Министерство образования Республики Беларусь
Белорусский государственный университет
Механико-математический факультет
Кафедра веб-технологий и компьютерного моделирования

СОГЛАСОВАНО
Заведующий кафедрой
_____М.В. Игнатенко
«28» декабря 2023 г.

СОГЛАСОВАНО
Декан факультета
_____ С.М. Босяков
«30» января 2024 г.
МП

Анализ и проектирование информационных систем

Электронный учебно-методический комплекс для специальности:
1-31 03 08 «Математика и информационные технологии (по направлениям)». Направления специальности:
1-31 03 08-01 «Математика и информационные технологии (веб-программирование и интернет-технологии)»;
1-31 03 08-02 «Математика и информационные технологии (математическое и программное обеспечение мобильных устройств)»

Регистрационный № 2.4.2-24 / 550

Автор:
Блинов И. Н., кандидат физико-математических наук, доцент.

Рассмотрено и утверждено на заседании Научно-методического совета БГУ

28.11.2024 г., протокол № 4.

Минск 2024

УДК 004.415.2(075.8)
Б 695

Утверждено на заседании Научно-методического совета БГУ.

Протокол № 4 от 28.11.2024 г.

Решение о депонировании вынес:
Совет Механико-математического факультета.
Протокол № 6 от 30.01.2024 г.

Автор:
Блинов Игорь Николаевич, кандидат физико-математических наук наук, доцент, доцент кафедры веб-технологий и компьютерного моделирования механико-математического факультета БГУ

Рецензенты:
Лапицкая Н.В., доцент, зав. кафедрой Программного обеспечения информационных технологий факультета компьютерных систем и сетей УО «Белорусский государственный университет информатики и радиоэлектроники», кандидат технических наук,
Левчук В. Д., доцент кафедры Программного обеспечения информационных технологий факультета компьютерных систем и сетей УО «Белорусский государственный университет информатики и радиоэлектроники», кандидат технических наук

Блинов, И.Н. Анализ и проектирование информационных систем : электронный учебно-методический комплекс для специальности 1-31 03 08 «Математика и информационные технологии (по направлениям)», направления специальности 1-31 03 08-01 «Математика и информационные технологии (веб-программирование и интернет-технологии)», 1-31 03 08-02 «Математика и информационные технологии (математическое и программное обеспечение мобильных устройств)» / И.Н. Блинов ; БГУ, Фак. механико-математический, Каф. веб-технологий и компьютерного моделирования. – Минск: БГУ, 2024. – 96 с. – Библиогр.: с. 93-94.

Электронный учебно-методический комплекс (ЭУМК) по учебной дисциплине «Анализ и проектирование информационных систем» предназначен для студентов специальности 1–31 03 08 «Математика и информационные технологии (по направлениям)». Направления специальности: 1–31 03 08–01 «Математика и информационные технологии (веб-программирование и интернет-технологии)», 1–31 03 08-02 «Математика и информационные технологии (математическое и программное обеспечение мобильных устройств)». ЭУМК содержит тексты лекций, планы лабораторных занятий, перечень контрольных вопросов, списки рекомендованной литературы.

СОДЕРЖАНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Электронный учебно-методический комплекс (ЭУМК) по учебной дисциплине «Анализ и проектирование информационных систем» предназначен для специальности 1-31 03 08 Математика и информационные технологии (по направлениям). Направления специальности: 1-31 03 08-01 Математика и информационные технологии (веб-программирование и интернет-технологии) 1-31 03 08-02 Математика и информационные технологии (математическое и программное обеспечение мобильных устройств).

Комплекс подготовлен в соответствии с требованиями Положения об учебно-методическом комплексе на уровне высшего образования, утвержденного Постановлением министерства образования Республики Беларусь от 26.07.2011 № 167.

Содержание разделов ЭУМК соответствует образовательным стандартам, структуре и тематике типового учебного плана.

Главные цели ЭУМК: помощь студентам в организации самостоятельной работы, повышение качества подготовки и усиление практико-ориентированности учебного процесса по дисциплине «Анализ и проектирование информационных систем».

ЭУМК состоит из следующих разделов.

Теоретический. Включает аннотацию учебного пособия, написанного в соответствии с программой дисциплины. Материал данного пособия может быть использован для самостоятельной подготовки студентов к лабораторным занятиям, контрольным работам и зачёту в соответствиии с «Анализ и проектирование информационных систем». Учебная программа учреждения высшего образования по учебной дисциплине для специальности 1-31 03 08 Математика и информационные технологии (по направлениям). Направления специальности: 1-31 03 08-01 Математика и информационные технологии (веб-программирование и интернет-технологии) 1-31 03 08-02 Математика и информационные технологии (математическое и программное обеспечение мобильных устройств) № УД – 11151 уч. [Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/255226 – Дата доступа: 10.06.2024. И в соответствиии с «Анализ и проектирование информационных систем». Учебная программа учреждения высшего образования по учебной дисциплине для специальности первой ступени высшего образования 1-31 03 08 Математика и информационные технологии (по направлениям). Направления специальности: 1-31 03 08-01 Математика и информационные технологии (веб-программирование и интернет-технологии) 1–31 03 08-02 Математика и информационные технологии (математическое и программное обеспечение мобильных устройств). № УД-9959/уч. [Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/255226– Дата доступа: 10.06.2024.

Практический. Содержит планы лабораторных работ. Данные материалы используются для подготовки к лабораторным занятиям и их организации, для самостоятельной работы над курсом.

Раздел контроля знаний представлен вопросами к зачёту.

Описаны формы диагностики и технология определения оценки по дисциплине с учетом текущей успеваемости.

Вспомогательный раздел включает рекомендуемую литературу и содержание учебной программы курса по отдельным темам, на основе которой построено изучение дисциплины и контроль знаний, вопросы к зачёту.

Учебная дисциплина «Анализ и проектирование информационных систем» предназначена для студентов специальности 1-31 03 08 Математика и информационные технологии (по направлениям). Направления специальности: 1-31 03 08-01 Математика и информационные технологии (веб-программирование и интернет-технологии) 1-31 03 08-02 Математика и информационные технологии (математическое и программное обеспечение мобильных устройств). В учебной дисциплине изучаются теоретические основы баз данных, рассматриваются вопросы анализа и проектирования информационных систем. Изучается работа с программными средствами автоматизации проектирования информационных систем.

Изучение дисциплины должно способствовать формированию у студентов основ алгоритмического мышления и представления о современных подходах к программному решению научных и прикладных задач, позволяет применить знания, полученные при изучении проектирования информационных систем, для практического решения задач проектирования, программирования и использования информационных систем, возникающих в различных областях промышленности и науки.

Цели и задачи учебной дисциплины

Дисциплина «Анализ и проектирование информационных систем» имеет прикладную направленность.

Цель учебной дисциплины — формирование у студентов устойчивых теоретических знаний и практических навыков в области разработки и эксплуатации информационных систем, использования средств автоматизированного проектирования и программных продуктов, реализующих функционирование и управление информационных систем.

Задачи учебной дисциплины

- изучение теоретических основ UML и архитектур информационных систем;
- формирование у студентов навыков проектирования информационных систем;
- получение студентами практических навыков по проектированию информационных систем.

Место учебной дисциплины в системе подготовки специалиста с высшим образованием.

Учебная дисциплина «Анализ и проектирование информационных систем» относится к государственному компоненту.

Связи с другими учебными дисциплинами, включая учебные дисциплины компонента учреждения высшего образования, дисциплины специализации и др.

Учебная дисциплина «Анализ и проектирование информационных систем»

взаимосвязана с учебной дисциплиной «Технологии программирования».

Требования к компетенциям

Освоение учебной дисциплины «Анализ и проектирование информационных систем» должно обеспечить формирование следующих компетенций:

базовые профессиональные компетенции:

БПК-3. Применять современные компьютерные математические системы для проведения вычислительного (компьютерного) эксперимента.

БПК-6. Применять современные технологии и базовые конструкции языков программирования для реализации алгоритмических прикладных задач и разработки веб-проектов.

БПК-9. Применять инновационные информационные технологии и современные языки программирования.

специализированные компетенции***:***

– СК-1. Осуществлять анализ контекста и поставленной проблемы, аргументированно выбирать оптимальный способ ее решения, согласовывать частичные проекты решения в общую согласованную архитектуру, выполнять реализацию проекта с учетом накопленных и поступающих данных.

В результате изучения дисциплины студент должен:

знать:

- основные модели UML;
- способы и CASE-средства проектирования информационных систем;
- язык проектирования UML;

уметь:

- проектировать логическую и физическую модели информационных систем;
- проводить построение диаграмм;
- использовать язык UML для создания проекта.

владеть:

- навыками практического проектирования и использования языка UML.

Структура учебной дисциплины

Учебная дисциплина «Анализ и проектирование информационных систем» изучается в 4 семестре дневной формы получения высшего образования, в семестре заочной формы получения высшего образования для направления специальности 1-31 03 08 Математика и информационные технологии (по направлениям). Направления специальности: 1-31 03 08-01 Математика и информационные технологии (веб-программирование и интернет-технологии) 1-31 03 08-02 Математика и информационные технологии (математическое и программное обеспечение мобильных устройств):

— для очной формы получения высшего образования– 102 часа, в том числе – 68 аудиторных часов, из них: лекции – 34 часа, лабораторные занятия – 30 часов, управляемая самостоятельная работа – 4 часа.

– на заочной форме получения высшего образования – 16 аудиторных часов,

из них: лекции – 8 часов, лабораторные занятия – 8 часов.

Трудоемкость учебной дисциплины составляет 3 зачетных единицы.

Форма текущей аттестации – зачет в 4-м семестре дневной формы и в 6 семестре заочной формы получения высшего образования.

1. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

В теоретическом разделе изложен краткий конспект лекций в соответствии с программой дисциплины.

Данное пособие ориентировано на студентов механико-математического факультета БГУ, изучающих программирование и проектирование.

В первой части пособия (1.1.) рассмотрены теоретические основы реляционных баз данных и основы их проектирования.

Вторая часть пособия (1.2.) посвящена изучению UML (Унифицированный Язык Моделирования).

1.1. Рациональный унифицированный процесс

Создаваемое программное обеспечение (ПО) постоянно усложняется. При работе над любыми информационными (распределенными) системами в первую очередь возникает проблема взаимопонимания программиста и заказчика уже на стадии обсуждения структуры системы. До начала кодирования программы предлагаемая концепция предусматривает решение двух предварительных задач: проанализировать поставленную перед разработчиком задачу и разработать проект будущей системы. Программа разбивается на отдельные модули, взаимодействующие между собой с помощью механизмов передачи параметров.

Рациональный унифицированный процесс (RUP) – методологическая основа объектно-ориентированного проектирования, представляет собой итеративный процесс создания и сопровождения объектно-ориентированных моделей разного уровня опирается на варианты использования системы.

1.1.1. Жизненный цикл RUP

В процессе создания распределенной системы рассматриваются следующие этапы:

1. *Стратегическое планирование и анализ* – определение требований, предъявляемых пользователем к создаваемой системе:
   - целесообразность разработки и сравнение с аналогами;
   - необходимость ресурсов для реализации проекта;
   - формулировка общих требований к функциональности системы;
   - оценка корректности и реализуемости требований.
2. *Проектирование* – разработка программных решений, удовлетворяющих требованиям пользователя, которые можно разбить на следующие фазы проектирования:
   - определение общей архитектуры или состава подсистем;
   - анализ прецедентов;
   - спецификация подсистем и проектирование их компонентов;
   - проектирование интерфейса взаимодействия каждой подсистемы с другими;
   - проектирование базы данных;

- проектирование алгоритмов обработки данных;
- проверка и оценка результатов.

3. *Кодирование* – инкрементное создание и тестирование кода.
4. *Тестирование* – проверка кода на соответствие требованиям.
5. *Управление изменениями и их контроль* – обеспечение поддержки совершенствование системы при эксплуатации.

Как аналог этапов разработки можно рассматривать их фазы:

1. *Начало* (Inception) – определение бизнес-целей проекта, т.е. требований, предъявляемых пользователем к создаваемой системе.
2. *Исследование* (Уточнение)(Elaboration) – разработка плана и архитектуры, конкретизация требований к системе, разработка программных решений, удовлетворяющих требованиям пользователя.
3. *Разработка* (Конструирование)(Construction) – создание системы в виде, в котором она может быть предоставлена пользователю.
4. *Внедрение* (Переход)(Transition) – контроль созданного программного обеспечения на соответствие предъявляемым требованиям, поставка системы конечному пользователю.

Последовательная и спиральная модели проектирования

Традиционный подход состоит в последовательном выполнении этапов создания ПО. В настоящее время применяется спиральная модель проектирования, использующая циклическое повторение предыдущих этапов при совершенствовании проекта и создании кода.

При спиральной модели проектирования следует начинать с анализа и выявления важнейшего элемента будущей системы, связанного с риском нарушения сроков разработки. Для этой части системы создается собственный проект, разрабатывается программное обеспечение и выполняется тестирование. Затем цикл повторяется, как это представлено на рисунке 1.

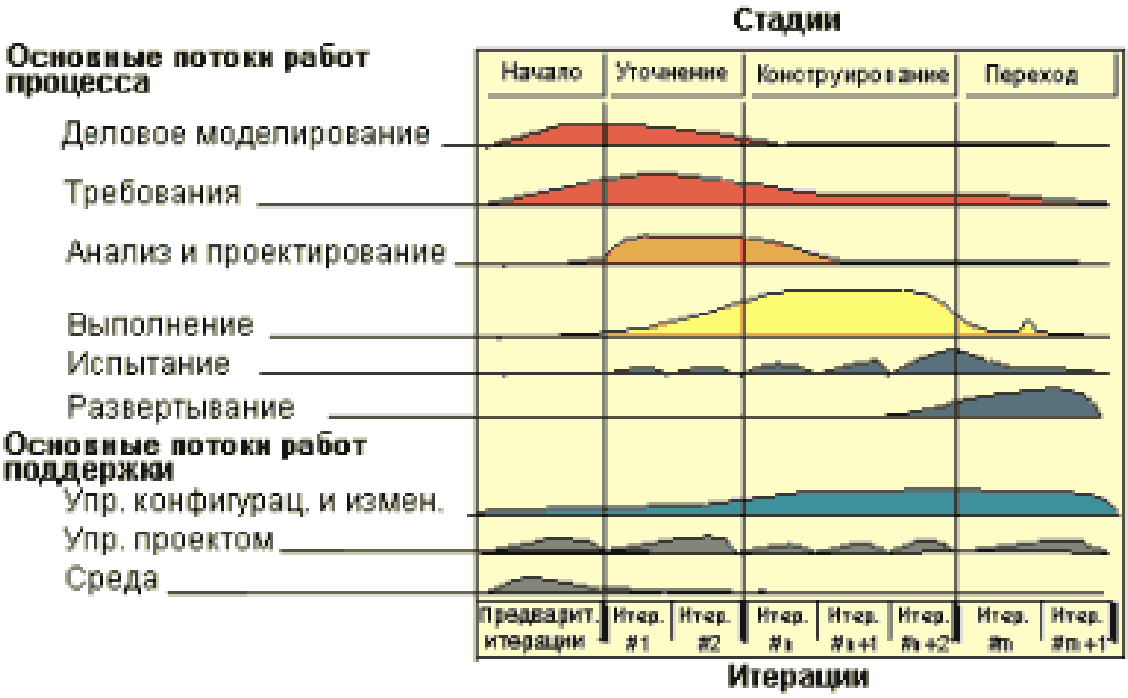


Рисунок 1 – Схема RUP по стадиям работы над проектом.

RUP предлагает:

- выпускать программное обеспечение, пользуясь принципом промышленного подхода, определяя стадии, потоки, уточняя обязанности каждого участника проекта;
- использовать итеративную разработку вместо каскадной для доведения компонента до безошибочного состояния;
- обязательное управление требованиями, так как по ходу разработки в систему вносятся изменения либо самой группой разработчиков, либо заказчиком;
- унифицированный документооборот, приведенный в соответствие со всеми известными стандартами.

RUP поставляется не как обычный программный продукт, а как он-лайн документация, оформленная в виде Web-страницы, что позволяет размещать его на внутренней сети предприятия для широкого использования сотрудниками.

В начальной фазе проекта пишется экономическое обоснование и определяются границы проекта. Именно в этой фазе спонсор проекта принимает на себя определенные обязательства и дает Вам согласие на дальнейшую работу. В фазе исследования (уточнения) требования выясняются более детально, выполняются высокоуровневый анализ и проектирование для построения базовой архитектуры и создается план для конструирования.

Проекты отличаются друг от друга количеством соблюдаемых формальностей. Сильно формализованные проекты включают множество официальных отчетов и согласований. В слабо формализованных проектах

начальная фаза может включать часовую беседу с заказчиком построение плана. Чем больше проект, тем больше он требует формальностей.

Начальная фаза может принимать множество разных форм. Для некоторых проектов все начинается с обычного разговора, для крупных она может вылиться во всестороннее изучение всех возможностей реализации проекта, которое займет месяцы. Вырабатывается бизнес-план проекта – приблизительная стоимость и доход, который он принесет.

Разработчику также нужно определить границы проекта и сделать некоторый начальный анализ для оценки его размеров. Начальная фаза должна занимать несколько дней, в течение которых следует решить, стоит ли проводить в фазе уточнения более глубокие исследования.

Концепция

В начальной фазе у разработчика имеется нечеткое представление относительно требований к системе. Например:

Создать систему для электронных продаж информации, хранящейся в виде файлов (графических, текстовых, архивных), причем некоторая часть файлов доступна бесплатно, в целях рекламы. Предусмотреть наличие незарегистрированных пользователей и постоянных (зарегистрированных клиентов)

или

Создать информационную систему функционирования госпиталя, ориентированную на гибкую систему назначения и контроля медицинского обслуживания.

Документированное описание требований будет объемнее представленного выше, но не содержательнее.

В данный момент следует попытаться лучше уяснить свои задачи:

- Что будет создано?
- Как это можно сделать?
- Какую технологию использовать?

Определяя вопросы для рассмотрения во время начальной фазы, необходимо оценить степень риска, ожидающего проект.

- Что может привести к провалу проекта?
- Чем больше риск, тем больше внимания ему следует уделить.

Наличие ясной концепции – ключ к разработке продукта, который соответствует реальным потребностям ваших заказчиков. Концепция являет собой сущность *Requirements Workflow* (рабочего потока требований) в RUP:

- анализ проблемы;
- понимание потребностей заказчиков;
- определение системы;
- отслеживание требований по мере их изменения.

Концепция предоставляет руководителю основу для более подробных технических требований. Таким образом, под концепцией подразумевается четкий взгляд на проект создания ПО, который может быть ясно сформулирован

для любого руководителя или разработчика в процессе выполнения проекта. Концепция содержит самые высокоуровневые требования и конструктивные ограничения, предоставляя читателю понимание принципов разрабатываемой системы. Она также является отправной точкой для процесса утверждения проекта, и поэтому тесно связана с бизнес-прецедентами.

1.1.2. Определение требований при проектировании

Требования содержат ответы на основные вопросы по проекту, они являются средством обоснования будущих решений. При определении требований необходимо ответить на следующие вопросы, которые также могут быть разбиты на отдельные пункты.

- Каковы ключевые термины? (Словарь)
- Какую проблему необходимо решить? (Формулировка высокоуровневых требований)
- Кто заказчик? Кто пользователь? Каковы их запросы?
- Каковы свойства продукта? (Контекст системы)
- Каковы функциональные требования? (Прецеденты)
- Каковы нефункциональные требования?
- Каковы конструктивные ограничения?

При формулировке требований необходимо учитывать следующие аспекты.

1. **Хранение списка идей.** Некоторые из них могут превратиться в требования. Список предложений используется исключительно для планирования работ. Каждое предложение имеет короткое название и содержит краткое объяснение или определение, а также набор планируемых значений:
   - состояние предложений (одобрено, включено в план, утверждено);
   - сметная себестоимость выполнения (человеко-часы);
   - приоритет (критический, значительный, обычный);
   - уровень риска (критический, значительный, обычный);
   - оценка размера проекта.
2. **Моделирование предметной области и бизнес-моделирование.** Предметная область описывает важные понятия контекста как объекты предметной области. Предметная область связывает объекты друг с другом.

   Идентификация и наименование объектов помогают разработать словарь терминов. Цель бизнес-моделирования состоит в описании процессов: существующих или воспринимаемых.
3. **Определение всех Use Cases (UC),** которые охватывают все функциональные требования. Для пользователя вариант использования – это способ, которым он взаимодействует с системой.
4. **Ограничения среды и реализации, производительность, зависимость от платформы, расширяемость и надежность.**

1.1.3. Словарь системы

С помощью классов обычно моделируют абстракции, извлеченные из решаемой задачи или технологии, применяемой для ее решения. Такие

абстракции являются составной частью словаря Вашей системы, т. е. представляют сущности, важные для пользователей и разработчиков.

Для пользователей не составляет труда идентифицировать большую часть абстракций, поскольку они созданы на основе тех сущностей, которые уже использовались для описания системы.

Моделирование словаря системы включает в себя следующие этапы.

Определение элементов, которые пользователи и разработчики применяют для описания задачи или ее решения. Для отыскания правильных абстракций используется анализ прецедентов.

Выявление для каждой абстракции соответствующее ей множество обязанностей. Каждый класс должен быть четко определен, а распределение обязанностей между ними хорошо сбалансировано.

Разработка атрибутов и операций, необходимых для выполнения классами своих обязанностей.

Например: набор классов для системы розничной торговли: Customer (Клиент), Order (Заказ) и Product (Товар). В словаре может быть представлено несколько соотнесенных с ними абстракций, взятых из словаря проблемной области: Shipment (Отгрузка), Invoice (Накладная), Warehouse (Слад). Одна абстракция, а именно Transaction (Транзакция), применяемая к заказам и отгрузкам, связана с терминологией решения задачи.

1.1.4. Выработка плана

В RUP вся информация, необходимая для управления проектом, собирается в план развития программного обеспечения – Project Definition (PD). PD может корректировать ряд отдельных пунктов, разработанных в начальной фазе; должен постоянно поддерживаться в актуальном состоянии в процессе выполнения проекта.

PD определяет график выполнения проекта и включает:

- Project Plan (План Проекта);
- Iteration Plan (План Итераций) и потребности в ресурсах (Ресурсы и Средства). Используется для контроля выполнения графика, а также управляет планированием для других компонентов процесса;
- Project Organization (Организация Проекта);
- Requirements Management Plan (План Управления Требованиями);
- Configuration Management Plan (План Управления Конфигурацией);
- Problem Resolution Plan (План Решения Проблем);
- QA Plan (План Контроля Качества);
- Test Plan (План Тестирования);
- Test Cases (Тестирование Образцов);
- Evaluation Plan (План Экспертизы);
- Product Acceptance Plan (План Приемки Продукта).

