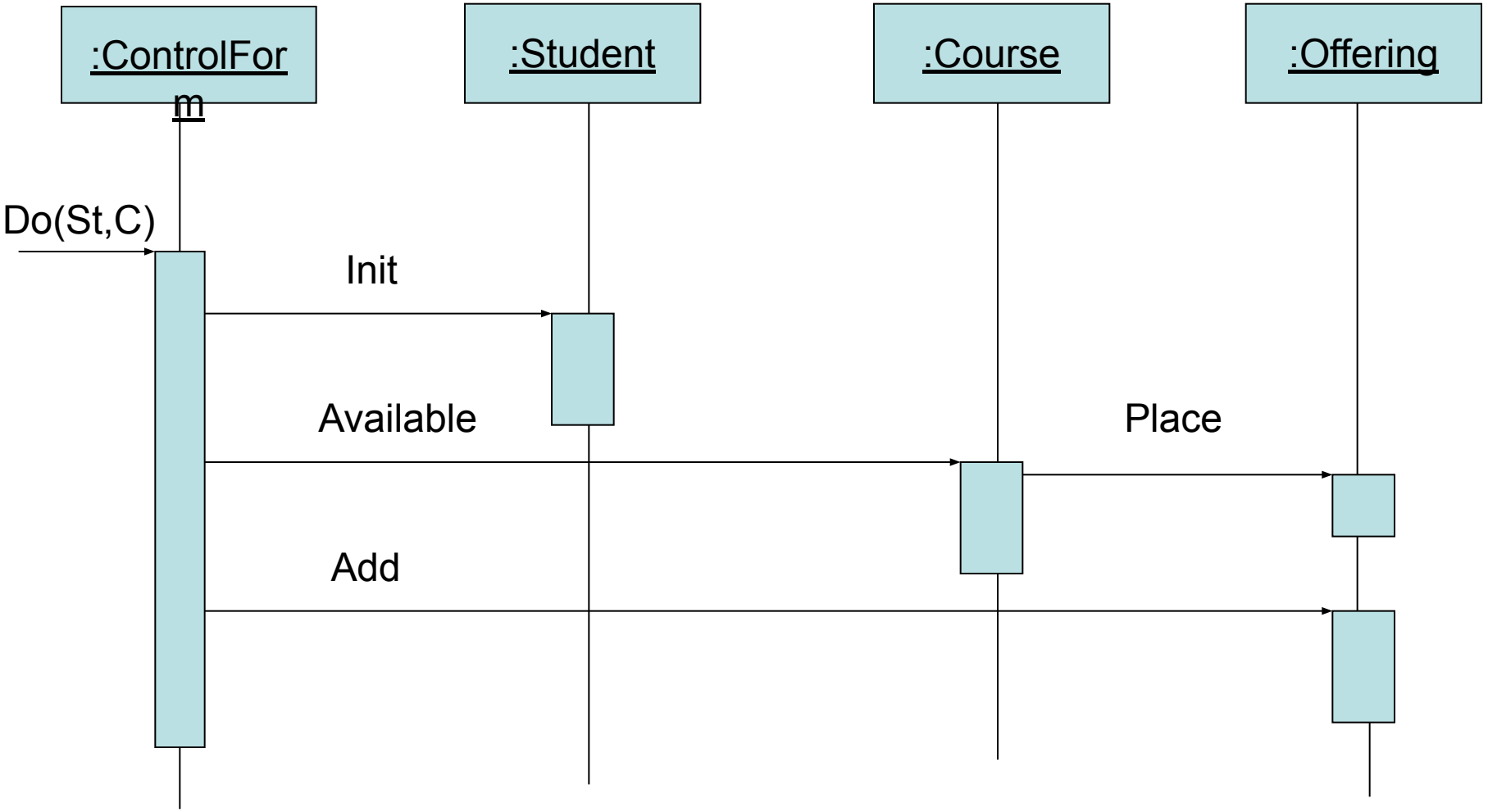
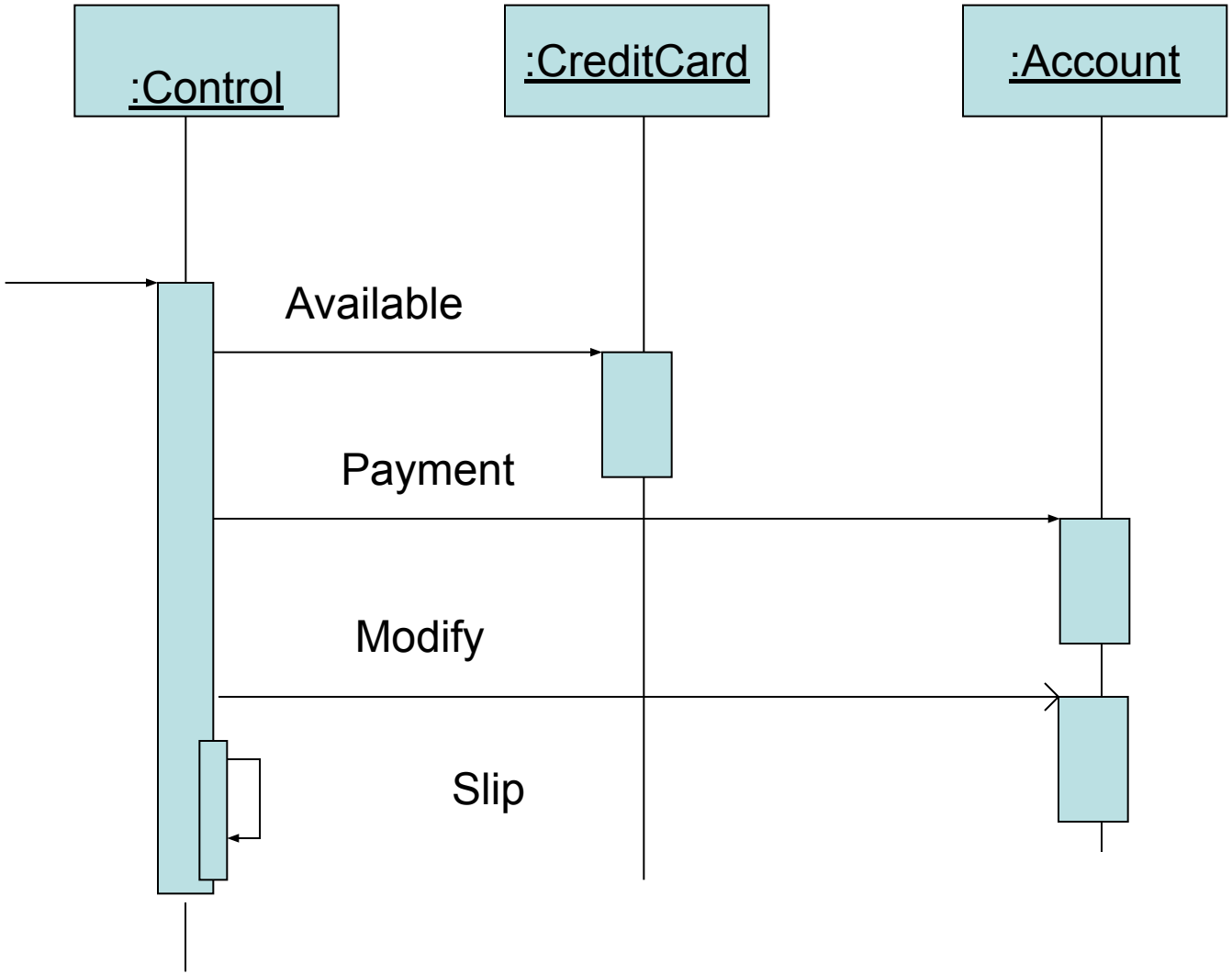