Формат PD не так важен, как та деятельность и соображения, которые в него вошли.

Основными результатами фазы исследования является определение базовой архитектуры системы:

- список вариантов использования, определяющих требования к системе;
- модель предметной области, которая отражает понимание бизнеса разработчиком (аналитиком) и служит основой для формирования классов;
- основные элементы технологии реализации системы и их взаимодействие.

Фаза исследования занимает около 1/5 общей продолжительности проекта.

Необходимо также определить последовательность итераций конструирования и вариантов использования. Планирование завершается, когда для каждого варианта использования определена своя операция и получена дата каждой итерации.

Первый и основная задача заключается в отнесении отдельных Use Cases к различным категориям:

- эта функция абсолютно необходима для реальной системы;
- какое-то время можно обойтись без этой функции;
- функция нужна, но может быть реализована гораздо позднее.

Требуется также оценить объемы и время для выполнения работ с точностью до человеко-месяца исходя из предположения, что придется выполнить анализ, проектирование, тестирование и документирование.

1.1.5. Разработка

На стадии разработки построение системы выполняется путем серии итераций. На каждой итерации для конкретных вариантов использования выполняются анализ, проектирование, кодирование, тестирование и интеграция. Итерация завершается демонстрацией результатов пользователям и выполнением системных тестов. Такие действия снижают степень риска. Не следует откладывать решений сложных проблем на конец проекта.

Итерации являются инкрементными в соответствии с той функцией, которую они выполняют. Каждая итерация добавляет очередные конструкции к вариантам использования, реализованным во время предыдущих итераций. Они повторяются по отношению к разрабатываемому коду. На каждой итерации некоторая часть кода заново переписывается с целью сделать его более гибким. В этом процессе очень часто используется метод реорганизации. Реорганизационные изменения обычно незначительны: переименование метода, перемещение атрибута из одного класса в другой, консолидация двух подобных методов в суперкласс. Каждое отдельное изменение невелико, однако даже пара часов, потраченных на внесение изменений, могут значительно улучшить программу. Не следует одновременно заниматься реорганизацией программы и расширением ее функциональности.

1.1.6. Бизнес-моделирование

Описание структуры и динамики моделируемой информационной системы (ИС).

Выработка одинакового понимания заказчиками, пользователями и разработчиками моделируемой ИС.

Четкое определение «границы системы».

Предоставление информации, необходимой для планирования работ.

Описание бизнес-процессов проводится в целях их дальнейшего анализа и реорганизации. Целью реорганизации может быть внедрение ИС, сокращение затрат на выпуск продукции, повышение качества обслуживания клиентов и т. д. Для каждой такой задачи существуют определенные параметры, определяющие набор критических знаний по бизнес-процессу. От задачи к задаче требования к описанию бизнес-процессов могут меняться.

1.1.7. Задачи бизнес-моделирования.

Модель бизнес-процесса должна давать ответы на следующие вопросы.

1. Какие процедуры и в какой последовательности необходимо выполнить для получения заданного конечного результата.
2. Какие механизмы контроля и управления существуют в рамках рассматриваемого бизнес-процесса.
3. Кто исполнитель процедуры процесса.
4. Какие входящие документы использует и какие исходящие документы генерирует каждая процедура процесса.
5. Какие ресурсы необходимы для выполнения каждой процедуры процесса.
6. Какие параметры и документация характеризуют выполнение процедур и процесса в целом.

Описание бизнес-процесса формируется при помощи нотации и инструментальной среды, отражающих все вышеуказанные аспекты. Только в этом случае модель бизнес-процесса окажется полезной для предприятия, так как ее можно будет подвергнуть анализу и реорганизации.

1.1.9. Идентификация и смягчение рисков

Определить и предпринять энергичные меры против рискованных элементов проекта как можно раньше – один из существенных принципов RUP. Каждый элемент риска, определенный коллективом разработчиков, должен иметь соответствующий план уменьшения риска. Список таких элементов должен служить и как инструмент планирования деятельности по проекту, и как база для определения итераций.

Непрерывный анализ объективных данных, получаемых непосредственно в результате текущей деятельности и изменяющих конфигурацию продукта, важен для любого проекта. В RUP регулярные оценки состояния обеспечивают механизм для адресации, сообщения и решения деловых проблем, технических проблем и рисков проекта. Если соответствующая команда выявила препятствия, она должна определить каждой из этих проблем соответствующую дату и человека, ответственного за ее разрешение. Ход дела должен регулярно контролироваться, а обновления должны выполняться по мере необходимости.

Эти «отпечатки» проекта подчеркивают проблемы, требующие внимания руководства. Хотя время между проверками может меняться, регулярные оценки позволяют менеджерам быть в курсе развития проекта и устранять любые остановки и препятствия, мешающие движению вперед.

Выделяют основные четыре категории рисков.

1. Риски, связанные с требованиями. Опасность заключается в том, что построенная Вами система не будет удовлетворять требованиям пользователей. Во время фазы уточнения необходимо как следует разобраться с требованиями и их приоритетами.

2. Технологические риски. Следует задать себе вопросы:
*будут ли использоваться объекты? Имеется ли у исполнителей достаточный опыт объектно-ориентированного программирования?
*будет ли использоваться язык Java? Насколько хорошо эта технология работает? Возможно ли действительно реализовать те функции, которые пользователи ждут от Web-браузера, связанного с базой данных?

3. Риски, связанные с квалификацией персонала. Имеются ли сотрудники с необходимым опытом и квалификацией?

4. Политические риски. Имеются ли на Вашем пути внешние силы, которые могут серьезно повлиять на проект?

5. Риски, связанные с требованиями. Учет всех требований к системе очень важен и является первым серьезным испытанием для методов UML. Исходным пунктом являются варианты использования, которые служат «движущей силой» всего процесса разработки. Варианты использования (Use Cases) отражают типичное взаимодействие пользователя с системой для достижения некоторых целей.

Варианты использования могут значительно различаться размерами. Важно при этом то, что каждый из них определяет некоторую функцию, которая понятна пользователю и имеет для него определенное значение.

Одним из Use Cases может быть «регистрация клиента», другим – «формирование заказа и получение счета».

Варианты использования являются основой для общения и взаимопонимания между заказчиком и разработчиком при тестировании проекта.

Одна из наиболее важных вещей, которые необходимо сделать на фазе уточнения, – определение всех возможных вариантов использования системы, которая будет разработана.

Варианты использования не должны быть слишком детализированными. Пакет должен быть достаточно содержательным и для пользователя, чтобы они поняли основную идею, и для разработчиков, чтобы они хорошо представляли, что скрывается за этим описанием.

Другой важной задачей является разработка основы концептуальной модели предметной области. Картина бизнес-логики находится в голове у одного или нескольких пользователей. Например:

Заказчик может выбрать способ оплаты за услуги (файлы с информацией):

- оплата счет-фактурой;
- по кредитной карточке;
- используя предварительно внесенный кредит;
- указывая счет в банке-поручителе, с которым и производятся расчеты.

Предыдущий абзац содержит слова «заказчик», «услуга», «счет». Что они означают и как связаны друг с другом?

Концептуальная модель закладывает основы моделей объектов, которые будут позднее использоваться в процессе разработки для представления объектов системы.

Модели предметной области и варианты использования охватывают функциональные требования к системе; модели анализа исследуют, каким образом эти требования отражаются в конкретном приложении; модели проектирования добавляют внутреннюю инфраструктуру, чтобы сделать приложения работающими.

Назначение модели предметной области – исследование ее лексики области в терминах, имеющих значение и смысл для экспертов.

Далее разрабатывается модель проектирования, которая представляет информацию системы (объекты предметной области) и поведение системы (Use Cases). Добавляются классы, выполняющие некоторую реальную работу и образующие повторно используемую архитектурную основу для ее последующих расширений.

Новые варианты использования могут постепенно добавляться и включаться в модель проектирования как часть интерактивного процесса разработки.

При построении диаграмм классов следует делать поясняющие примечания на наиболее важных узлах и местах предполагаемого риска.

Команда разработчиков должна интенсивно работать во время всей фазы уточнения, пока не завершит модуль. В течение этого времени менеджер проекта должен следить, чтобы команда, с одной стороны, не завязла в мелких деталях, а с другой – не создавала слишком поверхностную модель.

Когда они разберутся в сути того, что делают, погружение в мелкие детали станет самой большой опасностью. В этой ситуации, чтобы сконцентрировать их внимание в нужном направлении, следует установить сжатый срок завершения работы.

1.1.10. Технологические риски

Наилучший способ – это строить прототип с помощью той технологии, которая будет применяться. При использовании Java-технологий и реляционной базы данных следует:

- определить версию компилятора Java и сопутствующих средств;
- сделать простейший прототип для ранней версии предметной области и определить степень соответствия этих средств для проекта;
- создать базу данных и связать ее с приложением на Java;
- попробовать несколько различных средств и технологий, определить, какие из них наиболее удобны.

Сложности возникают при сборке разнородных компонентов в единый проект, поэтому на ранней стадии процесса создания системы необходимо испытать все средства, которые будут использованы в дальнейшем. На этой стадии отрабатываются архитектурные решения. Особое внимание уделяется тем решениям, которые трудно будет изменить впоследствии, и следовательно, необходимо стремиться к тому, чтобы можно было легко вносить изменения в проект.

Риски, связанные с квалификацией персонала. *Обучение – это способ избежать ошибок, поскольку учителя их уже сделали. Ошибки отнимают время, а время стоит денег.*

Политические риски. *Возникают, как правило, из-за конфликта интересов в рамках организации, выполняющей проект.*

1.1.11. Вид системы с точки зрения прецедентов

Системы не существуют в изоляции. Как правило, они взаимодействуют с актерами – людьми или программами, которые используют систему в своих целях, причем каждый актер ожидает, что она будет вести себя определенным, вполне предсказуемым образом.

Прецедент (Use Case) специфицирует поведение системы или ее части и представляет собой описание множества последовательностей действий (включая варианты), выполняемых системой для того, чтобы актер мог получить определенный результат.

Графически прецедент изображается в виде эллипса. Нотация прецедента похожа на нотацию кооперации.

1.1.12. Имена

Любой прецедент должен иметь имя, отличающее его от других прецедентов. Оно должно быть уникально внутри объемлющего пакета. Имя прецедента представляет собой текстовую строку. Взятое само по себе, оно называется простым именем. К составному имени впереди добавлено имя пакета, в котором он находится. Обычно при изображении прецедента указывают только его имя.

Имя прецедента может состоять из любого числа букв, цифр и некоторых знаков препинания (за исключением таких, как двоеточия, которые применяются для отделения имени прецедента от имени объемлющего пакета). Имя может занимать несколько строк. На практике для наименования прецедентов используют короткие глагольные фразы в активной форме, обозначающие некоторое поведение и взятые из словаря моделирующей системы. Например:

Администратор: Удаление пользователя

Модератор: Создание раздела

С помощью прецедентов можно описать поведение разрабатываемой системы, не определяя ее реализацию. Таким образом, они позволяют достичь взаимопонимания между разработчиками, экспертами и конечными пользователями продукта. Кроме того, прецеденты помогают проверить архитектуру системы в процессе ее разработки. Реализуются они кооперациями.

Прецедент описывает множество последовательностей. Каждая из них представляет взаимодействие сущностей, находящихся вне системы (ее актеров), системой как таковой и ее ключевыми абстракциями. Такие взаимодействия являются в действительности функциями уровня системы, которые используются для визуализации, конструирования и документирования ее желаемого поведения на этапах сбора и анализа требований. Прецедент представляет функциональные требования к системе в целом.

Развитие прецедентов может осуществляться по-разному. В любой хорошо продуманной системе существуют прецеденты, которые

- являются специализированными версиями других, более общих;
- входят в состав прочих прецедентов;
- расширяют их поведение.

Общее поведение множества прецедентов, допускающее повторное применение, можно выделить, организуя их в соответствии с тремя описанными видами отношений.

Любой прецедент должен выполнять некоторый объем работы. С точки зрения актера, прецедент делает нечто представляющее для него определенную ценность, например передает сообщение, создает новый объект или изменяет состояние другого объекта.

Прецеденты могут быть применены ко всей системе или к ее части, в том числе к подсистемам или даже к отдельным классам и интерфейсам. В любом случае прецеденты не только представляют желаемое поведение этих элементов, но и используются как основа для их тестирования на различных этапах разработки. Прецеденты в применении к подсистемам – это источник регрессионных тестов.

1.1.13. Прецеденты и актеры

Прецеденты предполагают взаимодействие актеров и системы. Актер представляет собой логически связанное множество ролей, которые играют пользователи прецедентов во время взаимодействия с ними. Актерами могут быть как люди, так и автоматизированные системы. Например, при моделировании работы сервера подписки процесс рассылки файлов включает в себя, помимо всего прочего, активацию рассылки в заданный час актером «Время».

Актер представляет собой связное множество ролей, которые пользователи прецедентов исполняют во время взаимодействия с ними.

Обычно актер представляет роль, которую в данной системе играет человек, аппаратное устройство, время или другая система. Например, человек может играть роль Администратора системы, но при определенных обстоятельствах может играть роль Клиента. Таким образом, экземпляр актера представляет собой конкретную личность, взаимодействующую с системой определенным образом. Хотя актеры используются при моделировании, они не являются частью системы, так как существуют вне ее.

Актер изображается в виде человеческой фигурки. Можно определить общие типы актеров (например, Гость) и затем специализировать их (например, создав разновидности Модератор, Клиент) с помощью отношений обобщения.

Актеров можно связывать с прецедентами только отношениями ассоциации. Ассоциация между актером и прецедентом показывает, что они общаются друг с другом, посылая или принимая сообщения.

1.1.14. Прецеденты и поток событий

Прецедент описывает, что делает система (подсистема, класс или интерфейс), но не определяет, каким образом она это делает. В процессе моделирования всегда важно разделять внешнее и внутреннее представления.

Можно специфицировать поведение прецедента путем описания потока событий в текстовой форме – в виде, понятном для постороннего читателя. В описание необходимо включить указание на то, как и когда прецедент начинается и заканчивается, когда он взаимодействует с актерами и какими сообщениями они обмениваются. Важно обозначить основной и альтернативный потоки поведения системы.

Например, в контексте систему организации банкомата можно было бы следующим образом описать прецедент *ValidateUser* (Проверить Пользователя).

Основной поток событий

Прецедент начинается, когда система запрашивает у клиента его персональный идентификационный номер (PIN). Клиент (Customer) может ввести его с клавиатуры. Завершается ввод нажатием клавиши *Ввод***.** После этого система проверяет введенный PIN и, если он правильный, подтверждает ввод. На этом прецедент заканчивается.

Исключительный (альтернативный) поток событий. Клиент может прекратить транзакцию в любой момент нажатием клавиши *Отмена*. Это действие начинает прецедент заново. Никаких изменений на счету у клиента не производится.

Исключительный поток событий. Клиент может в любой момент до нажатия клавиши *Ввод* стереть свой PIN и ввести новый.

Исключительный поток событий. Если клиент ввел неправильный PIN, прецедент запускается сначала. Если это происходит 3 раза подряд, система отменяет всю транзакцию и блокирует карту.

Поток событий в прецеденте можно описать различными способами, как в виде неформализованного структурированного текста (как в вышеприведенном примере), так и в виде формализованного структурированного текста (с пред- и постусловиями).

1.1.15. Прецеденты и сценарии

В начале работы потоки событий прецедента описываются в текстовой форме. По мере уточнения требований к системе удобнее перейти к графическому изображению потоков на диаграммах взаимодействия. Для описания основного потока событий прецедента используется диаграмма последовательностей, а для дополнительных – ее варианты.

Часто все детали прецедента выразить с помощью одной последовательности невозможно. Например, в системе управления человеческими ресурсами присутствует прецедент. *Нанять работника*. У общей бизнес-функции существует множество вариаций. Вы можете переманить сотрудника из другой компании (так бывает чаще всего), перевести человека из одного подразделения в другое (эта практика распространена в

транснациональных компаниях) или нанять иностранца (особый случай, регулируемый специальными правилами). Каждый вариант описывается своей последовательностью.

В итоге один прецедент. *Нанять работника* описывает несколько последовательностей (сценариев), каждый из которых представляет одну из возможных последовательностей данного потока событий.

Сценарий (Scenario) – это некоторая последовательность действий, иллюстрирующая поведение системы. Сценарии находятся в таком же отношении к прецедентам, как экземпляры к классам, т. е. сценарий – это экземпляр прецедента.

Относительно сложная система содержит несколько десятков прецедентов, каждый может состоять из нескольких десятков сценариев. Для любого прецедента можно выделить основные сценарии (для описания важнейших последовательностей) и вспомогательные (для описания альтернативных последовательностей).

1.1.16. Прецеденты и кооперации

Прецедент описывает желательное поведение системы (подсистемы, класса или интерфейса), но не специфицирует его реализацию. Это важная особенность, поскольку анализ системы (по результатам которого специфицируется ее поведение) по возможности не должен учитывать проблемы реализации, однако прецеденты придется реализовать. Для этого необходимо создать пакет классов и других элементов, в результате совместной работы которых будет реализовано желаемое поведение. Такой пакет, включая его динамическую и статистическую структуру, называется и UML кооперацией.

1.1.17. Организация прецедентов

Для организации прецедентов их группируют в пакеты, так же как и классы.

Прецеденты можно организовывать, распределив между ними отношения обобщения, включения и расширения. Эти отношения применяют, чтобы выделить некоторое общее поведение (извлекая его из других прецедентов, которые его включают) или, наоборот, вариации (поместив такое поведение в другие прецеденты, которые его расширяют).

Отношение обобщения между прецедентами аналогично отношениям обобщения между классами. Это означает, что прецедент-потомок наследует поведение и семантику своего родителя, может замещать его или дополнять его поведение и, кроме того, может быть подставлен всюду, где появляется его родитель (как родитель, так и потомок могут иметь конкретные экземпляры). Например, в банковской системе возможно наличие прецедента. *Проверить клиента*, который отвечает за проверку личности клиента. Он может иметь двух специализированных потомков (*Проверить пароль* и *Сканирование сетчатки*). Оба потомка ведут себя так же, как прецедент. *Проверить клиента*, и могут использоваться везде, где и их родитель, но при этом каждый из них добавляет свое собственное поведение (первый проверяет текстовый пароль, а второй – рисунок сетчатки глаза).

Отношение включения между прецедентами означает, что в некоторой точке базового прецедента инкорпорировано поведение другого прецедента.

Включаемый прецедент никогда не существует автономно, а понимается только как часть всеобъемлющего прецедента. Можно считать, что базовый прецедент заимствует поведение включаемых прецедентов.

Благодаря наличию отношений включения удается избежать многократного описания одного и того же потока событий, поскольку общее поведение можно описать в виде самостоятельного прецедента, включаемого в базовые прецеденты. Отношение включения является примером делегирования, при котором некоторые обязанности системы описываются в одном месте (во включаемом прецеденте), а остальные прецеденты, когда необходимо, включают эти обязанности в свой набор.

Отношения включения изображаются в виде зависимостей со стереотипом include. Чтобы специфицировать место в потоке событий, где базовый прецедент включает поведение другого, пишется это include, за которым следует имя включаемого прецедента.

Основной поток событий

Отношение расширения подразумевает, что базовый прецедент неявно содержит поведение другого прецедента в точке, которая косвенно задается расширяющим прецедентом. Базовый прецедент может быть автономным, но при определенных обстоятельствах его поведение расширяется за счет другого. Базовый прецедент можно расширить только в некоторых точках, называемых точками расширения. Можно считать, что расширяющий прецедент передает свое поведение базовому.

Например: *получить и проверить номер заказа. Include (Проверить Клиента). Запросить статус каждой части заказа и доложить клиенту.*

Отношение расширения применяют для моделирования таких частей прецедента, которые пользователь воспринимает как необязательное поведение системы. Тем самым можно разделить обязательное и необязательное поведение. Отношения расширения используются также для моделирования отдельных подпотоков, выполняемых лишь при определенных обстоятельствах. Их применяют также для моделирования нескольких потоков, которые могут включаться в некоторой точке сценария в результате явного взаимодействия с актером.

Основной поток событий. Include (*Проверить Клиента*). Собрать все пункты сделанного клиентом заказа. (*Установить приоритет*). Отправить заказ на обработку.

В данном примере фраза *Установить приоритет* – это точка расширения. Прецедент может содержать несколько точек расширения (причем каждую по нескольку раз), идентифицируемых по именам. При обычных обстоятельствах базовый прецедент в этом примере выполняется без учета приоритетности заказа. Если же поступает приоритетный заказ, то поток будет выполняться, как обычно, до точки расширения (*Установить приоритет*), после чего возобновится работа главного потока. Если несколько точек расширения, то

расширяющие прецеденты будут последовательно выполняться в собственных потоках.

Организация прецедентов путем выделения общего поведения (отношение включения) и различных вариаций (отношение расширения) является важной составной частью процесса разработки простого, сбалансированного и понятного набора прецедентов системы.

Прецеденты являются классификаторами и могут иметь атрибуты и операции, которые изображаются так же, как и для классов. Атрибуты можно считать объектами внутри прецедента, которые требуются для описания его внешнего поведения, а операции – действиями системы, необходимыми для описания потока событий. Эти объекты и операции разрешается включать в диаграммы взаимодействия, чтобы специфицировать поведение прецедента.

1.1.18. Моделирование бизнес-процессов диаграммами классов

Самое главное в разработке ПО – получение работающего кода. Диаграмма – это всего лишь красивые картинки, а описание последовательности действий – всего лишь слова.

Следует задать вопрос: чем все это поможет, когда дело дойдет до создания кода?

Большинство людей начинают изучение программирования в его обычном виде, не используя объектно-ориентированные методы. В некоторых случаях переход к объектно-ориентированным методам происходит болезненно. Проблема заключается в философском осмыслении преимуществ объектно-ориентированного языка, который, безусловно, обладает преимуществами, но не предоставляет их автоматически. Чтобы использовать эти преимущества, необходимо изменить образ мышления.

Чтобы привыкнуть к новому образу мыслей, лучше начинать с диаграмм классов. Существуют три различные точки зрения на построение любой модели и, в частности, на построение диаграмм классов.

1.1.19. Концептуальная точка зрения

Классы должны отображать понятия изучаемой предметной области. Эти понятия будут соответствовать реализующим их классам, однако прямое соответствие обычно отсутствует. Концептуальная модель может иметь слабое отношение к реализующему ее программному обеспечению, поэтому ее следует рассматривать как независимую от языка программирования.

1.1.20. Точка зрения спецификации

Рассматривается уровень ПО, но определяются только интерфейсы, а не реализация. Объектно-ориентированная разработка подчеркивает различие между интерфейсами и реализацией, но класс в объектно-ориентированном языке объединяет как интерфейс, так и реализацию. Часто интерфейсы выступают в роли типов, а их реализации – в роли классов. Тип представляет интерфейс, который может иметь множество реализаций, отличающихся из-за среды реализации.

1.1.21. Точка зрения реализации

В этом рассматривается уровень реализации. Этот подход наиболее распространенный, однако во многих случаях точка зрения спецификации является предпочтительной, учитывая возможность повторного использования классов, что требует серьезной подготовки и достаточного опыта от разработчика.

В дальнейшем большинство тем будет рассматриваться с точки зрения реализации, так как конечной целью является создание конкретного ПО.

Когда строится диаграмма, точка зрения должна быть ясной и единственной. При чтении диаграммы первым делом необходимо выяснить, с какой точки зрения она строилась. Если нужно интерпретировать диаграмму правильным образом, то без этого знания не обойтись.

Выбранная для описания моделей точка зрения должна соответствовать конкретной стадии проекта:

- на стадии анализа строятся концептуальные модели;
- при работе с ПО рассматриваются модели спецификаций;
- модели реализации используются в тех случаях, когда иллюстрируется конкретный способ реализации.

Методы проектирования позволяют получить общее представление о системе. Один взгляд на диаграмму классов позволяет определить, какого рода абстракции имеются в системе и какие ее части нуждаются в дальнейшем уточнении.

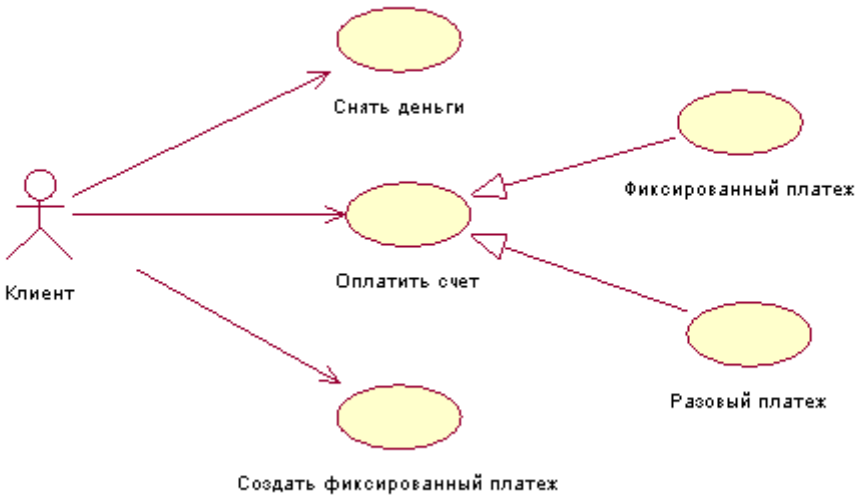


Рисунок 2 – Диаграмма прецедентов.

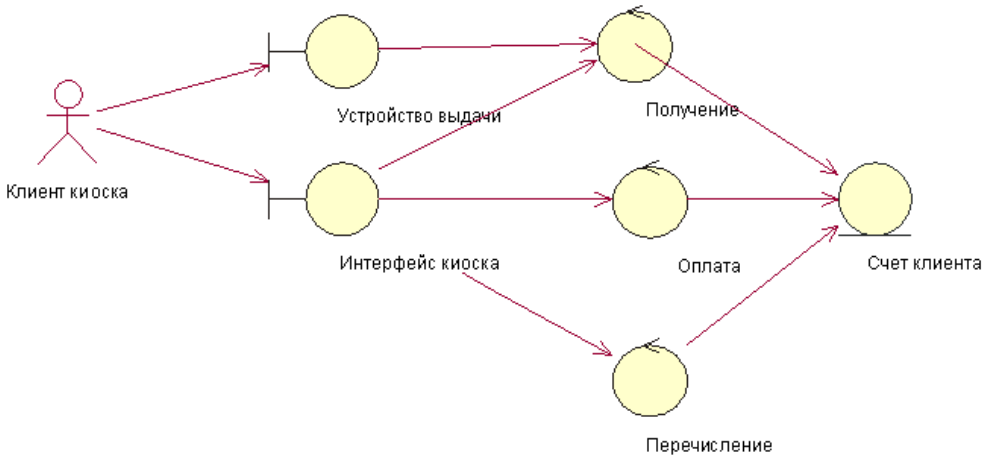


Рисунок 3 – Диаграмма классов.

Устройство выдачи и *Интерфейс киоска* – граничные классы, *Получение* – управляющий класс, *Счет клиента* – класс сущности.

Граничный класс моделирует взаимодействие между системой и ее актерами.

Управляющий класс отвечает за представление координации, последовательности, взаимодействие и управление другими объектами. Инкапсуляция управления, относящегося к определенному варианту использования.

Класс сущности моделирует информацию длительного хранения.

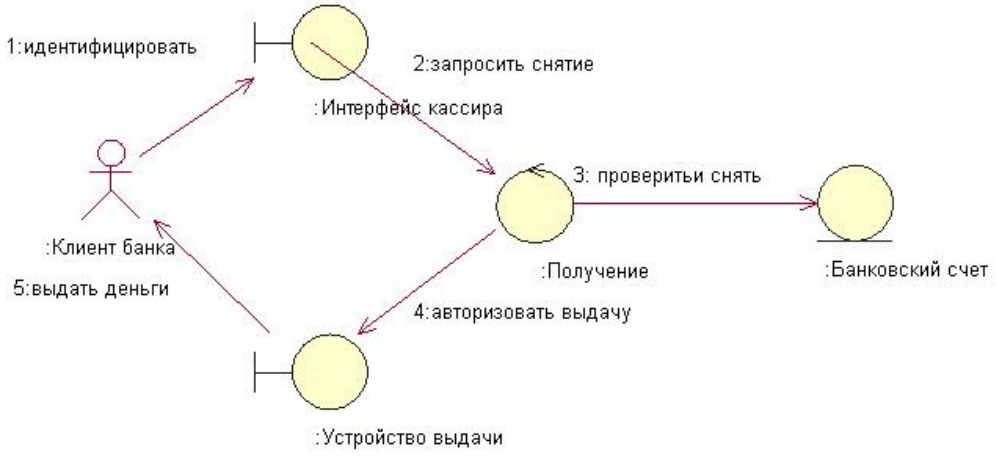


Рисунок 4 – Диаграмма взаимодействий классов.

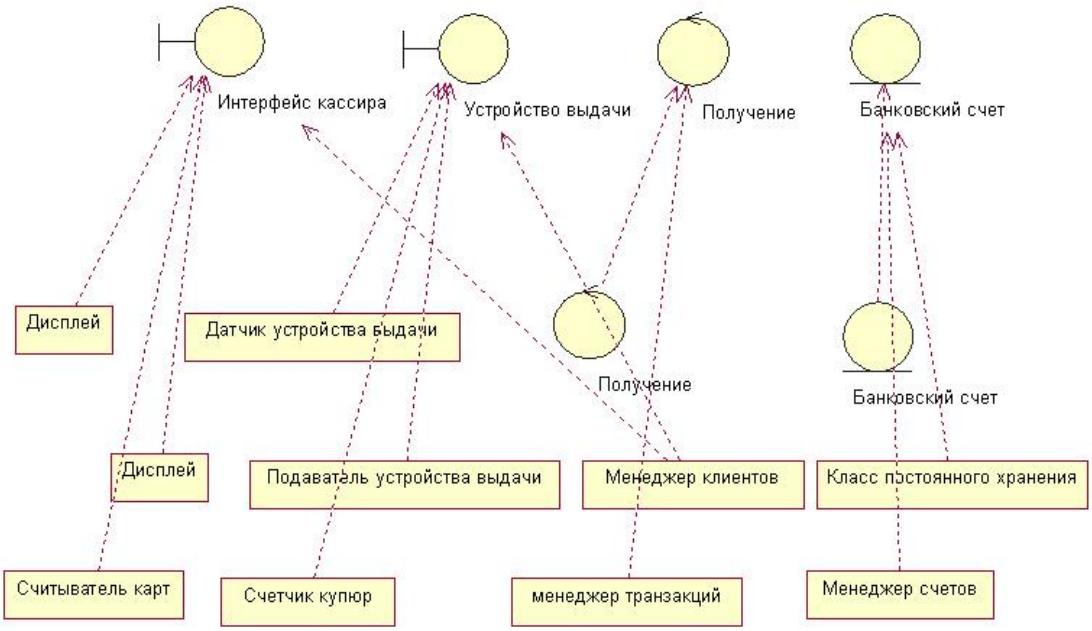


Рисунок 5 – Модель анализа.

Рисунок 6 – Модель проектирования.

На рисунке 4 представлена диаграмма классов, являющаяся частью реализации варианта использования «Снять деньги со счета» модели проектирования. Каждый класс проектирования участвует в реализации варианта использования.

1.1.22. Предметная область

Многие классы предметной области можно определить из технического задания или в ходе опроса специалистов по проблемной области.

Классы предметной области можно разбить на три типовые категории.

1. Бизнес-объекты – сущности, используемые в бизнесе (заявки, счеты, контракты).

2. Объекты и понятия реального мира, которые система должна отслеживать (вражеские ракеты, самолеты, тракторы).

3. События, которые произойдут или произошли (прибытие и отлет самолетов, перерыв на обед).

Пример: заказ, счет, предмет, банковский счет.

Заказ – запрос покупателя продавцу на поставку изделий. Изделие занимает строку в заказе. Заказ имеет атрибуты – дату выписки, адрес поставки.

Счет – запрос на оплату, посылаемый продавцом покупателю в ответ на заказ товаров и услуг. Атрибуты – сумма к оплате, дата выписки, отсрочка. На несколько заказов может быть выслан один счет.

Счет считается оплаченным, когда деньги с банковского счета покупателя перешли на банковский счет продавца. *Банковский счет* имеет атрибуты – баланс, владелец.

Число основных классов предметной области может быть от десятка до нескольких сотен. Кандидаты в классы сохраняются в словаре.

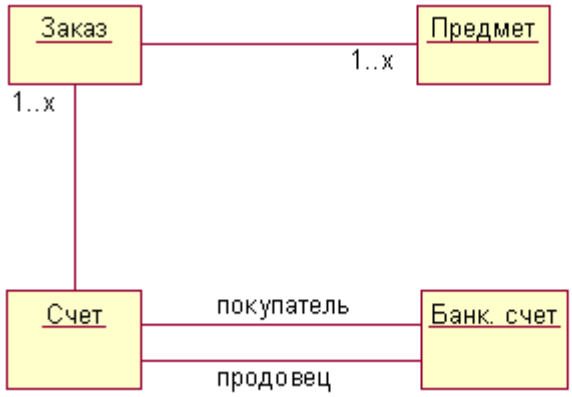


Рисунок 7 – Классы и связи.

Более регулярный способ определения вариантов использования и поиска классов в системе – разработка бизнес модели. Модель предметной области является частным случаем более полной бизнес-модели.

Модель бизнес-объектов описывает, как каждый Use Case реализуется сотрудниками, использующими бизнес-объекты и рабочие модели.

Пример: Use Case «*Продажа. От Заказа до Поставки*».

Сотрудники должны последовательно выполнять следующие действия.

1. Покупатель заказывает товары или услуги.
2. Продавец посылает счет покупателю.
3. Продавец предоставляет покупателю товары.
4. Покупатель платит платеж.

1.1.23. Архитектура системы

Архитектура системы – это совокупность определенных решений относительно:

организации программной системы;

выбора структурных элементов, составляющих систему и их интерфейсов;

поведения этих элементов при взаимодействии с другими элементами;

составления из этих элементов более крупных подсистем;

стиля, направляющего и определяющего всю организацию системы – статические и динамические элементы, их интерфейсы, кооперации и способ их объединения.

Для визуализации, конструирования и документирования программных систем необходимо рассматривать их с различных точек зрения. Все участники проекта – конечные пользователи, аналитики, менеджеры проектов, разработчики, системные интеграторы, тестировщики – преследуют собственные интересы, и каждый смотрит на создаваемую систему по-разному.

Архитектура программной системы наиболее оптимально может быть описана с помощью пяти взаимодействующих видов или представлений.

Вид с точки зрения прецедентов (Use case view). Охватывает прецеденты, которые описывают поведение системы, наблюдаемое конечным пользователем, аналитиками и тестировщиками. Этот вид специфицирует движущие силы, от которых зависит формирование системной архитектуры.

Вид с точки зрения процессов (Process view). Описывает процессы и взаимодействия между элементами системы. Охватывает главным образом производительность и пропускную способность системы.

Вид с точки зрения проектирования (Design view). Охватывает классы и интерфейсы. Этот вид поддерживает функциональные требования, предъявляемые к системе, т. е. услуги, которые она должна предоставлять конечным пользователям.

Вид с точки зрения реализации (Implementation view). Охватывает компоненты и файлы, используемые для сборки и выпуска конечной продукции.

Вид с точки зрения развертывания (Deployment view). Охватывает узлы, формирующие топологию аппаратных средств, на которой система выполняется.

1.1.24. Архитектура

Архитектура включает в себя следующую информацию:

- об организации программной системы;
- о структурных элементах, входящих в систему;
- о составе структурных элементов и элементов поведения наиболее крупных подсистем;
- о стиле архитектуры, принятом в данной организации

Архитектура необходима для того, чтобы:

- понять систему;
- организовать разработку;
- способствовать повторному исполнению кода;
- развивать систему в дальнейшем.

Понимание системы сложно, так как:

- она реализует сложное поведение;
- работает в сложном окружении;

- сложна технологически;
- часто сочетает распределенные вычисления, коммуникации, продукты и платформы, многократно используются компоненты и структуры;
- должна удовлетворять запросам как отдельных моделей, так и организующих;
- система очень велика, так что приходится разделять на отдельные проекты.

При проектировании архитектуры возможно внесение следующих изменений при:

- обнаружении новых абстрактных классов и интерфейсов;
- добавлении новых функциональных возможностей с существованием подсистем;
- обновлении компонентов многократного использования до новых версий;
- перестройке структуры процесса.

1.1.25. Трехуровневая архитектура

1. Уровень представления (Presentation, View) – окна, отчеты.
2. Уровень логики приложения (Application Logic, Controller) – бизнес-логика, правила управления процессом.
3. Уровень данных (Storage, Model) – постоянное хранилище данных.

Основной частью трехуровневой архитектуры является вынесение логики приложения на отдельный уровень.

Объекты уровня представления направляют запросы на средний уровень бизнес-логики, а средний уровень в свою очередь взаимодействует с самым нижним уровнем хранения данных.

Такое разделение обеспечивает возможность:

- представления бизнес-логики посредством отдельных компонентов, обеспечивающих возможность повторного использования объектов;
- распространения логики приложения на другой компьютер.

В объектно-ориентированных системах выполняется декомпозиция бизнес-логики на несколько мелких уровней. Например на два уровня:

- специальные объекты (domain objects) – классы, представляющие понятия предметной области;
- службы (services) – служебные объекты для взаимодействия с базой данных (БД), создания отчетов, security и т. д.

После выполнения декомпозиции бизнес-логики трехуровневая архитектура превращается в многоуровневую архитектуру выраженных границ уровня логики. Существующие уровни можно подвергать дальнейшей декомпозиции и добавлять в систему дополнительные уровни.

Например: уровень служб можно разделить на высокоуровневые и низкоуровневые службы (для генерации отчетов и обработки файлов ввода-вывода соответственно).

Приложения с трехуровневой архитектурой могут физически располагаться в различных конфигурациях, в том числе:

- компоненты бизнес-логики и уровней представления размещаются на клиентском компьютере, а БД – на сервере;
- компоненты уровня представления располагаются на клиентском компьютере, уровень бизнес-логики – на сервере приложений, а данные – на отдельном сервере БД.

Обоснования многоуровневой архитектуры:

- бизнес-логика представляется в виде изолированных компонентов, которые можно использовать в других системах;
- различные уровни бизнес-логики можно распределять по разным компьютерам и (или) процессам;
- разработку отдельных уровней можно поручить специализированным группам разработчиков, т. е. поддерживается уровень специализации профессионалов и возможность параллельной разработки всех уровней приложения.
- Всю систему можно рассматривать как единый пакет верхнего уровня – пакет System. В рамках пакета определяется вложенное пространство имен, в различных пакетах могут встречаться элементы с одинаковыми именами.

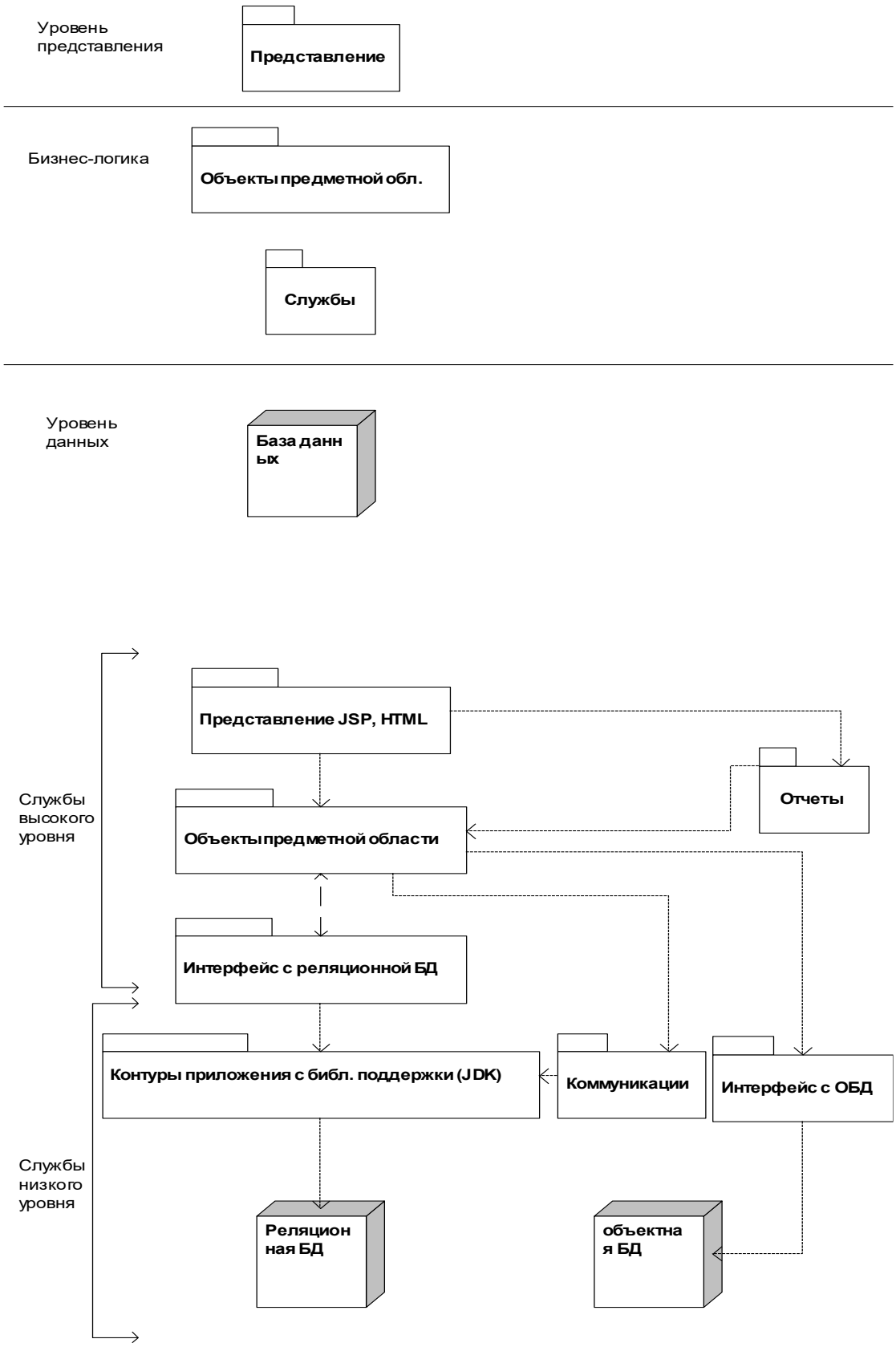


Рисунок 8 – Диаграмма архитектуры пакетов.

Пакеты *(комментарии)*.

Интерфейсы реляционной и объектной БД – механизмы взаимодействия с БД. Интерфейс с объектной БД обеспечивается разработчиком самой БД, а реляционной – разрабатывается самостоятельно.

Отношения зависимости свидетельствуют о том, что один пакет знает о существовании другого. Отсутствие зависимости между пакетами говорит об отсутствии ссылок между классами, методами и т. д.

При объединении элементов в пакеты используются следующие принципы:

- в один пакет объединяются элементы, обеспечивающие общие функции с высоким уровнем связывания и кооперации;
- с точки зрения более высокого уровня абстракции пакет должен иметь высокую степень зацепления, его элементы должны выполнять подобные обязанности.

При этом связывание и кооперация между пакетами должны быть достаточно слабыми.

1.1.26. Архитектурные представления модели проектирования

1.1.27. Уровни и разделы

Многоуровневую архитектуру можно разделить на уровни и разделы.

Уровни (layer) – деление системы по вертикали; разделы (partition) – деление по горизонтали на параллельные подсистемы в рамках одного уровня.

Например: уровень служб можно разделить на разделы, отвечающие за выполнение требований безопасности и формирование отчетов.