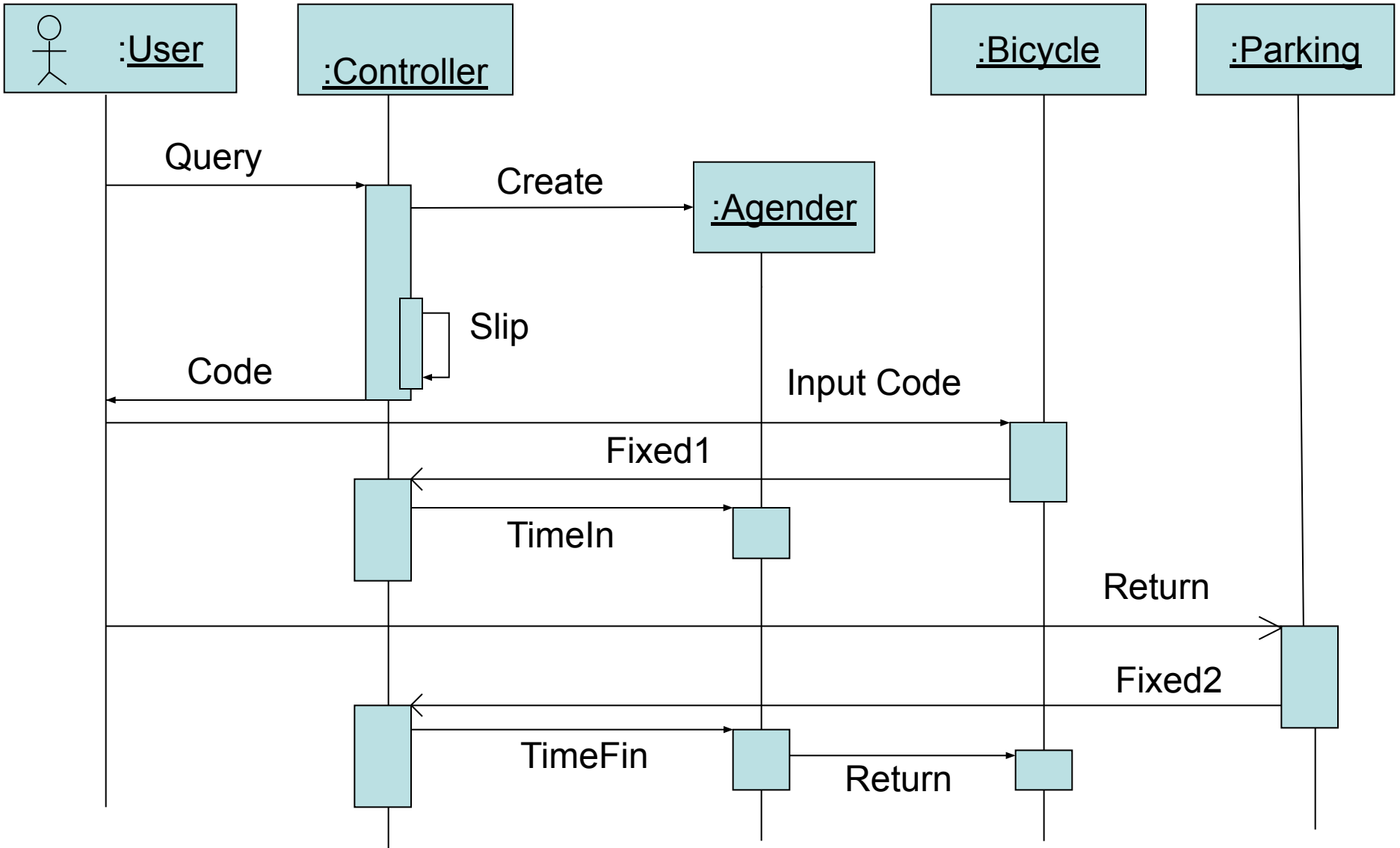
Диаграмма последовательностей “оплата услуги кредитной картой”



Прокат велосипедов

1. Оплата времени проката кредитной картой в паркомате.
2. Получение кода.
3. Ввод кода на клавиатуре стоянки велосипеда.
4. Получение велосипеда.
5. Катание.
6. Сдача велосипеда на стоянку.

Диаграмма последовательностей “прокат велосипедов”



Виды интерфейса

1. Командная строка (DOS, UNIX).

2. Текстовый интерфейс (оболочки для DOS, например, Norton Commander).

3. Графический интерфейс (Mac OS, Windows).

a. Оконный интерфейс.

b. Трехмерный интерфейс.

4. Альтернативные виды интерфейса (голосовой, жесты).

Проектирование графического интерфейса

Основные принципы разработки

- Управление со стороны пользователя
- Следование стандартам
- Возможность настройки
- Толерантность
- Обратная связь
- Удобство и эстетичность
- Простота

Удобство и эстетичность

Факторы:

- Цветовая гамма
- Симметрия
- Выравнивание
- Расстояния между элементами
- Пропорциональность
- Группирование связанных элементов
- Порядок расположения

Проектирование графического интерфейса

Элементы интерфейса

Главное окно и вторичные окна. Главное окно обычно содержит дочерние окна. Дочерние окна размещаются внутри главного и уничтожаются вместе с ним.

Вторичных окон может быть много. Вторичные окна не зависят от главного. Вторичные окна обычно являются модальными; они расширяют функциональность главного окна.

Содержимое главного окна как правило организовано в виде панелей. В панелях размещаются дочерние окна. Часто дочерние окна располагают своими собственными элементами управления.