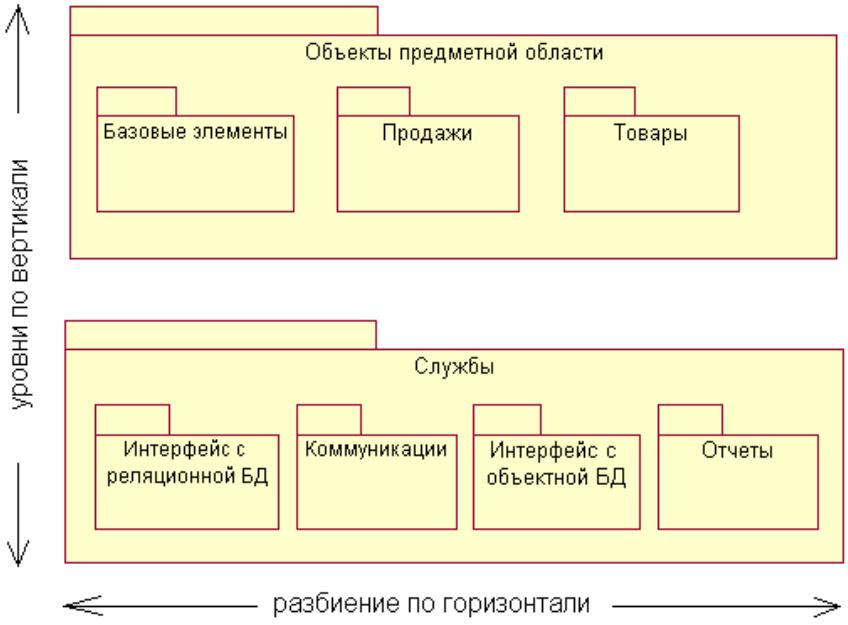
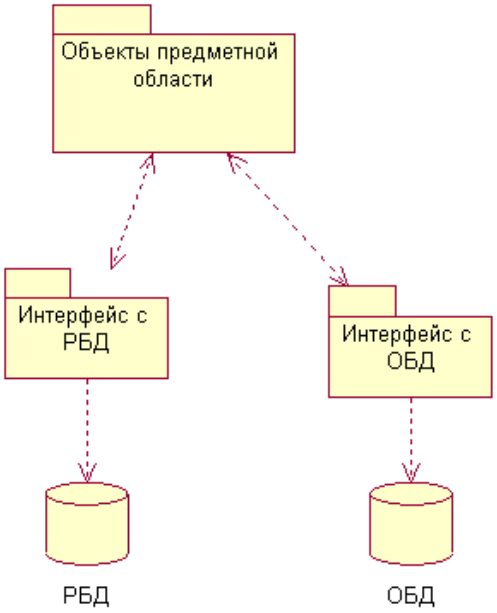


Рисунок 9 – Уровни архитектуры.

1.1.28. Объекты хранения данных

Если для хранения объектов уровня представления используется объектная БД, то для взаимодействия с ней требуются минимальные усилия разработчиков. БД сама поддерживает этот интерфейс. Однако при использовании необъектной БД (реляционной) необходима специальная подсистема, обеспечивающая преобразование объектов в строки таблиц.



1.2. Унифицированный язык моделирования

Унифицированный язык моделирования (UML) − графический язык визуализации, специфицирования, конструирования и документирования программного обеспечения. С помощью UML можно разработать детальный план создаваемой системы, отображающий системные функции и бизнес-процессы, а также конкретные особенности реализации. А именно:

- классы, написанные на специальных языках программирования;
- схемы БД;
- программные компоненты многократного использования.

1.2.1. Нотации и метамодели

Нотация – совокупность графических объектов, которые используются в моделях. Она является синтаксисом языка моделирования.

Нотация диаграммы классов определяет способ представления класса, ассоциации, множественности. Причем эти понятия должны быть точно определены.

Проектирование подразумевает всесторонний анализ всех ключевых вопросов разработки. И строгое определение всех понятий может не позволить описать реальные требования ситемы.

Большинство объектно-ориентированных методов являются не слишком строгими. Их нотация прибегает в большей степени к интуиции, чем к формальному определению.

Разработчики объектно-ориентированных методов ищут способы добиться большей строгости методов, не сужая области их использования.

Метамодель – диаграмма, определяющая нотацию.

Метамодель помогает понять, что такое хорошо организованная, т. е. синтаксически правильная, модель.

Уровень владения и понимания языка моделирования зависит от задач, которые решаются с его помощью. В основном диаграммы используются как средства обмена информации между разработчиками.

Если не придерживаться официальной нотации, то другие разработчики просто не поймут, что Вы хотели выразить своей диаграммой.

Основные понятия

К основным понятиям UML относятся:

- *Сущности* – абстракции, являющиеся основными элементами модели;
- *Отношения* – связывают различные сущности;
- *Диаграммы* – группируют представляющие интерес совокупности сущностей.

Сущности

- структурные – статические части модели, соответствующие концептуальным или физическим элементам модели;
- поведенческие – динамические составляющие, описывающие поведение модели во времени и в пространстве;
- группирующие;
- аннотационные.

Структурные сущности

Класс (Class) – описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Реализует несколько интерфейсов.

Интерфейс (Interface) – совокупность операций, которые определяют набор услуг, предоставляемый классом или компонентом. Описывает видимое извне поведение элементов.

Кооперация (Collaboration) – совокупность операций, которые производят некоторый общий эффект, не сводящийся к простой сумме слагаемых.

Вариант использования (Use Case) – описание последовательности выполняемых системой действий, которая производит наблюдаемый результат, значимый для какого-либо определенного действующего лица (Actor).

Активный класс (Active class) – класс, объекты которого вовлечены в один или несколько процессов и могут инициировать управляющее воздействие.

Компонент (Component) – физическая заменяемая часть системы, которая соответствует некоторому набору интерфейсов и обеспечивает его реализацию.

Узел (Node) – элемент реальной (физической системы), который существует во время функционирования программного комплекса и представляет собой вычислительный ресурс.

Поведенческие сущности

Взаимодействие (Interaction) – поведение, суть которого заключается в обмене сообщениями между объектами в рамках конкретного контекста для достижения определенных целей.

Автомат (State machine) – поведение, определяющее последовательность состояний, через которые объект или взаимодействие проходят на протяжении своего жизненного цикла в ответ на различные события, а также реакция на эти события.

Отношения

Отношением (relationship) называется связь между сущностями. В объектно-ориентированном моделировании особое значение имеют четыре типа отношений: зависимости, обобщения, ассоциации и реализации. Отношения изображаются в виде линий различного начертания.

Зависимость (Dependency) – семантическое отношение между двумя сущностями, при котором изменение одной (независимой) может повлиять на семантику другой (зависимой). Обратное не обязательно.

Ассоциация (Association) – структурное отношение, описывающее совокупность связей (соединений между объектами).

Обобщение (Generalization) – отношение наследования или «специализация/обобщение».

Реализация (Realization) – семантическое отношение между классификаторами, при котором один классификатор определяет контракт, а другой гарантирует его выполнение. Встречается в двух случаях – между интерфейсом и реализующим его классом и между вариантом использования и реализующей его кооперацией.

Диаграммы

Диаграммы – графическое представление набора элементов, изображаемое в виде графа с вершинами-сущностями и ребрами-отношениями.

Диаграммы являются некоторыми проекциями системы.

Диаграммы состояний – автомат, включающий в себя состояния, переходы, события и виды действий. Относятся к динамическому виду системы.

Диаграммы деятельности – частный случай диаграмм состояний, описывающий переходы потока управления от одной деятельности к другой. Относятся к динамическим аспектам системы.

Диаграммы последовательностей – являются частным случаем диаграмм взаимодействия. Относятся к динамическому виду системы и отражают временную упорядоченность сообщений. Взаимозаменяемы с диаграммами кооперации.

Диаграммы вариантов использования – относятся к статическому виду системы с точки зрения организации и моделирования поведения системы.

Диаграммы классов – показывают классы, интерфейсы, объекты и кооперации, а также их отношения и соответствуют статическому виду системы с точки зрения проектирования.

Диаграммы объектов (экземпляров) – являются статическими отображениями экземпляров сущностей, представленных на соответствующих диаграммах классов.

Диаграммы кооперации – частный случай диаграмм взаимодействия. Относятся к динамическому виду системы и отражают структурную организацию обменивающихся сообщениями объектов сообщений. (Взаимозаменяемы с диаграммами последовательностей.)

Диаграммы компонентов – представляют собой организацию совокупности компонентов и существующие между ними зависимости.

Диаграммы развертывания – представляют собой конфигурацию обрабатывающих узлов системы и размещенных в них компонентов.

1.2.2. Отношения

Классы редко существуют самостоятельно. Они разными способами взаимодействуют между собой. Моделируя систему, необходимо не только идентифицировать сущности, но и указать, как они соотносятся друг с другом. Моделируя сущности, составляющие словарь системы, необходимо показать и отношения между ними. Эти отношения могут быть весьма сложными. Для визуализации, специфицирования, конструирования и документирования сетей отношений необходимо воспользоваться более развитыми возможностями UML.

Отношения зависимости, обобщения и ассоциации представляют собой три наиболее важных связующих строительных блока в UML.

Зависимости применяются тогда, когда экземпляр одного класса использует экземпляр другого, например в качестве аргумента. В этом случае изменение экземпляра одного класса отразится на работе другого, так как используемый класс имеет свой интерфейс:

Зависимостью (dependency) называется отношение использования, определяющее, что изменение в спецификации одной сущности может повлиять на другую сущность, которая ее использует, причем обратное в общем случае неверно. Графически зависимости изображают в виде пунктирной линии со стрелкой, направленной в сторону сущности, от которой зависит еще одна. Применяются зависимости, если требуется показать, что одна сущность использует другую.

Для большинства отношений использования вполне достаточно обычной зависимости без каких-либо дополнений. Но если необходимо передать некий смысловой нюанс, то стоит учесть, что в UML определен целый ряд стереотипов, применимых к зависимостям. Всего существует 17 таких стереотипов, объединенных в шесть групп.

Рисунок 10 – Отношение зависимости.

Обобщение (generalization) означает, что объекты класса-потомка могут использоваться всюду, где встречаются объекты класса-родителя, но не наоборот. Потомок может быть подставлен вместо родителя. Он наследует свойства родителя (атрибуты и методы).

Понятия CashPayment, CreditPayment, CheckPayment очень похожи одно на другое, и в этом случае полезно объединить их в иерархию обобщения – специализации классов. Payment представляет более общее понятие, а его подклассы – специализированные свойства.

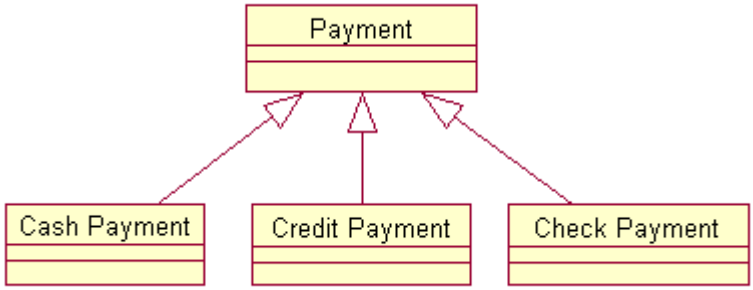


Рисунок 11 – Отношение наследования.

Обобщение – вид деятельности, связанный с идентификацией общности между понятиями и определением суперкласса и связанных с ним подклассов. Идентификация суперклассов и подклассов осуществляется с использованием концептуальной модели, так как она позволяет проанализировать понятия в более обобщенных, переопределенных и абстрактных терминах. В результате выражения становятся более емкими, улучшается понимание и качество кода, уменьшается объем повторяемой информации.

Определение суперкласса является более общим, чем определение подкласса. Подклассы создаются в следующих случаях:

- подкласс имеет дополнительные атрибуты, интересующие разработчика;
- подкласс имеет дополнительные ассоциации, интересующие разработчика;
- подклассу соответствует понятие, управляемое, обрабатываемое, реагирующее или используемое способом, отличным от способа, определенного суперклассом или другими подклассами;
- подкласс представляет «живой» сущности поведение, которое отлично от поведения, определяемого суперклассом или другими подклассами.

Ассоциации показывают, что объекты одного класса связаны с объектами другого класса и отражают некоторое отношение между ними. В этом случае можно перемещаться от объектов одного класса к объектам другого.

С помощью простой, не содержащей дополнений ассоциации между двумя классами (скажем, Книга и Библиотека) может осуществлять навигацию между их объектами. Если явно не оговорено противное, то навигация по ассоциации может осуществляться в обоих направлениях. Однако бывают случаи, когда необходимо ограничить ее только одним направлением. При моделировании сервисов операционной системы можно столкнуться с ассоциацией между объектами User (пользователь) и Password (пароль). Если дан объект класса User, то нужно уметь находить соответствующие объекты класса Password, но обратное недопустимо, т. е. не должно быть возможности по паролю определить пользователя. Направление ассоциации можно выразить явно, дополнив ее стрелкой, указывающей на допустимое направление движения.

Если задано направление движения, это не обязательно означает, что Вы никогда не сможете добраться от объектов на одном конце ассоциации к объектам на другом ее конце. Навигация описывает, скорее, эффективность перемещения. Так, в приведенном выше примере все-таки можно найти пользователя, зная его пароль, с помощью других ассоциаций, не показанных на

этом рисунке. Возможность осуществлять навигацию в данном случае означает только то, что до объектов на другом конце ассоциации можно добраться легко и быстро (как правило, потому что в объекте - источнике хранятся ссылки на объекты - цели).

При наличии ассоциации между классами объекты одного класса могут «видеть» объекты другого и осуществлять навигацию к ним, если это не запрещено явным указанием односторонней навигации. Однако иногда необходимо ограничить видимость объектов, связанных ассоциацией, для объектов, внешних по отношению к ней. Например, между объектами UserGroup (группа пользователей) и User существует одна ассоциация, а другая – между объектами User и Password. Зная пользователя, можно найти его пароль, но эта информация принадлежит пользователю и не должна быть доступна извне (если, конечно, объект User явно не даст доступ к объекту Password, возможно, посредством открытой операции). Поэтому, как следует из рисунка, можно осуществлять навигацию от объекта UserGroup к входящим в нее объектам User и обратно, но из объекта UserGroup нельзя видеть объекты Password, принадлежащие отдельным объектам User.

В языке UML можно описать три уровня видимости для концевой точки ассоциации подобно тому, как это делается для классов. Для этого достаточно добавить символ видимости к ролевому имени. Если явно не оговорено противное, видимость роли устанавливается открытой. Закрытая видимость означает, что объекты на соответствующем конце недоступны для внешних по отношению к ассоциации объектов. Защищенная видимость показывает, что объекты на соответствующем конце недоступны внешним объектам, за исключением тех, что являются потомками объектов на противоположном конце ассоциации.

Одной из наиболее распространенных идиом в контексте ассоциаций является задача поиска. Как, зная объект на одном конце ассоциации, можно определить объект или группу объектов на другом ее конце? Рассмотрим для примера моделирование рабочего стола в мастерской, на котором сортируются возвращенные изделия, подлежащие ремонту. Для этого нужно смоделировать ассоциацию между классами РабочийСтол и ВозвращенноеИзделие. Применительно к РабочемуСтолу определен jobID – идентификатор задания, связанный с каждым конкретным ВозвращеннымИзделием. В этом смысле jobID – атрибут ассоциации. Он не является свойством объекта ВозвращенноеИзделие, поскольку изделия ничего не знают ни о ремонте, ни о заданиях. Если известен объект РабочийСтол и значение jobID, то можно осуществить навигацию к нулю или одному объекту ВозвращенноеИзделие. В языке UML эта идиома моделируется с помощью квалификатора, являющегося атрибутом ассоциации, значения которого разбивают множество объектов на подмножества, связанные с объектом на другом конце ассоциации. Квалификатор изображается в виде маленького прямоугольника, присоединенного к одной из концевых точек ассоциации, внутри которого располагаются атрибуты квалификатора. Объект – источник вместе со значениями атрибутов квалификатора порождает один

целевой объект (если краткость цели не больше единицы) или множества таких объектов (если краткость больше единицы).

Семантика спецификаторов нетривиальна, и существует ряд сложных примеров их использования. Однако чаще всего ситуации, в которых нужны спецификаторы, довольно просты. Если на одном конце ассоциации Вы можете поместить поисковую структуру данных (например, хэш-таблицу или B-дерево), то объявите индекс, по которому производится поиск как квалификатор. Обычно кратность на исходном конце будет «много», а на целевом – 0.1.

Интерфейсом называется набор операций, которые используются для спецификации услуг, предоставляемых классом или компонентом, причем один класс может реализовать несколько интерфейсов. Перечень всех реализуемых классом интерфейсов образует полную спецификацию поведения класса. Однако в контексте ассоциации с другим целевым классом исходный класс может не раскрывать все свои возможности. Так, в словаре системы управления человеческими ресурсами класс Person (Сотрудник) способен реализовать несколько интерфейсов: IManager (управляющий), IEmployee (Работник), IOfficier (Служащий) и т.д. Отношения между начальником и подчиненными можно моделировать с помощью ассоциации типа «один ко многим», явно пометив роли в ней как supervisor (контролер) и worker (рабочий). В контексте данной ассоциации объект класса Person в роли supervisor предоставляет объектам worker только интерфейс Imanager; он же в роли worker предоставляет объекту supervisor лишь интерфейс Iemployee.

Некоторые свойства ассоциации и их разновидности:

- *A* является физической частью *B*;
- *A* является логической частью *B*.

Рисунок 12 – Отношение ассоциации.

Агрегация (композитная и коллективная)

Агрегирование – ассоциация, моделирующая взаимосвязь «часть-целое» между сущностями, которые в то же время могут быть равноправными.

Оба класса при этом находятся на одном концептуальном уровне, и ни один не является более важным, чем другой. Применяется для моделирования отношений типа «часть-целое», в котором один из классов имеет более высокий ранг (целое называется композитным объектом) и состоит из нескольких мелких по рангу частей, однако его части стандартного имени не имеют.

Композитная агрегация (объединение или композиция) означает, что со стороны составного объекта кратность может быть не более единицы. Составной объект является владельцем своих частей, и эти части можно расположить в иерархии древовидной структуры.

Рисунок 13 – Композиция.

Коллективная агрегация означает, что со стороны составного объекта кратность может быть больше единицы.

Рисунок 14 – Коллективная агрегация.

- Время жизни компонента ограничено временем жизни составного объекта, т. е. между частью и целым существует зависимость создания – удаления.
- В физическом или логическом агрегате очевидно наличие отношений «часть-целое».
- Некоторые свойства составного объекта распространяются и на его компоненты, например место их расположения.
- Операции, применяемые к составному объекту, осуществляются и над его частями, например разрушение, перемещение и запись.

Роли

Каждый конец ассоциации называется ролью характеристики. Роль определяется именем, кратностью, направлением связи.

Кратность – указывает, сколько экземпляров класса *A* может быть ассоциировано с одним экземпляром класса *B* в определенный момент.

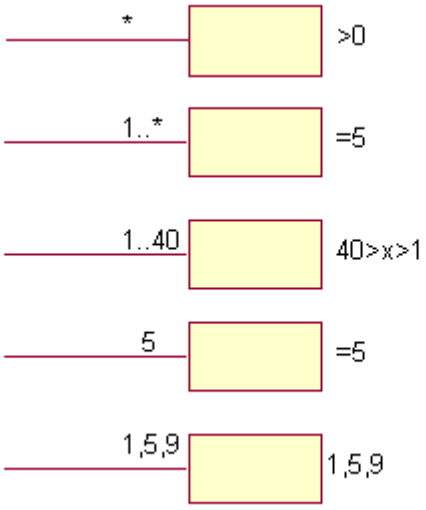


Рисунок 15 – Кратность.

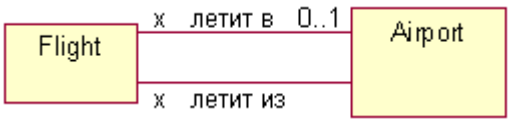


Рисунок 16 – Пример на кратность.

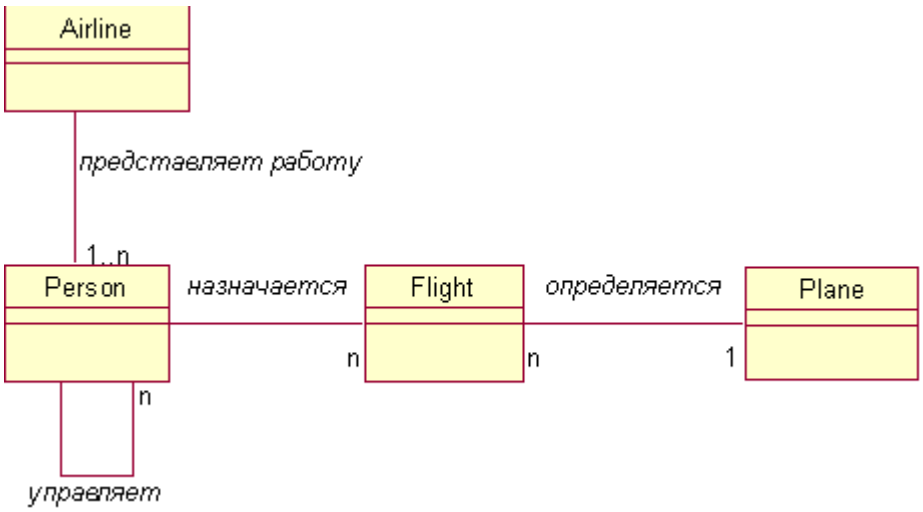


Рисунок 17 – Диаграмма классов с заданной кратностью.

Реализацией (Realization) называется семантическое отношение между классификаторами, при котором один из них описывает контракт, а другой гарантирует его выполнение. Реализация изображается в виде пунктирной линии с большой не закрашенной стрелкой, указывающей на классификатор, который определяет контракт.

Реализации настолько отличаются от зависимостей, обобщений и ассоциаций, что выделяются в особый тип отношений. Семантически реализация – это нечто среднее между обобщением и зависимостью, и нотация для нее несет в себе черты того и другого.

Реализация употребляется в двух ситуациях – в контексте интерфейсов и в контексте кооперации.

Чаще всего реализации используют для определения отношений между интерфейсом и классом или компонентом, который предоставляет объявленные в интерфейсе операции или услуги. Интерфейс – это набор операций, которые применяются для спецификации услуг, предоставляемых классом или компонентом. Таким образом, интерфейс определяет обязательства, которые должен выполнять компонент или класс. Один интерфейс может реализовываться несколькими классами или компонентами, а класс или компонент может реализовывать несколько интерфейсов. Пожалуй, самой главной особенностью интерфейсов является то, что они позволяют отделить спецификацию контракта (сам интерфейс) от их реализации (классом или компонентом). Кроме того, интерфейсы могут пересекать границу между логическими и физическими частями системной архитектуры.

При моделировании отношений в UML соблюдаются следующие правила:

- используется зависимость, если моделируемое отношение не является структурой;
- используются обобщения, только если место отношение типа «является»;
- множественное наследование практически всегда можно заменить агрегированием;
- не должно быть циклических отношений обобщения;

- иерархия наследования не должна быть ни слишком глубокой (<=5), ни слишком широкой (лучше применить абстрактные промежуточные классы или интерфейсы);
- ассоциации применяется там, где между объектами существуют структурные отношения.

1.2.3. Среда проектирования Rational Rose

Rational Rose – CASE-средство фирмы IBM – предназначено для автоматизации анализа и проектирования ПО, а также для генерации кодов на различных языках и создания проектной документации. Rational Rose использует методологию объектно-ориентированного проектирования, основанную на подходах трех ведущих специалистов в данной области: Буча, Рамбо и Джекобсона. Конкретный вариант Rational Rose определяется языком, на котором генерируются коды программ (C++, Java и т. д.).

Вариант – Rational Rose/Java – позволяет разрабатывать проектную документацию в виде диаграмм и спецификаций, а также генерировать программные коды на Java. Кроме того, Rational Rose содержит средства реинжиниринга программ, обеспечивающие повторное использование программных компонентов в новых проектах.

При разработке любой информационной системы в первую очередь возникает проблема взаимопонимания подрядчика и заказчика уже на стадии договоренности о структуре системы. Имея такой инструмент, как Rose, проектировщик (аналитик) всегда может показать заказчику не абстрактное словесное описание процесса, а его конкретную модель. Значит, Rational Rose позволит быстрее уточнить с заказчиком все детали планируемой системы. RUP описывает все артефакты (документы), возникающие по ходу проекта, так и в Rational Rose результатом моделирования является файл с моделью, которую проектировщик передает следующему звену сотрудников – кодировщикам, которые дополняют полученную логическую модель системы моделями конкретных классов на конкретном языке программирования.

1.2.4. Структура и функции

В основе Rational Rose лежит построение диаграмм и спецификаций, определяющих логическую и физическую структуры модели, ее статические и динамические детали. В их число входят диаграммы классов, состояний, сценариев, процессов.

В составе Rational Rose можно выделить шесть основных структурных компонентов: репозиторий, графический интерфейс пользователя, средства просмотра проекта (browser), средства контроля проекта, средства сбора статистики и генератор документов. К ним добавляются генератор кодов (индивидуальный для каждого языка).

Репозиторий представляет собой объектно-ориентированную базу данных. Средства просмотра обеспечивают «навигацию» по проекту, в том числе перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому и т. д. Средства контроля и сбора статистики дают возможность находить и устранять ошибки по мере развития проекта, а не после

завершения его описания. Генератор отчетов формирует тексты выходных документов на основе содержащейся в репозитории информации.

Средства автоматической генерации кодов программ на языке Java, используя информацию, содержащуюся в логической и физической моделях проекта, формируют файлы заголовков и файлы описаний. Создаваемый таким образом скелет программы должен быть уточнен путем прямого программирования на языке Java. Но обычно автоматическая генерация кода применяется только для получения SQL-кода создания базы данных.

Инструментарий программы допускает как высокоуровневое абстрактное представление, (например, схема автоматизации предприятия), так и низкоуровневое проектирование, например интерфейс программы, схема базы данных, частичное описание классов. Все это базируется на семи диаграммах, которые в зависимости от ситуации способны описывать различные действия.

Что может делать Rational Rose:

- проектировать системы любой сложности;
- давать развернутое представление о проекте в сочетании со средствами документирования (SoDA);
- проводить кодогенерацию;
- проводить обратное проектирование имеющихся систем;
- имеет открытый для дополнений интерфейс;
- поддержка языка UML;
- наличие средств автоматического контроля, в том числе проверки соответствия двух моделей;
- удобный для пользователя графический интерфейс;
- многоплатформенность;
- интегрировать с другими инструментальными средствами, поддерживающими жизненный цикл программных систем, в том числе со средством управления требованиями (Requisite Pro), тестирования, конфигурационного управления (ClearCase).

В результате разработки проекта с помощью CASE-средства Rational Rose формируются следующие документы:

Набор диаграмм:

- Use Case diagram (диаграммы прецедентов);
- Activity diagram (диаграммы описаний технологий, процессов, функций);
- Statechart diagram (диаграммы состояний);
- Class diagram (диаграммы классов);
- Sequence diagram (диаграммы последовательностей действий);
- Collaboration diagram (диаграммы взаимодействий);
- Component diagram (диаграммы компонент);
- Deployment diagram (диаграммы развертывания).

Тексты программ, полученных в результате генерации кода, являются заготовками для последующей работы программистов. Они формируются в рабочем каталоге в виде файлов типов .java (заготовки программ). Система

включает в программные файлы собственные комментарии. В дальнейшем эти исходные тексты развиваются программистами в полноценные программы.

Use case diagram используются для отображения списка операций, которые должна выполнять наша система; иначе говоря, это требования к системе. Каждый *Use case* – это некоторый процесс (последовательность действий), поэтому следует использовать *Sequence diagram* для его детализации. На этой диаграмме отображаются объекты из предметной области (объекты, участвующие в бизнес-процессе).

Таким образом, будут получены экземпляры некоторых классов и их взаимодействие. *Sequence diagram* отображает сам процесс, а *Class diagram* статическую картина взаимодействия объектов. Переходим к *Class diagram*, на которой изображаются классы нашей ИС. Далее классы объединяются в компоненты, которые отображаются на *Component diagram*, где показывается зависимость компонентов между собой. На *Deployment diagram* отображается размещение этих компонентов по компьютерам (узлам сети) для проектируемой ИС.

В итоге последовательность построения диаграмм следующая: *Use case diagram –> Activity diagram -> Statechart diagram -> Sequence diagram -> Class diagram -> Component diagram -> Deployment diagram.* Этот набор диаграмм должен присутствовать всегда в моделях RR, остальные типы диаграмм нам пригодятся для большей детализации нашей модели.

Существует список требований к информационной системе, составленный с помощью Requisite Pro.

Начнем создавать диаграмму прецедентов – *Use case diagram*, на которой отображаем взаимодействие между ролями (актерами) и прецедентами (т. е. это случаи использования ИС). Например, фраза «Заказчик формирует заказ на доставку товара» приводит к пониманию следующего: Актер – «Заказчик», прецедент – «Сформировать заказ на доставку» (прецедент обычно начинается с глагола), требование к системе – «ИС должна поддерживать операцию формирования заказа на доставку» (список требований ранее сформирован). Теперь необходимо изобразить всех актеров и те действия, которые они могут выполнять (т. е. прецеденты) на одной или нескольких *Use case diagrams,* а затем связать каждый прецедент с требованием, ранее сформированном в Requisite (R Pro (используйте контекстное меню на *Use case:* Requirement Properties --> Associate…).

Прецедент – это процесс, в котором обычно участвуют несколько объектов, т. е. “Сформировать заказ на доставку” предполагает некоторую последовательность операций: *открыть бланк заказа --> заполнить реквизиты заказчика --> заполнить тип и количество товара --> отправить заказ на исполнение (или отменить заказ или еще что-то)*. Вот этот процесс необходимо отобразить на *Sequence diagram.* На этих диаграммах слева всегда изображают одного Actor, далее стоят последовательно объекты (или ассоциированные с ними классы), а стрелочками отображается передача сообщения (или вызов метода класса).

Например, передача сообщение «Открыть бланк заказа» отображается в виде стрелки от актера «ЗАКАЗЧИК» к объекту «ЗАКАЗ» с соответствующей записью на диаграмме. При переходе на *Class diagram* следует создать класс COrder с методом New Order () для данного сообщения.

Проведенный анализ бизнес-процессов у заказчика должен отобразиться в диаграммных последовательностях действий – *Sequence diagrams.* Можно создать эту диаграмму для каждого прецедента (Open Specifications --> Diagrams --> Insert Sequence diagram) или общую на группу прецедентов (New --> Sequence diagram). Построить *Collaboration diagrams* (диаграммы взаимодействий) достаточно просто по *Sequence diagrams* (они также детализируют процесс взаимодействия, но не представляют его во времени): Browse --> Create Collaboration diagram (F5). Имея набор Sequence diagrams, можно переходить к проектированию классов и построению диаграмм классов, которые представляют собой статическую картину взаимодействия объектов.

Для каждого объекта необходимо создать класс, в котором будет реализовано поведение этого объекта через методы класса. Каждый объект в *Sequence diagram* должен быть ассоциирован с классом. Для этого открывается спецификация объекта и указывается класс, затем для каждого сообщения на *Sequence diagram* его имя заменяется на имя метода из класса. Для проверки модели следует выполнить Tolls --> Check Model.

Построение диаграмм классов (*Class diagrams*) является самым важным и трудоемким этапом в создании модели. Понятно, что поведение объекта невозможно встроить в один класс, поэтому необходимо создавать диаграммы классов, устанавливая связи между классами. Каждый класс имеет набор методов (Operations) и переменных (Attributes).

По диаграмме классов можно провести генерацию проекта, понятно, что имплементации методов не будет, только их определения.

Имея набор классов, можно перейти к формированию компонентов ИС – физических модулей ИС (DLL, EXE и др.). Зависимости между компонентами показываются на *Component diagram*. Каждый компонент ассоциирован с одним или несколькими классами, которые и определяют содержимое компонента.

Далее следует открыть спецификацию компонента и на вкладке «Realizes» назначьте классы для данного компонента (из контекстного меню «Assign»), а также на вкладке «General» указать язык программирования (теперь спецификация классов дополнится новыми вкладками, влияющими на кодогенерацию для данного языка программирования).

Deployment diagram построить достаточно просто, так как она не содержат привязки к компонентам в модели RR, т. е. не обязательно строить эту диаграмму на последнем этапе проектирования. На этой диаграмме изображаются компьютеры (Processor) и связи между ними. Чтобы отобразить задачи, которые выполняются на этих компьютерах (узлах сети) следует открыть спецификацию и на вкладке «Detail» для поля «Processes» ввести имена задач, которые могут соответствовать именам компонентов проектируемой ИС.

Процесс построения модели завершен. Следует выполнить проверку модели с помощью команды Tolls --> Check Model и посмотреть результаты в Log-окне. Если ошибок нет, можно переходить к генерации проекта и имплементации ИС.

Процесс построения модели носит итерационный характер. Невозможно построить модель сразу, часто необходимо возвращаться на другие уровни представления информации, вносить изменения, детализировать и снова возвращаться на уровень имплементации (диаграммы классов уровня генерации кода). К тому же ИС подвержена постоянному изменению (причины Вам известны), поэтому и код ИС, и ее модель должны меняться согласованно и соответствовать друг другу.

1.2.5. Диаграммы вариантов использования

В качестве примера рассмотрим построение модели банковской системы. Начинаем с описания функциональных возможностей системы. Для этого служит *диаграмма вариантов использования* (Use Cases). На представленной диаграмме показаны три действующих лица и шесть основных действий, выполняемых моделируемой системой. Взглянув на варианты использования, клиенты поймут, какие функциональные возможности будут заложены в систему. Часто для одной системы создаются несколько таких диаграмм. Конечная цель этих диаграмм – документирование вариантов использования, действующих лиц и связей между ними.

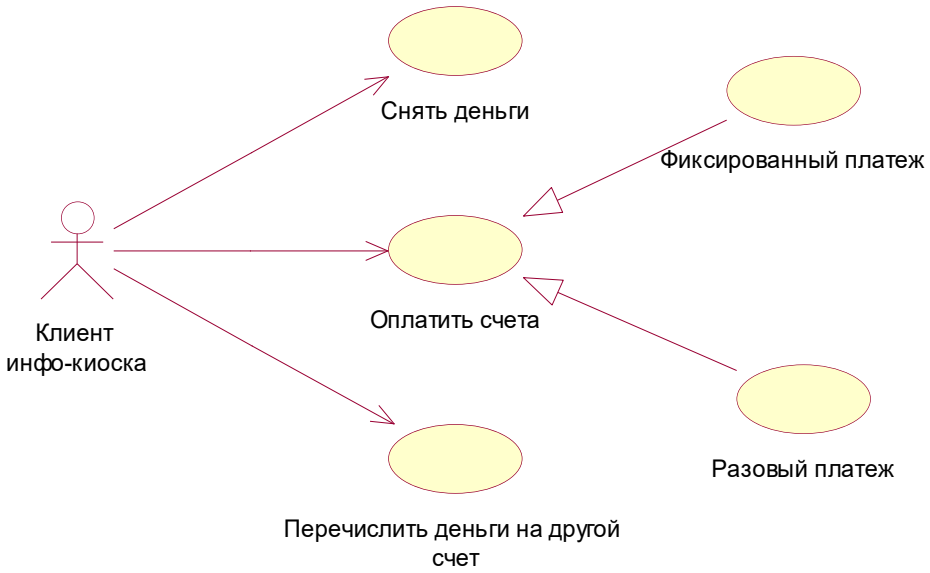


Рисунок 18 – Диаграмма прецедентов.

Для создания диаграммы необходимо:

- определить перечень главных пользователей (внешних входов-выходов) системы, это целесообразно сделать методом мозгового штурма с участием большого количества людей, начиная от руководителя разработки и кончая программистами;
- определить методику участия внешних пользователей в проекте (методы привлечения их к составлению требований);

- определить главные цели пользователей (что они хотят сделать с помощью системы) и задачи, которые нужно решить для достижения этих целей;
- составить диаграмму использования, в которой каждой задаче каждого пользователя сопоставить блок использования; соединить эти блоки с блоками внешних пользователей, ответственных за данную задачу;
- проверить каждый вариант использования какой-либо количественной метрикой (например, можно использовать оценку времени выполнения задачи или объема ресурсов, необходимого для ее решения); здесь же полезно составить прототип программного средства для оценки реализуемости проекта (но ни в коем случае не интерфейса пользователя).

Все перечисленные шаги не должны приводить к слишком сложной модели и занимать слишком много времени. Далее можно перечислить рекомендации по составлению диаграмм использования:

диаграммы должны быть простыми и понятными любому пользователю;

автор проекта и идеи системы должен участвовать в спецификации требований;

каждый участник проекта должен понимать конечную цель и предназначение разрабатываемой системы;

блоки использования должны быть основаны на их целях (а, например, не на структуре);

не использовать слишком много отношений «extends» и «uses», так как они усложняют понимание диаграммы;

блоки использования не должны отражать вопросов пользовательского интерфейса;

необходимо вовремя остановить внешних пользователей, так как они могут хотеть от системы слишком много, и она будет нереализуема.

1.2.6. Диаграммы последовательностей

Для каждого потока варианта использования составляется *диаграмма последовательности действий* (Sequence). Перед созданием диаграммы требуется выявить все объекты, действующие в конкретном сценарии.

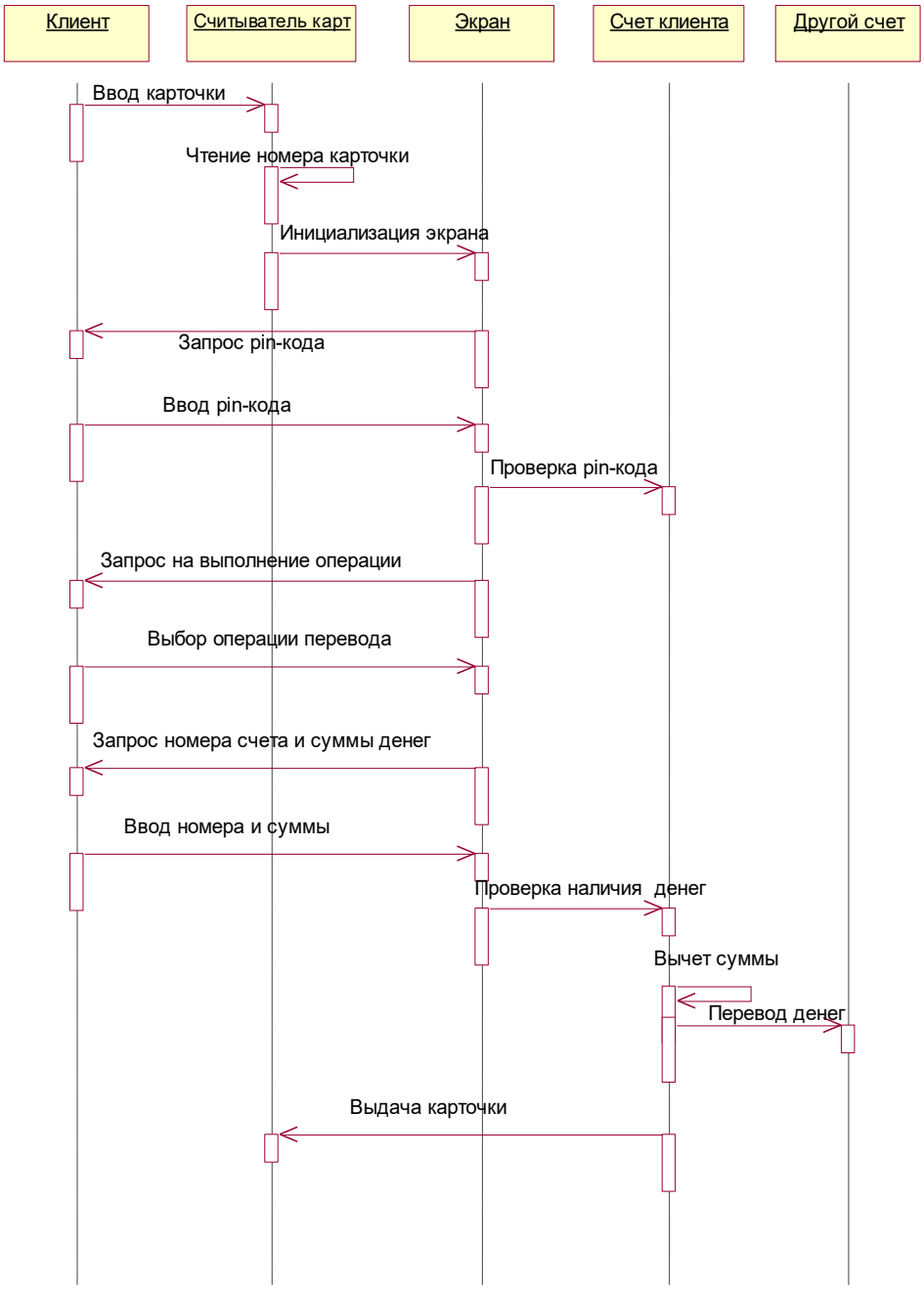


Рисунок 19 – Диаграмма последовательности.

Участвующие в потоке объекты нарисованы в прямоугольниках в верхней части диаграммы. На диаграмме приведена реализация варианта использования. *Перевести Деньги*. Объект действующего лица, инициирующий вариант использования, показан в верхнем левом углу для границ.

У каждого объекта имеется линия жизни (lifeline), изображаемая в виде вертикальной штриховой линии под объектом. Сообщения, соответствующие связям между объектами, изображаются между линиями жизни объектов. Сообщение показывает, что один объект вызывает функцию другого. Сообщения могут быть рефлексивными, т. е. объект вызывает свою собственную функцию.

1.2.7. Диаграмма классов

Диаграммы классов являются ключевым инструментом проектирования. С их помощью разработчики могут увидеть, как планировать структуру системы еще до фактического написания кода. На диаграммах классов отображаются некоторые классы и пакеты систем, а также связи между ними.

Для моделирования статического вида используются системы с точки зрения проектирования. Однако основным результатом группы разработчиков являются не диаграммы, а программные обеспечения. Все усилия по созданию моделей направлены только на то, чтобы обеспечить написание программы. Важно, чтобы модели и основанные на них реализации соответствовали друг другу.

Модели могут быть преобразованы в код. В наибольшей степени это относится к диаграммам классов. Из пакета Rational Rose можно получить код на Java, C++, Oracle.

Для их создания необходимы:

- диаграммы взаимодействий, по которым разработчик определяет, какие классы должны быть задействованы в проектном решении, а также методы этих классов;
- концептуальная модель, на основе которой разработчик детализирует определения классов.

1.2.8. Классы

Классы используются для составления словаря разрабатываемой системы. Это могут быть абстракции, являющиеся частью предметной области, либо классы, на которые упирается реализация. С их помощью описываются программные, аппаратные или чисто концептуальные сущности.

Хорошо структурированные классы характеризуются четкими границами и помогают формировать сбалансированное распределение обязанностей в системе.

Классом (Class) называется описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Графически класс изображается в виде прямоугольника.

У каждого класса должно быть свое имя, отличающее его от других классов. Имя класса – это текстовая строка. Взятое само по себе оно называется простым именем; к составному имени впереди добавлено имя пакета, куда входит класс. Имя класса в объемлющем пакете должно быть уникальным, при графическом изображении класса показывается только его имя.

Диаграмма классов иллюстрирует спецификации программных классов и интерфейсов в приложении содержит следующую информацию:

- классы, ассоциации и атрибуты;
- интерфейсы со своими операциями и константами;
- методы;
- информацию о типах атрибутов;
- способы навигации;
- зависимости;

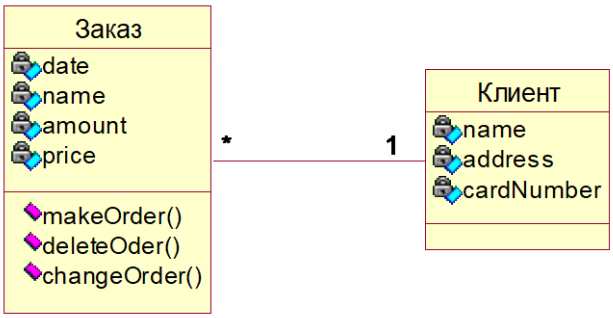


Рисунок 20 – Диаграмма классов.

В отличие от концептуальной модели, диаграммы классов отражают определения программных сущностей, а не понятия предметной области.

При построении диаграммы классов можно следовать следующим правилам:

определить все классы, задействованные в программном решении.

Для этого необходимо проанализировать диаграммы взаимодействий;

отобразить их на диаграмме классов;

перенести на диаграмму атрибуты соответствующих понятий из концептуальной модели;

добавить имена методов на основе анализа диаграмм взаимодействий;

добавить информацию о типах атрибутов и методов;

добавить ассоциации, необходимые для поддержки обеспечения видимости посредством атрибутов;

добавить линии зависимостей, определяющие другие способы обеспечения видимости, отличные от видимости посредством атрибутов.

Классификатор – механизм, описывающий структурные и поведенческие свойства. К числу классификаторов относятся классы, интерфейсы, типы данных, сигналы, компоненты, узлы, предметы и подсистемы.

Интерфейс – набор операций, используемых для специфицирования услуги, предоставляемой классом или компонентом.

Компонент – физически замещаемая часть системы, которая соответствует спецификации набора интерфейсов и обеспечивает их реализацию.

Такой инструмент, как Rational Rose, генерирует основу кодов классов, которую программисты заполняют деталями на выбранном ими языке. С помощью этих диаграмм аналитики могут показать детали системы, а архитекторы – понять ее проект.

Если какой-либо класс несет слишком большую нагрузку, и это видно на диаграмме классов, то можно переопределить нагрузку между другими классами.

Ниже приведена диаграмма классов для варианта использования. *Снять Деньги*. На основании диаграммы последовательности определены атрибуты и методы классов для данного варианта использования.