Вид главного окна (Delphi)

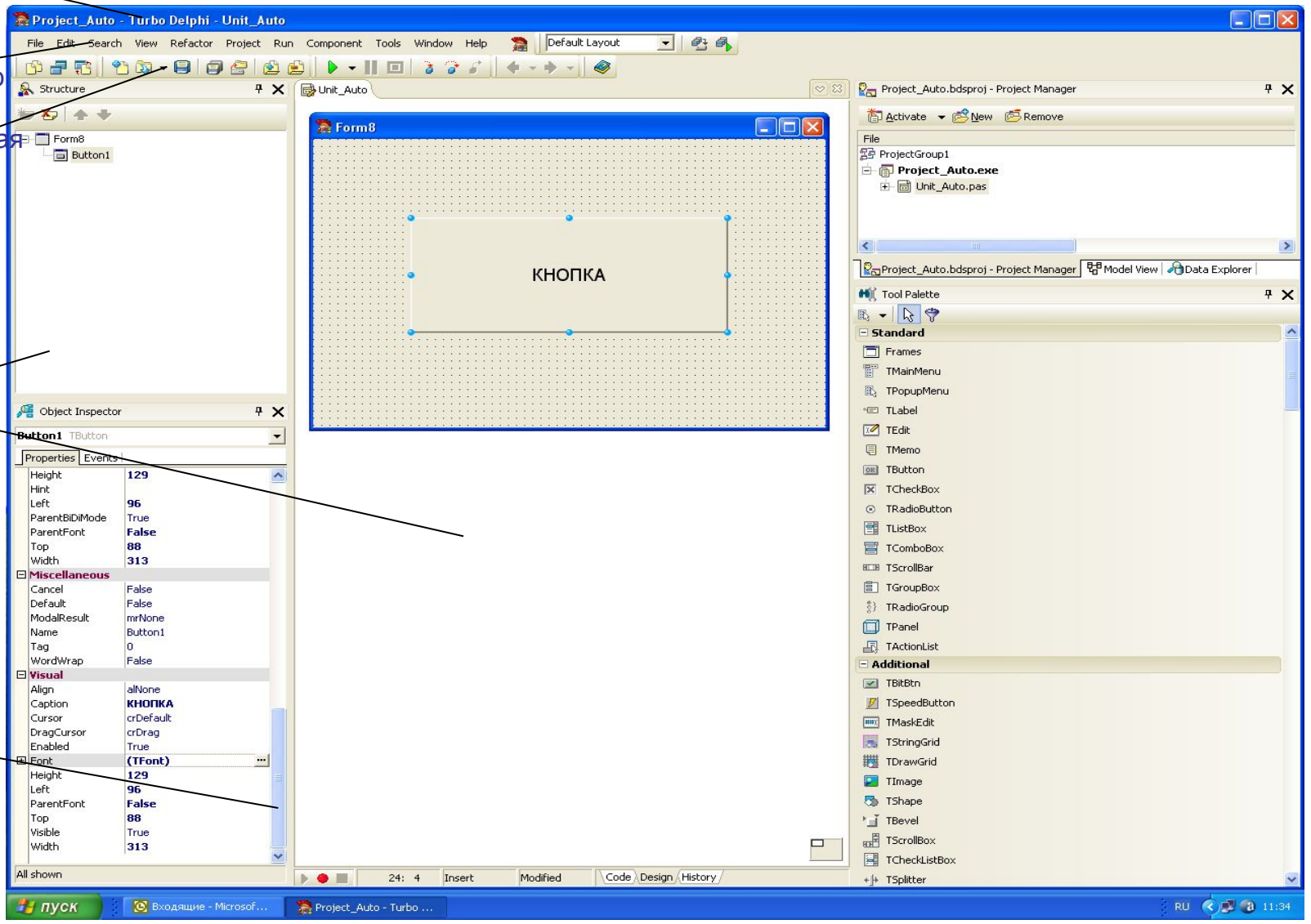
заголовок

главное меню

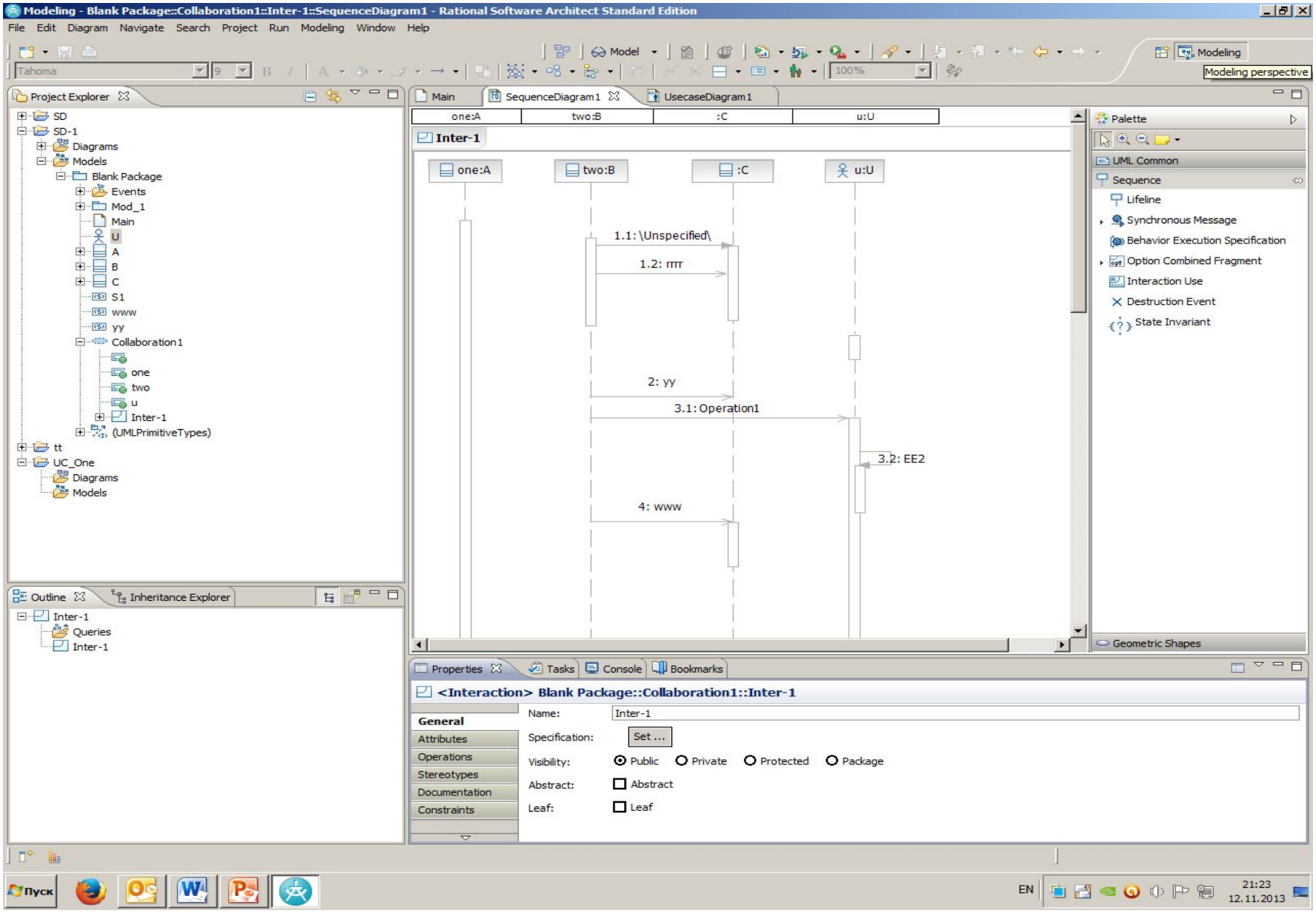
инструментальная линейка

панель

полоса прокрутки



Вид главного окна (Eclipse)



Вторичные окна

- Диалоговое окно

Требует ввода информации пользователем, обычно содержит строки или окна редактирования

- Папка с вкладками

Объединяет сразу несколько окон, инициируемых с помощью ярлыков

- Окно сообщений

Обычно требует только подтверждения или подтверждения/отказа. Всегда модально

Организация интерфейса в Eclipse

- Рабочее пространство workspace
- Рабочая среда workbench
- Перспектива perspective

Modeling, RAS, Java …

- Представление view

Project Explorer, Palette, Outline …

- Редактор editor

Разные редакторы для разных перспектив

Список вопросов

- Тенденции развития ИТ. Понятие программного обеспечения.
- Рынок ПО в России и в мире. Защита авторских прав разработчиков.
- Обобщенные критерии качества ПО.
- Элементарные критерии качества и метрики ПО.
- Жизненный цикл ПО.
- Технико-экономическое обоснование разработки и техническое задание.
- Функционально-ориентированная стратегия разработки ПО.
- Схема иерархии
- Характеристики модулей
- Объектно-ориентированная стратегия разработки ПО.
- Риски при разработке ПО.
- Диаграммы прецедентов.
- Сценарии.

Список вопросов

- Этап анализа требований.
- Отношения между классами: ассоциации.
- Классы-ассоциации.
- Отношение агрегирования.
- Диаграммы объектов.
- Этап объектно-ориентированного проектирования.
- Эволюция в процессе объектно-ориентированной разработки.
- CASE-средства.
- Сопоставление объектно-ориентированной и функционально-ориентированной стратегий разработки ПО.
- Диаграммы последовательностей.
- Фреймы в диаграммах последовательностей.
- Организация графического интерфейса.
- Интерфейс в Eclipse.