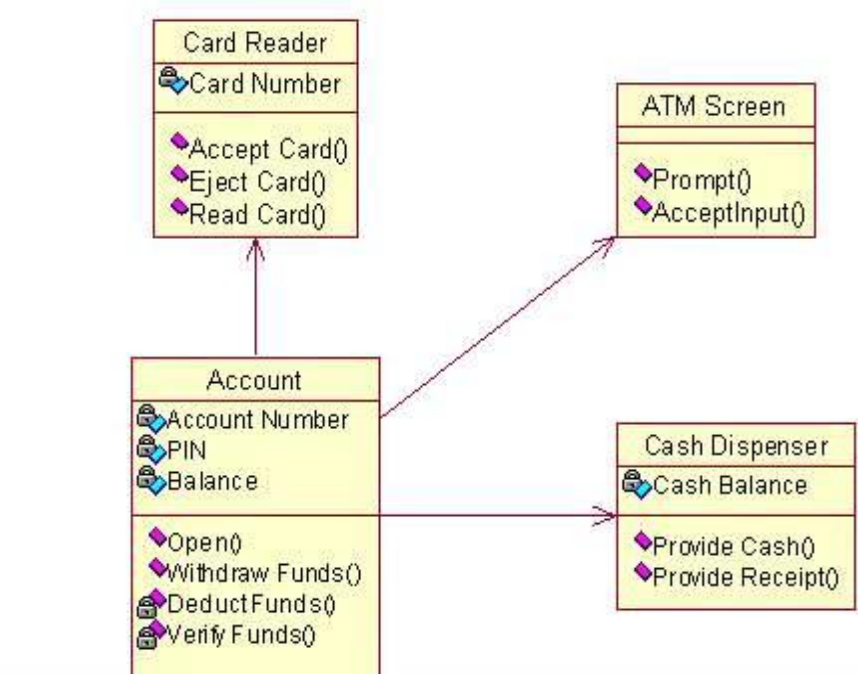


Рисунок 21 – Диаграмма классов.

1.2.9. Структурные отношения и конструирование базы данных

Отношения зависимости и обобщения применяются при моделировании классов, которые имеют различную значимость. При отношениях зависимости один класс зависит от другого, то тот может ничего не “знать” о наличии первого. При отношениях обобщения класс-потомок наследует от своего родителя, но сам родитель о нем не осведомлен. Таким образом, отношения зависимости и обобщения являются односторонними. Ассоциации предполагают участие равноправных классов и навигацию можно осуществлять в обоих направлениях. На диаграмме изображены классы вузовской информационной системы.

Рисунок 22 – Диаграмма классов.

Между классами **Студент** и **Курс** существует ассоциация: студент может посещать любое число курсов и на каждый курс может записаться любое количество студентов. Аналогичная ситуация между классом **Курс** и **Преподаватель**.

Отношения между классом **Вуз** и классами **Студент** и **Факультет** являются отношениями агрегирования. В вузе может быть любое количество студентов, и каждый студент может обучаться в одном или нескольких вузах. Можно было бы использовать простые ассоциации, определяя при этом **Вуз** как целое, а **Студент** и **Факультет** – как его части.

Между классами **Факультет** и **Преподаватель** установлены две ассоциации. Одна показывает, что каждый преподаватель работает на одном или нескольких факультетах, другая, что каждым факультетом управляет только один преподаватель – декан.

Рассмотрим расширение диаграммы, которое содержит достаточное количество деталей для конструирования физической базы данных. Все пять классов являются устойчивыми (persistent), т. е. их экземпляры должны содержаться в базе данных. Приведены атрибуты всех классов, которые являются примитивными типами. Два класса (**Вуз** и **Факультет**) содержат несколько операций, позволяющих манипулировать их частями. Эти операции включены для поддержания целостных данных.

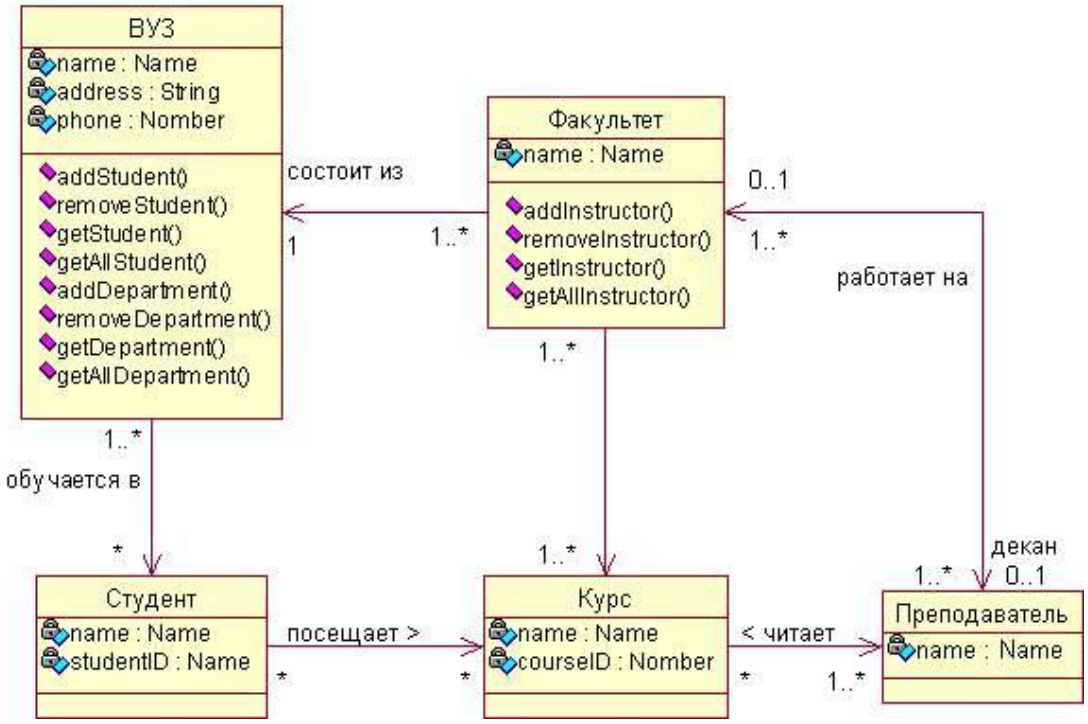


Рисунок 23 – Диаграмма классов.

1.2.10. Диаграммы деятельности *(Activity)*

С помощью диаграммы деятельности моделируются последовательные шаги варианта использования или процесса: жизнь системы или экземпляра, при переходе из одного состояния в другое в разных точках потока управления. Используются самостоятельно для конструирования динамики процесса.

Действие может заключаться в вызове другой операции, передаче сообщения, создании или уничтожении объекта, установке или разрыве соединения.

Диаграмма деятельности состоит из состояний деятельности, состояний действия (атомарные вычисления и действия), переходов и объектов.

Должны выполняться следующие условия:

- состояния действия не могут быть подвергнуты декомпозиции;
- выполняемое действие не может быть прервано;
- состояние деятельности может быть подвержено декомпозиции.

Ветвление и соединение

В модель можно включать ветвление, которое описывает различные пути выполнения в зависимости от значений сторожевых (как правило, булевских) выражений. Эти выражения должны быть взаимоисключающими. В точку ветвления может входить ровно один переход, а выходить два или более. Ветвление всегда должно заканчиваться соединением, которое имеет несколько входящих переходов и единственный выход.

Разделение и слияние

Параллельные потоки характерны для моделирования бизнес-процессов. Разделение характеризуется одним входящим переходом и несколькими выходящими, которые выполняются параллельно. Другими словами, последовательность работы потоков может быть произвольной. Должен поддерживаться баланс между точками разделения и слияния. Число потоков, исходящих из точки разделения, должно быть равно числу потоков, приходящих в соответствующую точку слияния.

Дорожки - *разновидность пакетов, описывающих связанную совокупность работ. Дорожка представляет собой сферу ответственности за часть всей работы, изображенной на диаграмме, и может быть реализована одним или несколькими классами. На диаграмме деятельности, разбитой на дорожки, каждая деятельность принадлежит только одной дорожке, но переходы могут пересекать границы дорожки.*

Диаграмма деятельности определяет правила последовательности действий, необходимые для реализации описываемого поведения. Ее можно использовать для моделирования некоторого динамического аспекта системы можно в контексте любого элемента модели. Чаще всего применяется в контексте поведения:

- системы в целом;
- любого уровня вида системной архитектуры;
- операции;
- компонента;
- узла;
- интерфейса;
- класса.

Применение:

- моделирование рабочего процесса, если рассматривается деятельность с точки зрения актеров, которые сотрудничают с системой. Рабочие процессы оказываются с обращенной к пользователю стороны программной системы и используются для визуализации, специализации и конструирования бизнес-процессов;

• моделирование операций – должны быть использованы как блок-схемы для моделирования деталей вычислений. Для такого применения важно моделирование точек ветвления, разделения и слияния.

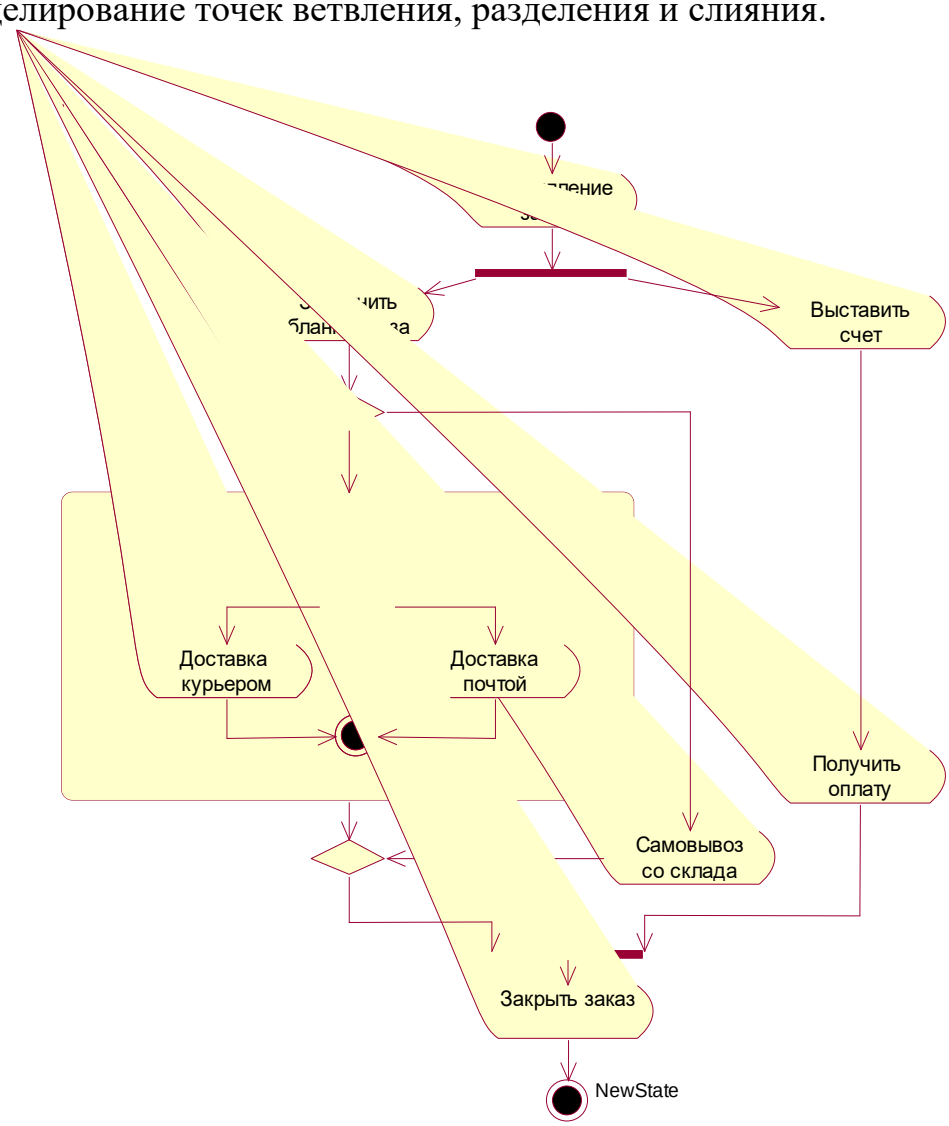


Рисунок 24 – Диаграмма деятельности.

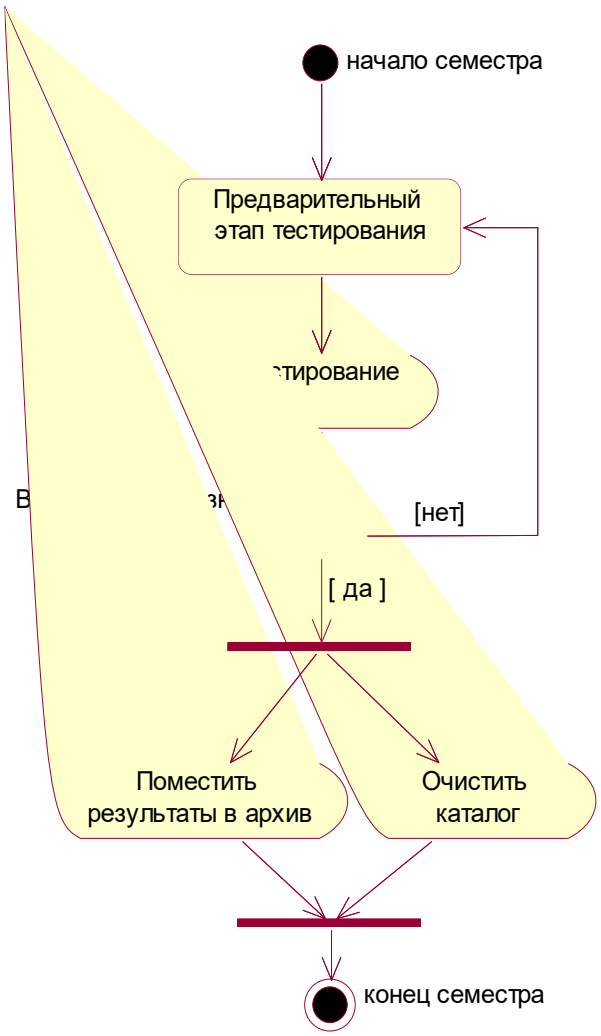


Рисунок 25 – Диаграмма деятельности.

1.2.11. Диаграммы состояний *(Statechart)*

Состояние (state) – это некое положение в жизни объекта, при котором он удовлетворяет определенному условию, выполняет некоторое действие или ожидает события. Состояние объекта можно описать с помощью значений одного или нескольких атрибутов класса.

Состояния автомата соответствуют состояниям объектов, в которых объект удовлетворяет некоторому условию, выполняет некоторое действие или ожидает некоторого события. Объект может находиться в каждом состоянии в течение конечного времени. Каждому состоянию может соответствовать вложенный автомат. Состояние изображается как прямоугольник со скругленными краями. Каждое состояние имеет две части. В верхней части отображается имя состояния, которое может быть пустым. Это так называемое анонимное состояние. Все анонимные состояния отличаются друг от друга. Нижняя часть предназначена для отображения внутренних действий, выполняемых в ответ на определенные события, возникающие, когда автомат-объект находится в данном состоянии, без смены текущего состояния. Описание внутренних действий имеет следующий формат:

имя события, список параметров, [условие] / действие.

Каждое имя события должно быть уникальным в пределах одного состояния. Определены зарезервированные имена событий:

On Entry – действие, выполняемое при входе в состояние.

On Exit – действие, выполняемое при выходе из состояния.

Эти два предопределенных события не имеют списка параметров и условия возникновения, так как и то и другое предопределено.

Введено специальное ключевое слово do, обозначающее вызов вло-женного автомата:

Do – имя автомата (список параметров).

Имя автомата задает конкретный автомат, которому при инициализации может быть передан список параметров, который должен быть совместимым со списком параметров этого автомата. Графические обозначения различных классов состояний автомата приведены в таблице ниже:

Событие (On Event) –любое действие, имеющее значение с точки зрения смены состояний автомата. Определены следующие виды событий (одно событие может относиться одновременно к нескольким видам):

- условие становится истинным;
- прием внешнего сигнала от одного объекта к другому;
- запрос на выполнение метода объекта;
- истечение заданного периода времени.

Событие, отмечающее переход, может иметь параметры, которые доступны внутри действий, сопоставленных переходу или тому состоянию, в который ведет переход. События, определяющие выполнение переходов, обрабатываются по одному в один момент времени. Если событие не приводит к выполнению ни одного перехода, то оно игнорируется. Переходы изображаются линией с указанием направления, и дополнительной текстовой информации.

Составные переходы между состояниями предназначены для отражения операций распараллеливания и синхронизации. Составной переход может иметь несколько входных и несколько выходных состояний и может выполнять только тогда, когда все его входные состояния активны. После выполнения составного перехода все его выходные состояния становятся активными, а все входные перестают быть активными. Таким образом, автомат может одновременно находиться в нескольких состояниях. Составной переход изображается как прямоугольник, входные и выходные линии определяют входные и выходные состояния:

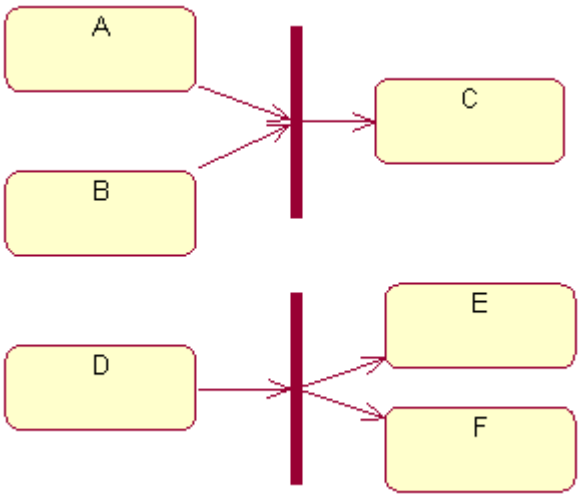


Рисунок 26 – Переходы между состояниями.

На данном рисунке изображены два составных перехода: первый имеет два входных состояния A, B и одно выходное стояние C; второй одно входное состояние D и два выходных стояния E и F.

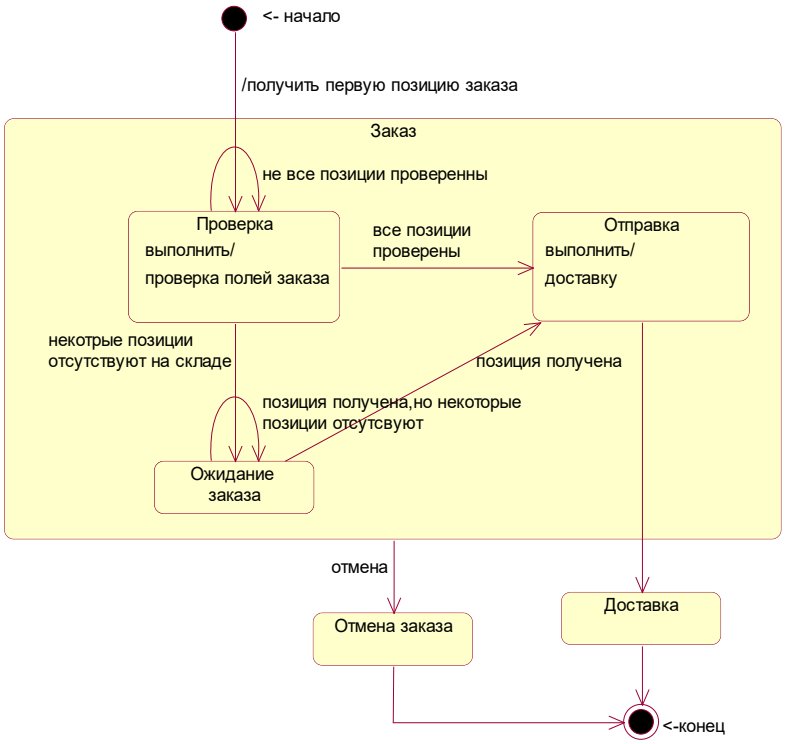


Рисунок 27 – Диаграмма состояний.

1.2.12. Шаблоны GRASP

Шаблоном называют описание проблемы и ее решение, которое можно применить при разработке других систем. Шаблон должен содержать советы по поводу его применения в различных ситуациях. Это именованные пары утверждений «проблема-решение», описывающие важные принципы, как правило, связанные с распределением обязанностей.

GRASP – общие шаблоны распределения обязанностей в программных системах.

Рассмотрим следующие шаблоны GRASP:

- Expert,
- Creator,
- High Cohesion,
- Low Coupling,
- Controller,
- Façade,
- Model – View Separation,
- Publish – Subscribe.

Прежде чем приступить к рассмотрению шаблонов, следует рассмотреть процедуры, предшествующие созданию кода системы.

1.2.13. Концептуальная модель

Разработчики должны разработать объектную бизнес-модель, состоящую из сотрудников, *бизнес-сущностей* и рабочих моделей, которые совместно реализуют бизнес-варианты использования.

Рассмотрим концептуальную модель в виде набора структурных статических диаграмм, на которых не определены никакие операции. Существует строгое соответствие понятиям предметной области, а не программирования.

Концептуальная модель может отражать понятия, ассоциации между понятиями и атрибуты понятий.

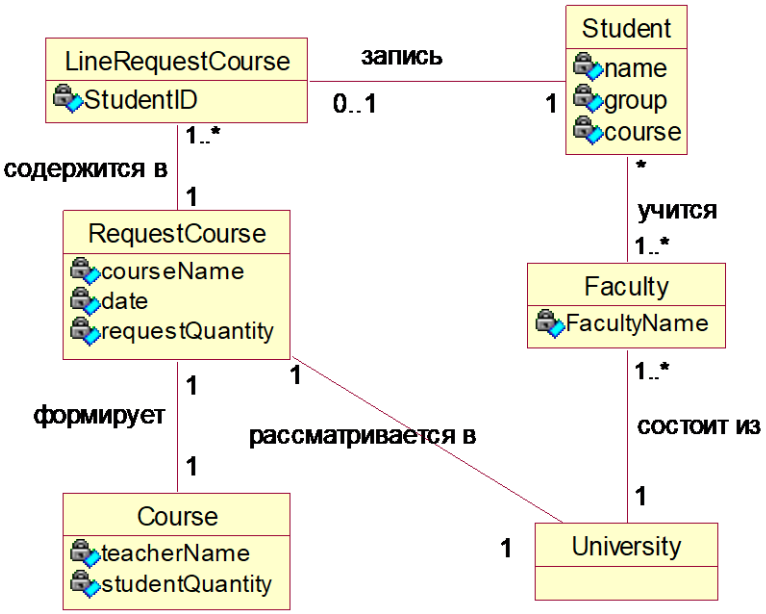


Рисунок 28 – Концептуальная модель.

Это не модель структуры программы, следовательно, здесь не приводятся обязанности и методы, не используются артефакты программирования: БЗ, окно и т. д.

Декомпозиция – стратегия борьбы со сложностью проблемы и ее разделение на мелкие составные части. Задача разбивается на понятия.

Основной задачей на стадии анализа является идентификация различных понятий из предметной области. Например: Факультет, ВУЗ, Заявка на прохождение курса.

Лучше излишне детализировать концептуальную модель, чем не доопределить ее.

Часто на начальной стадии идентификации некоторые понятия упускаются из виду, а появляются позднее при рассмотрении атрибутов и ассоциаций или даже на стадии проектирования. Обнаруженные новые понятия добавляются в концептуальную модель.

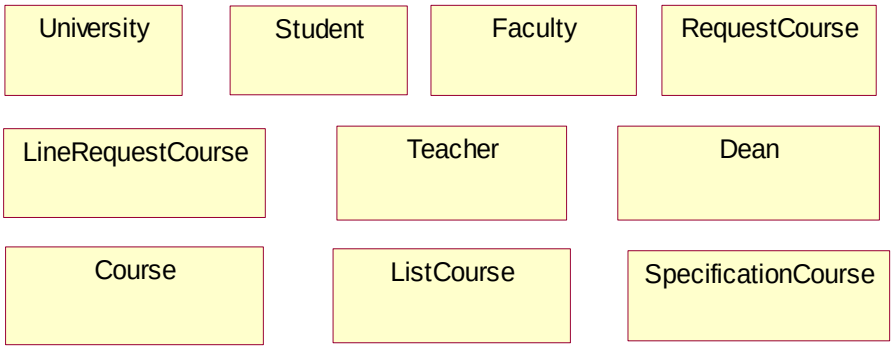


Рисунок 29 – Модель предметной области.

Объекты описаний, или спецификации, связаны с описываемыми понятиями. Связь между ними принято обозначать следующим образом:

XSpecification описывает X.

Еще один пример спецификации:

Рисунок 30 – Модель спецификации.

Предположим, все рейсы авиакомпании отменены на некоторое время из-за авиакатастрофы, т. е. все программные объекты были удалены из памяти после отмены рейсов.

Таким образом, после отмены рейсов не останется никакой информации об осуществляемых ею рейсах. Решение этой проблемы – ввод спецификации.

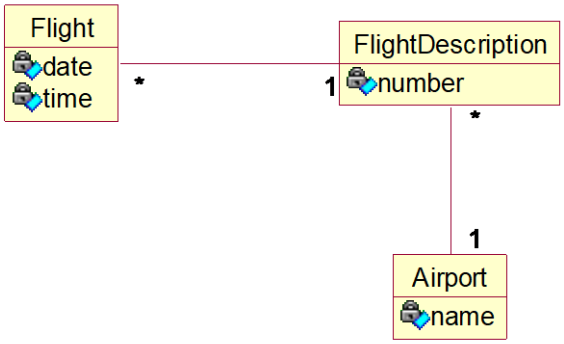


Рисунок 31 – Модель взаимодействия классов.

1.2.14. Атрибуты

Атрибут – это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число атрибутом или не иметь их вовсе. Атрибут представляет некоторое свойство моделируемой сущности. Общее для всех объектов данного класса. Например: у любой стены есть высота, ширина и толщина; при моделировании клиентов можно задать фамилию, адрес, номер телефона и дату рождения. Таким образом, атрибут является абстракцией данных объекта или его состояния. В каждый момент времени любой атрибут объекта, принадлежащего данному классу, обладает вполне определенным значением. Атрибуты представлены в разделе, который расположен под именем класса, при этом указываются только их имена.

Имя атрибута, как и имя класса, может быть произвольной текстовой строкой. На практике для наименования атрибута используется одно или несколько коротких существительных, соответствующих некоторому свойству объемлющего класса. Каждое слово в имени атрибута. Кроме самого первого, обычно пишется с заглавной буквы, например name или maxPrice.

В концептуальную модель включаются те атрибуты, которые определяются требованиям или выполнением варианта использования. Некоторые сущности удобнее представлять не в виде атрибутов, а в форме ассоциаций.

Типы большинства простых атрибутов рассматриваются обычно как примитивные типы данных. В концептуальной модели атрибуты должны быть простыми (simple attributes) или непримитивными простыми (pure date values).

Стандартные типы простых атрибутов: Simple – Date, Number, String, Text, Boolean.

Простыми данными являются Address, Color, ZIP, UPC, Point.

Сложные понятия предметной области не должны быть атрибутами.

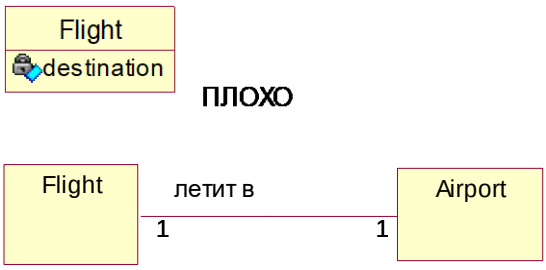


Рисунок 32 – Модель взаимодействия классов.

Тип данных, изначально считающийся примитивным (такой, как число или строка), может быть представлен в виде непримитивного типа данных в следующих случаях:

- если он составлен из отдельных частей (номер телефона, номер кредитной карты, имя человека, универсальный код товара UPC);
- если с этим типом обычно ассоциируются операции, такие как синтаксический анализ или проверка (номер кредитной карты, страхового полиса);

- если этот тип используется для задания количества с единицами измерения (цена товара).

Применение этих рекомендаций к атрибутам концептуальной системы приводят к формулировке, например, следующих требований:

- UPC должен быть не примитивным типом upc, так как он может использоваться при подсчете контрольной суммы или иметь другие атрибуты (например, код изготовителя);
- price, amount должны иметь тип Quantity (количество), поскольку они представляют собой количество, выраженное в денежных единицах.
- Address должен относиться не к примитивному типу Address, так как в нем имеются отдельные разделы.
- цвет должен иметь тип Color — здесь цвет является именем атрибута, Color — именем типа данного атрибута. Указанная запись может определять традиционно используемую RGB-модель (красный, зеленый, синий) для представления цвета. В этом случае имя типа Color как раз и характеризует семантическую конструкцию, которая применяется в большинстве языков программирования для представления цвета;
- имя_сотрудника должен иметь тип String — здесь имя_сотрудника является именем атрибута, который служит для представления информации об имени, а возможно, и отчестве конкретного сотрудника. Тип атрибута String как раз и указывает на тот факт, что отдельное значение имени представляет собой строку текста из одного или двух слов (например, «Кирилл» или «Дмитрий Иванович»). Поскольку во многих языках программирования существует тип данных String, использование соответствующего англоязычного термина не вызывает недоразумения у большинства программистов;
- видимость должен иметь тип Boolean — здесь видимость есть имя абстрактного атрибута, который может характеризовать наличие визуального представления соответствующего класса на экране монитора. В этом случае тип Boolean означает, что возможными значениями данного атрибута является одно из двух логических значений: истина (true) или ложь (false). При этом значение истина может соответствовать наличию графического изображения на экране монитора, а значение ложь — его отсутствию, о чем дополнительно указывается в пояснительном тексте.

1.2.15. Системные события и операции

Системное событие – это внешнее событие, инициированное для системы актером. Событие активизирует выполнение определенной операции.

Системная операция – операция, которую система выполняет в ответ на сгенерированное событие. Если администратор генерирует системное событие add Course (), то это приводит к выполнению системной операции add Course (). Имена события и операции совпадают. Операции могут быть сгруппированы как операции типа с именем System.

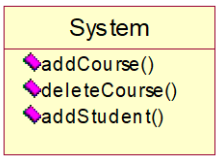


Рисунок 33 – Описание класса.

Представление типа System существенно отличается от объектов концептуальной модели, элементы которой иллюстрируют понятия реального мира, а тип System – искусственно создается конструкция. Кроме того, вместе с данным типом показаны и операции, чего не было ранее.

1.2.16. Диаграммы взаимодействий

На диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации. Дополнительно могут быть изображены динамические связи – потоки сообщений. Они представляются также в виде соединительных линий между объектами, над которыми располагается стрелка с указанием направления, имени сообщения и порядкового номера в общей последовательности инициализации сообщений.

Диаграммы кооперацій иллюстрируют взаимодействие объектов в формате графа или сети:

Диаграмма последовательностей, соответствующая этой диаграмме, имеет следующий вид:

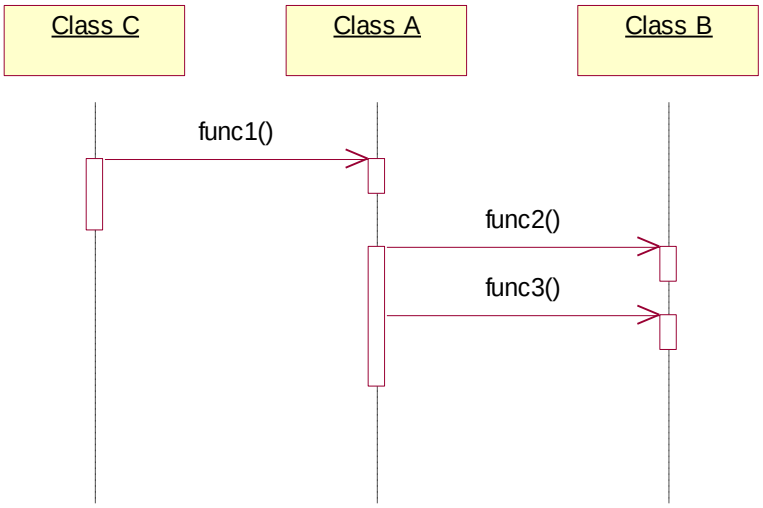


Рисунок 34 – Диаграмма последовательностей действий.

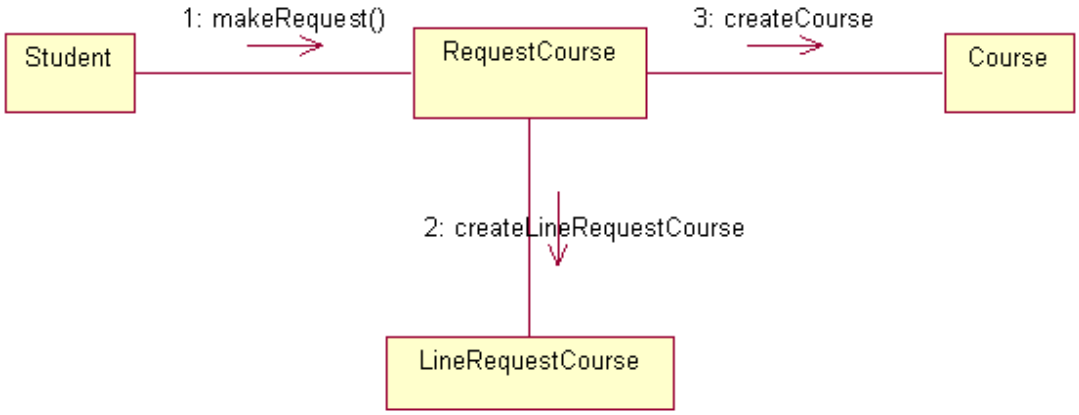


Рисунок 35 – Диаграмма взаимодействий.

1. Сообщение make Request () передается экземпляру объекта RequestCourse. Оно соответствует системной операции make Request ().
2. На основании сообщения make Request () объект RequestCourse создает экземпляр объекта LineRequestCourse.
3. Объект RequestCourse создает экземпляр объекта Course.

При распределении обязанностей и разработке способов взаимодействия объектов разработчику предоставляется достаточно свободы действий. Неудачный выбор может привести к тому, что системы и их отдельные компоненты окажутся непригодными для поддержки понимания, повторного использования и расширения. Некоторые из принципов объектно-ориентированного проектирования, применяемого при создании диаграммы взаимодействия и распределения обязанностей, систематизируют в шаблонах GRASP.

1.2.17. Поведение системы

Исследуем поведение ПО и определим систему как «черный ящик». Поведение системы представляет собой описание того, какие действия выполняет система, без определенного механизма их реализации.

Одной из составляющих такого описания являются диаграммы последовательностей. Назначение данной диаграммы – отображение событий, передаваемых исполнителями системе через ее границы.

1.2.18. Обязанности и методы

Обязанность – контракт или обязательство типа или класса. Обязанности описывают поведение объекта. Можно выделить два типа обязанностей знание (knowing) и действие (doing).

Обязанности, относящиеся к действиям объектов:

- выполнение действий самим объектом;
- инициация действий других объектов;
- управление и координирование действий других объектов.

Обязанности, относящиеся к знаниям объекта:

- наличие информации о закрытых инкапсулированных данных;
- наличие информации о связанных объектах;
- наличие информации о следствиях или вычисляемых величинах.

Обязанности присваиваются объектам в процессе ООП. Можно сказать, что «объект RequestCourse отвечает за вывод информации о себе самом (действие)» или «объект RequestCourse отвечает о наличие информации о себе самом (знание)». Возможность реализации обязанностей в виде классов и методов зависит от точности их описаний. Реализация обязанности «обеспечить доступ к базе данных» – потребует создания десятков классов и сотен методов, а для реализации обязанности «вывести информацию о продаже» достаточно одного или несколько методов. Между методами и обязанностями нельзя ставить знак равенства – реализация метода обеспечивает выполнение обязанностей. Обязанности реализуются посредством методов, действующих либо отдельно, либо во взаимодействии с другими методами и объектами.

1.2.19. Видимость объектов

Чтобы объект-отправитель мог отослать сообщение объекту-получателю, отправитель должен быть виден получателю, т. е. отправитель должен содержать некую ссылку на объект получателя.

Для отправки сообщения specification от объекта University объекту ListCourse необходимо обеспечить видимость экземпляра объекта ListCourse для экземпляра объекта University.

Понятие области видимости связано с понятием контекста (или области действия): попадает ли один ресурс (например, экземпляр объекта) в контекст другого?

Основных способа обеспечения видимости объекта B объектом A: по-

средством атрибутов. Объект B является атрибутом объекта A; посредством параметров. Объект B является параметром метода объекта A;

локальная видимость. Объект B объявлен как локальная переменная в методе объекта A; глобальная видимость. Объект B каким-то образом виден глобально.

Видимость посредством атрибутов *поддерживается до тех пор, пока существуют сами объекты.*

При видимости посредством параметров *объект B передается в качестве параметра метода объекта A. Существует только в контексте данного метода. Процесс передачи сообщения make LineRequest () экземпляру объекта RequestCourse, когда экземпляр объекта CourseSpecification передается в качестве параметра. В контексте метода make LineRequest () между объектами RequestCourse и CourseSpecification существует видимость, обеспечиваемая с помощью параметров.*

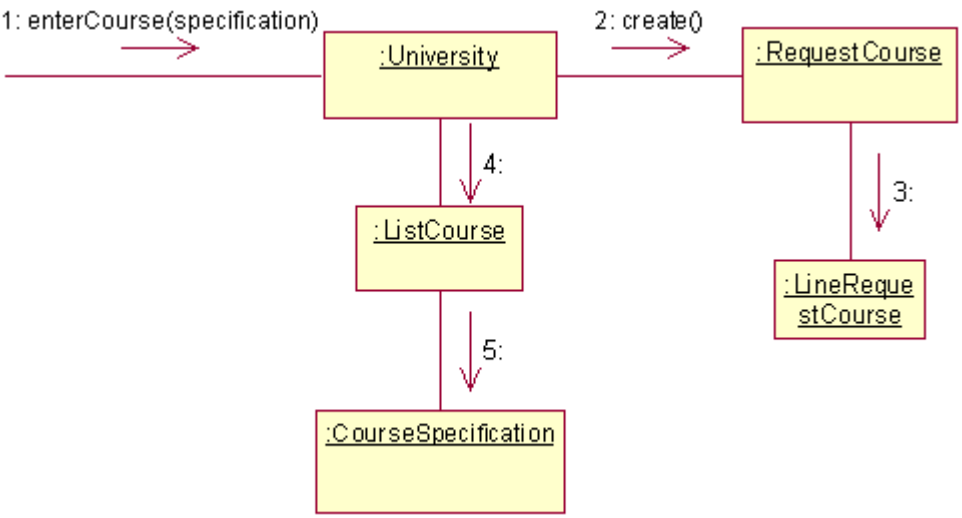


Рисунок 36 – Диаграмма взаимодействий.

Видимость, обеспечиваемую параметрами, можно преобразовать в видимость посредством атрибутов. Например: при создании нового экземпляра LineRequestCourse объектом RequestCourse методу инициализации в качестве параметра передается экземпляр CourseSpecification. В конструкторе значение этого параметра присваивается атрибуту и устанавливается видимость посредством атрибутов.

Локальная видимость м*ежду объектами A и B существует в том случае, если объект B объявлен в качестве локальной переменной в методе объекта A. Существует только в контексте одного метода. Эта форма видимости достигается двумя способами***.***

1. Создается новый локальный экземпляр и присваивается локальной переменной.
2. Локальной переменной присваивается объект, возвращаемый другим методом.

Существует вариант второго способа, когда переменная явно не объявляется в методе, но неявно существует как возвращаемое значение метода.

Глобальная видимость м*ежду объектами A и B существует в том случае, если объект B является глобальным по отношению к A. Эта видимость существует до тех пор, пока существуют объекты A и B.*

1.2.20. Шаблоны проектирования

При создании классов и распределении обязанностей и способов взаимодействия объектов программисту предоставляются широкие возможности выбора. Неоптимальный выбор может сделать системы и их отдельные компоненты непригодными для поддержки, восприятия, повторного использования и расширения. Систематизация приемов программирования и принципов организации классов получила название шаблон. Основные принципы объектно-ориентированного проектирования, применяемого при создании диаграммы классов и распределения обязанностей между ними, систематизированы в шаблонах GRASP (General Responsibility Assignment Software Patterns).

При этом были сформулированы основные принципы и общие стандартные решения, придерживаясь которых можно создавать хорошо структурированный и понятный код.

1.2.21. Шаблон Expert

При проектировании классов на первом этапе необходимо определить общий принцип распределения обязанностей между классами в распределенной системе, а именно: в первую очередь определить кандидатов в информационные эксперты – классы, обладающие информацией, требуемой для выполнения своей функциональности.

Например, в подсистеме прохождения назначенного теста некоторому классу необходимо знать число вопросов, на которые получен ответ, на текущий момент времени в процессе тестирования. Какой класс должен отвечать за знание количества вопросов, на которые дан ответ на текущий момент времени при прохождении теста, если определены следующие классы?

*/*пример # 1: шаблон Expert:LineRequestQuest.java :Test.java : Quest.java */*

```
class Test {
private int idTest;
private int numberQuest;
private String testName;
private int currentNumberQuest;

 // реализация
}
class LineRequestQuest {
 private int questID;

 // реализация
}
class Quest {
 private int idQuest;
 private int testID;

 // реализация
}
```

Необходимо узнать количество вопросов из теста, на которые дан ответ, то есть число созданных объектов класса **LineRequestQuest**. Такой информацией обладает лишь экземпляр объекта **Test**, так как этот класс ответственен за знание общего количества вопросов в тесте. Следовательно, с точки зрения шаблона Expert объект **Test** подходит для выполнения этой обязанности, т. е. является информационным экспертом.

*/*пример # 2: шаблон Expert: Test.java */*

```
class Test {
private int idTest;
private int numberQuest;
private String testName;
```

```
private int currentNumberQuest;

 public int getCurrentNumberQuest() {
// реализация
}
// реализация
}
```

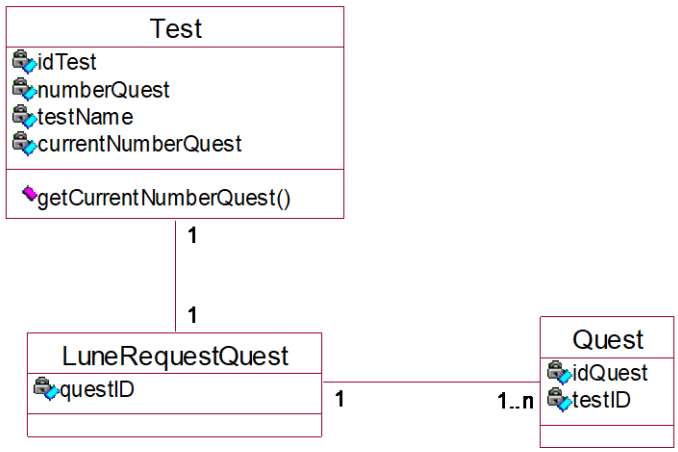


Рисунок 37 – Пример реализации шаблона Expert.

Преимущества:

назначение ответственности классам, которые уже обладают информацией для обеспечения необходимой функциональности, сохраняет инкапсуляцию этой информации;

уклонение от новых зависимостей способствует обеспечению низкой степени связанности между классами (low coupling);

если класс уже обладает информацией для обеспечения необходимой функциональности, то добавление соответствующего метода способствует высокому зацеплению (highly cohesive) классов.

Однако назначение чрезмерно большого числа ответственностей классу при использовании шаблона Expert может привезти к получению слишком сложных классов, которые перестанут удовлетворять шаблонам Low Coupling и High Cohesion.

1.2.22. Шаблон Creator

Существует большая вероятность того, что класс будет проще, если он будет большую часть своего жизненного цикла ссылаться на создаваемые объекты.

После определения информационных экспертов следует определить классы ответственные за создание нового экземпляра некоторого класса. Следует назначить классу **B** создавать экземпляры класса **A**, если выполняется одно из следующих условий:

- класс **B** агрегирует (aggregate) объекты **A**;
- класс **B** содержит (contains) объекты **A**;
- класс **B** записывает или активно использует (records or closely uses) экземпляры объектов **A**;

- классы **B** и **A** относятся к одному и тому же типу и их экземпляры составляют, агрегируют, содержат или напрямую используют другие экземпляры того же класса;
- класс **B** содержит или получает данные инициализации (has the initializing data), которые будут передаваться объектам **A** при его создании.

Если выполняется одно из указанных условий, то класс **B** – создатель (creator) объектов **A**.

Инициализация объектов – стандартный процесс. Грамотное распределение обязанностей при проектировании позволяет создать слабо связанные независимые простые классы и компоненты.

В соответствии с шаблоном необходимо найти класс, который должен отвечать за создание нового экземпляра объекта **Quest** (агрегирующий экземпляры объектов **Quest)**.

Поскольку объект **LineRequestQuest** использует объект **Quest**, согласно шаблону Creator он является кандидатом для выполнения обязанности, связанной с созданием экземпляров объектов **Quest**.

В этом случае обязанности могут быть распределены следующим образом:

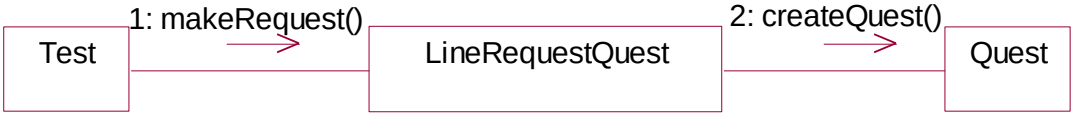


Рисунок 38 – Пример реализации шаблона Creator.

```
/* пример # 3: шаблон Creator:Qest.java:LineRequestQuest.java :Test.java */
class Test {
private int idTest;
private int numberQuest;
private String testName;
private int currentNumberQuest;
 // реализация
 public void answerQuest() {
 LineRequestQuest lr =
new LineRequestQuest();
      lr.makeRequest(параметры);
// реализация
  }
}
class LineRequestQuest {
 private int questID;

 public void makeRequest(параметры) {
      Quest q = Quest.createQuest(параметры);
// реализация
  }
}
class Quest{
 private int idQuest;
```

```
 private int testID;

 private Quest(){}
 public static Quest createQuest(параметры){
// реализация
  }
}
```

Шаблон Creator способствует низкой зависимости между классами (Low Coupling), так как экземпляры класса, которым необходимо содержать ссылку на некоторые объекты, должны создавать эти объекты. При создании некоторого объекта самостоятельно, класс тем самым перестает быть зависимым от класса, отвечающего за создание объектов для него. Распределение обязанностей выполняется в процессе создания диаграммы взаимодействия классов.

1.2.23. Шаблон Low Coupling

Степень связанности классов определяет – как сильно класс связан с другими классами, и какой информацией о других классах обладает. При проектировании отношений между классами следует распределить обязанности таким образом, чтобы степень связанности оставалась низкой.

Наличие классов с высокой степенью связанности нежелательно, так как

- изменения в связанных классах приводят к локальным изменениям в данном классе;
- затрудняется понимание каждого класса в отдельности;
- усложняется повторное использование, поскольку для этого требуется дополнительный анализ классов, с которыми связан данный класс.

Пусть требуется создать экземпляр класса **Quest** и связать его с объектом **Test**. В предметной области регистрация объекта **Test** выполняется объектом **Course**.

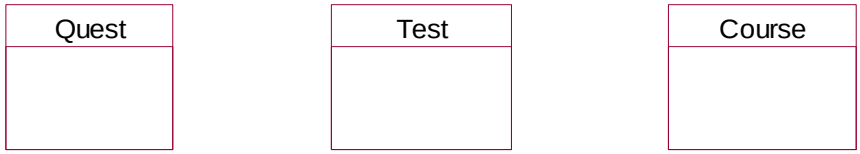


Рисунок 39 – Классы, которые необходимо связать.

Далее экземпляр объекта **Course** должен передать сообщение m**ake Quest ()** объекту **Test**. Это значит, что в текущем тесте были получены идентификаторы всех вопросов, составляющих тест.

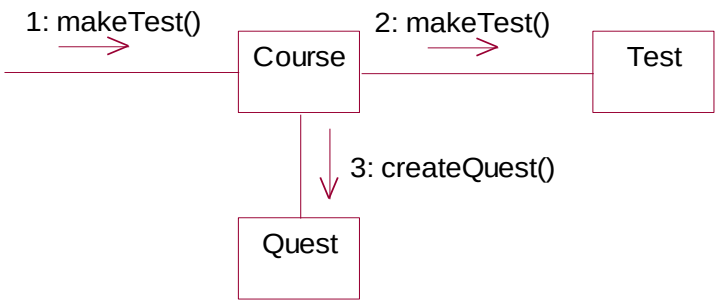


Рисунок 40 – Пример плохой реализации шаблона Low Coupling.

```
/* пример # 4: шаблон Low Coupling:Qest.java:Test.java:Course.java */

class Course {
 private int id;
 private String name;

 public void makeTest(){
      Test test = new Test();
      //....
      while(условие) {
 Quest quest = Quest.createQuest(параметры);
 //....
 test.makeTest(quest);
      }
// реализация
  }
}
class Test {
// поля и их инициализация
 public void makeTest(Quest quest){
// реализация
  }
}
class Quest {
// поля и их инициализация

 public static Quest createQuest(параметры){
      // реализация
  }
}
```

При таком распределении обязанностей предполагается, что класс **Course** связывается с данными класса **Quest**.

Второй вариант:

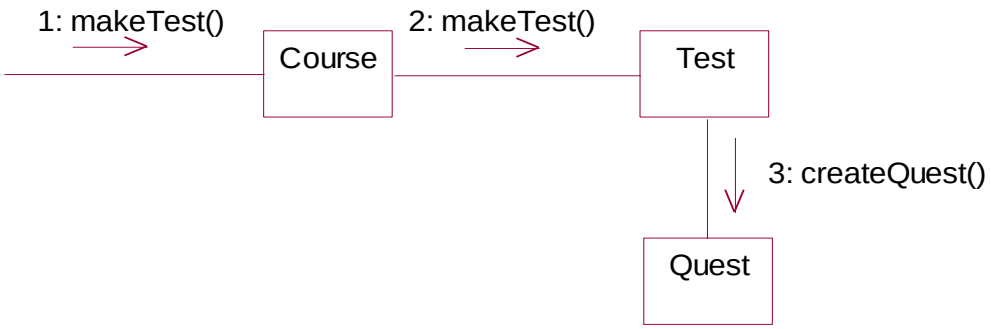


Рисунок 41 – Пример правильной реализации шаблона Low Coupling.

```
/* пример # 5: шаблон Low Coupling:Qest.java:Test.java:Course.java */
class Course{
private int id;
```

```
 private String name;

 public void makeTest(){
      Test test = new Test ();
      //....
      test.makeTest();
      // реализация
  }
}
class Test {
// поля и их инициализация

 public void makeTest(){
  while(условие) {
     Quest quest = Quest.createQuest(параметры);
// реализация
}
  }
}
class Quest {
// поля и их инициализация

 public static Quest createQuest(параметры){
      // реализация
  }
}
```

Какой из методов проектирования, основанный на распределении обязанностей, обеспечивает низкую степень связанности?

В обоих случаях предполагается, что объекту **Test** должно быть известно о существовании объекта **Quest**.

При использовании первого способа, когда объект **Quest** создается объектом **Course**, между этими двумя объектами добавляется новая связь, тогда как второй способ степень связывания объектов не увеличивает. Более предпочтителен второй способ, так как он обеспечивает низкую связываемость.

В ООП имеются некоторые стандартные способы связывания объектов **A** и **B**:

- объект **A** содержит атрибут, который ссылается на экземпляр объекта **B**;
- объект **A** содержит метод, который ссылается на экземпляр объекта **B**. Что подразумевает использование **B** в качестве типа параметра, локальной переменной или возвращаемого значения;
- объект **A** является подклассом объекта **B**;
- объект **B** является интерфейсом, а **A** реализует этот интерфейс.

Шаблон **Low Coupling** нельзя рассматривать изолированно от других шаблонов (**Expert**, **Creator**, **High Cohesion**). Не существует абсолютной меры для определения слишком высокой степени связывания.

Преимущества использования шаблона **Low Coupling**:

- изменение компонентов мало сказываются на других классах;
- принципы работы и функции компонентов можно понять, не изучая другие классы.

1.2.24. Шаблон High Cohesion

С помощью этого шаблона можно обеспечить возможность управления сложностью, распределив обязанности, поддерживая высокую степень зацепления.

Зацепление – мера связанности и сфокусированности класса. При высоком зацеплении обязанности класса тесно связаны между собой, и класс не выполняет работ непомерных размеров. Класс с низкой степенью зацепления выполняет много разнородных действий или не связанных между собой обязанностей.

Возникают проблемы, связанные с тем, что класс:

- труден в понимании так как, необходимо уделять внимание несвязным (неродственным) идеям;
- сложен в поддержке и повторном использовании из-за того, что он должен быть использован вместе с зависимыми классами;
- ненадежен, постоянно подвержен изменениям.

Классы со слабым зацеплением выполняют обязанности, которые легко можно распределить между другими объектами.

Пусть необходимо создать экземпляр класса **Quest** и связать его с заданным тестом. Какой класс должен выполнять эту обязанность? В предметной области сведения о вопросах на текущий момент времени при прохождении теста записываются в объекте **Course**, согласно шаблону для создания экземпляра объекта **Quest** можно использовать объект **Course**. Тогда экземпляр объекта **Course** сможет отправить сообщение **make Test ()** объекту **Test**. За прохождение теста отвечает объект **Course**, т. е. объект **Course** частично несет ответственность за выполнение операции **make Test ()**. Однако если и далее возлагать на класс **Course** обязанности по выполнению все новых функций, связанных с другими системными операциями, то этот класс будет слишком перегружен, и будет обладать низкой степенью зацепления.

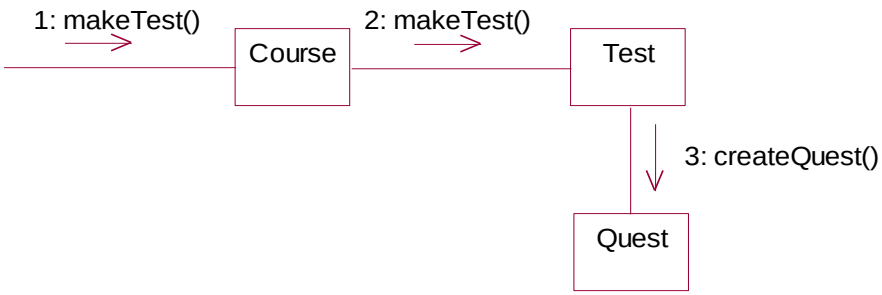


Рисунок 42 – Пример реализации шаблона High Cohesion.

Этот шаблон необходимо применять при оценке эффективности каждого проектного решения.

Виды зацепления:

Очень слабое зацепление. Единоличное выполнение множества операций.

*/*пример # 6: очень слабое зацепление: System.java */*

```
class System {
 public void createTCPServer(){
      //...
 }
 public int connectDataBase(URL url){
      //...
 }
 public void createTmpFile(){
      //...
 }
}
```

Слабое зацепление. Единоличное выполнение сложной задачи из одной функциональной области.

*/*пример # 7: слабое зацепление : NetLogic.java */*

```
class NetLogic{
 public void createTCPServer(){
      //...
 }
 public void createTCPClient(){
      //...
 }
 public void createUDPServer(){
      //...
 }
 public void createUDPClient(){
      //...
 }
}
```

Среднее зацепление. Несложные обязанности в нескольких различных областях, логически связанных с концепцией этого класса, но не связанных между собой.

*/*пример # 8: среднее зацепление : TCPServer.java */*

```
  class TCPServer{
 public void createTCPServer(){
      //...
 }
 public void receiveData(){
      //...
 }
 public void sendData(){
```

```
        //...
    }
    public void compression(){
        //...
    }
    public void decompression(){
        //...
    }
}
```

Сильное зацепление. Среднее количество обязанностей из одной функциональной области при взаимодействии с другими классами.

*/*пример # 9: сильное зацепление : TCPServer.java : OperateData.java : CodingData.java */*

```
class TCPServer {
    public void createTCPServer(){
        //...
    }
}
class OperateData {
    public void receiveData(){
        //...
    }
    public void sendData(){
        //...
    }
}
class CodingData{
    public void compression(){
        //...
    }
    public void decompression(){
        //...
    }
}
```

Если обнаруживается, что используется слишком негибкий дизайн, который сложен в поддержке, следует обратить внимание на классы, не обладающие свойством зацепления, или зависимы от других классов. Эти классы легки в узнавании, поскольку они сильно взаимосвязаны с другими классами или содержат множество неродственных методов. Как правило, классы, которые не обладают сильной зависимостью с другими классами, обладают свойством зацепления и наоборот. При наличии таких классом необходимо реорганизовать их структуру таким образом, чтобы они, по возможности, не являлись зависимыми и обладали свойством зацепления.

1.2.25. Шаблон Controller

Одной из важнейших задач при проектировании распределенных систем является определение класса, отвечающего за обработку системных событий. При необходимости посылки внешнего события прямо объекту приложения, которое обрабатывает это событие, как минимум один из объектов должен содержать ссылку на другой объект, что может послужить причиной очень негибкого дизайна, если обработчик событий зависит от типа источника событий или источник событий зависит от типа обработчика событий.

В простейшем случае зависимость между внешним источником событий и внутренним обработчиком событий заключается исключительно в передаче событий. Довольно просто обеспечить необходимую степень независимости между источником событий и обработчиком событий, используя интерфейсы. Интерфейсов может оказаться недостаточно для обеспечения поведенческой независимости между источником и обработчиком событий, когда отношения между этими источником и обработчиком достаточно сложны.

Можно избежать зависимости между внешним источником событий и внутренним обработчиком событий путем введения дополнительного объекта между ними, который будет работать в качестве посредника при передаче событий. Этот объект должен быть способен справляться с любыми другими сложными аспектами взаимоотношений между объектами.

Согласно шаблону Controller производится делегирование обязанностей по обработке системных сообщений классу, если он:

- представляет всю организацию или всю систему в целом (внешний контроллер);
- представляет активный объект из реального мира, который может участвовать в решении задачи (контроллер роли);
- представляет искусственный обработчик всех системных событий прецедента и называется `Прецедент`**`Handler`** (контроллер прецедента).

Для всех системных событий в рамках одного прецедента используется один и тот же контролер.

Controller – класс, не относящийся к интерфейсу пользователя и отвечающий за обработку системных событий. Использование объекта-контроллера обеспечивает независимость между внешними источниками событий и внутренними обработчиками событий, их типом и поведением. Выбор определяется зацеплением и связыванием.

Раздутый контроллер (волшебный сервлет) выполняет слишком много обязанностей. Признаки:

- в системе имеется единственный класс контроллера, получающий все системные сообщения, которых поступает слишком много (внешний или ролевой контроллер);
- контроллер имеет много атрибутов (информации), которую необходимо распределить между другими объектами.

1.2.26. Шаблон Facade

Проблема. Требуется общий унифицированный интерфейс с подсистемой.

Шаблон **Façade** обеспечивает интерфейс со служебными пакетами типа **Service**.

Вводится класс, обеспечивающий общий интерфейс с группой других компонентов или набором разнообразных интерфейсов (классы, наборы функций или подсистемы – локальные или удаленные).

Решение. Единый класс, обеспечивающий общий интерфейс с обязанностями взаимодействовать с системой.

Например, пакет MyDBAccess обеспечивает преобразование объектов в строки таблиц. Можно определить один класс (DBFacade), обеспечивающий общий интерфейс со всеми службами пакетами. Классы других пакетов должны отправлять сообщения только экземпляру DBFacade и не должны иметь дела с другими классами пакета. Поддерживается низкий уровень связывания между объектами.

1.2.27. Шаблон Model – View Separation

Обеспечивает отсутствие прямого взаимодействия с окнами (отделение модели от внешнего представления).

Почему или зачем это нужно?

Окна связаны с конкретным приложением, а остальные компоненты (в идеале) могут повторно использоваться в новых приложениях или дополняться другим интерфейсом. Между специальными классами и классами уровня представления не должно существовать прямой видимости.

Рекомендация шаблона – специальные классы (классы уровня реализации) должны инкапсулировать информацию и поведение, связанные с логикой приложения, а классы (уровня и представления) отвечают только за ввод и вывод информации, но не поддерживают обработку данных.

Решение. Определить классы таким образом, чтобы они не имели прямого доступа к классам окон, и чтобы данные приложения и его функции поддерживались специальными классами.

Обоснование шаблона:

1) поддерживает высокий уровень специализации объектов и позволяет разрабатывать предметную область, а не интерфейс;
2) разделяет процесс разработки уровней модели и интерфейса пользователя;
3) минимизирует влияние изменений интерфейса на специализированные объекты;
4) легко подключает новый интерфейс, не затрагивая уровень реализации;
5) одновременное использование нескольких различных представлений для одного специального объекта. Например, вывод информации в таблице или диаграмме.

1.2.28. Непрямое взаимодействие

Объекты интерфейса отправляют сообщения специальным объектам с запросами на получение информации, которую необходимо отобразить. Это модель с использованием опросов.

Однако иногда модели опросов оказывается недостаточно, так как специальные объекты должны непрямо взаимодействовать с окнами, чтобы обеспечить отображение информации в реальном времени.

Например:

- приложения мониторинга (системы управления телекоммуникационными сетями);
- моделирование с визуализацией результатов (аэродинамическое моделирование).

В такой ситуации требуется модель предоставления информации снизу (push-from-below).

При этом непрямое направление уведомления об обновлении информации с нижнего уровня реализации системы.

Чтобы обеспечить низкий уровень связывания отправителя и получателя, необходимо использовать другие механизмы, отличные от прямой передачи сообщений, т. е. организовать поддержку вещательной передачи сигнала.

1.2.29. Шаблон Publish – Subscribe

Проблема. Поведение объектов системы должно зависеть от изменения состояния (события) объекта издателя.

Однако объект – издатель не должен напрямую взаимодействовать теста.

Решение. Определить систему уведомления о событиях, чтобы объект – издатель мог уведомлять о прохождении теста.

Определяем класс EventManager, поддерживающий взаимосвязь между тестом и объектом, который проходит тест.

Тогда тест уведомляет о прохождении путем передачи сообщения SignalEvent классу EventManager. После этого объект EventManager находит всех объектов, заинтересованных в получении этого сообщения, и уведомляет их посредством параметризованного сообщения. Событие представляется в виде простой строки или экземпляра класса Event.

В простых системах обычно существует единственный экземпляр класса EventManager, к которому предоставляется глобальный доступ в соответствии с шаблоном.

EventManager – независимая от языка иллюстрация стандартного понятия. Вместо него можно использовать средства конкретного языка. Использование непрямого класса EventManager, управляющего подписчиками, снижает уровень связывания объекта – издателя и ограничивает круг его обязанностей. Однако при таком подходе могут возникать проблемы, связанные с производительностью системы, поскольку все события обрабатываются одним классом.

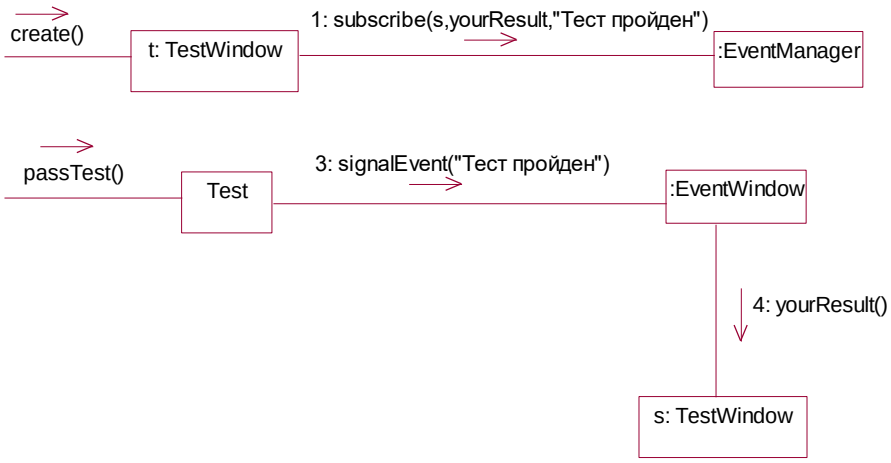


Рисунок 43 – Пример реализации шаблона Publish-Subscribe.

Предполагается передача параметризованного сообщения и списка потенциальных получателей классу EventManager. Сообщение и список получателей могут быть инкапсулированы в классе Callback. Тогда объекту EventManager передается экземпляр Callback, которому передается сообщение execute ().

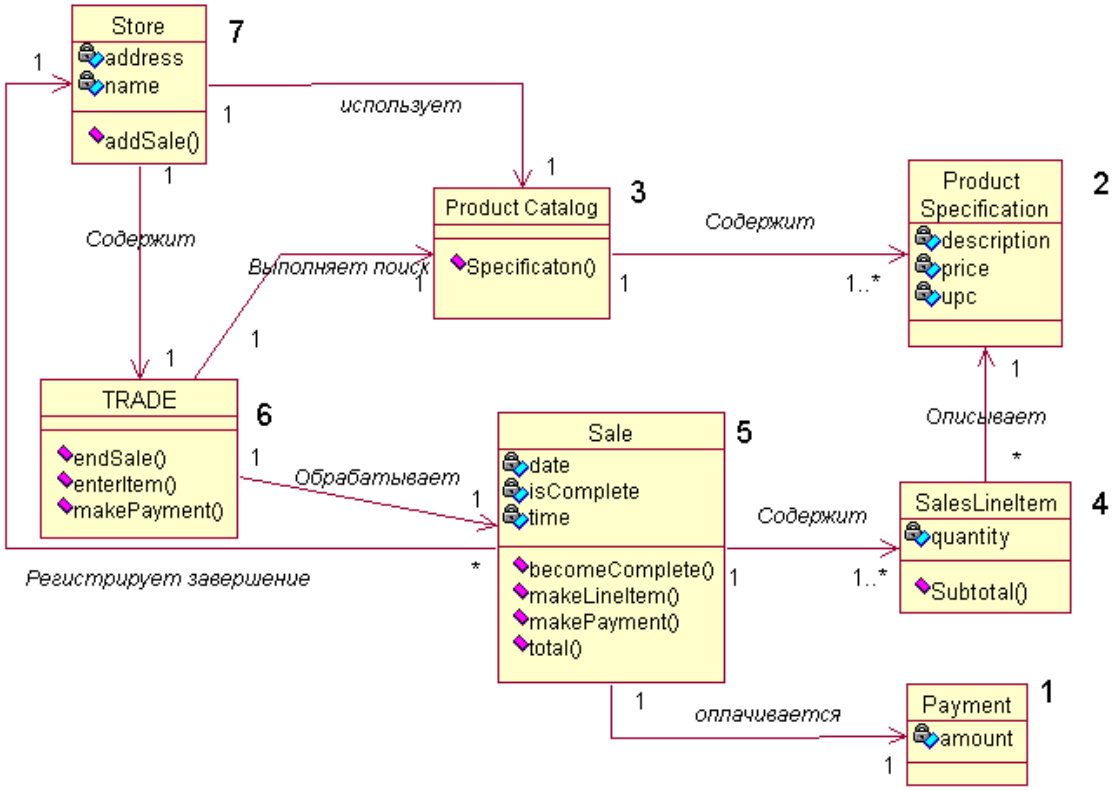


Рисунок 44 – Порядок реализации.

Классы нужно реализовывать последовательно, начиная от минимально связанных с другими классами до максимально связанных.

2. ПРАКТИЧЕСКИЙ РАЗДЕЛ

2.1. Программа лабораторных занятий

1.Общие понятия проектирования
2.Проектирование на основе анализа требований и тестирования
3.Принципы построения проекта
4.Перепроектирование или рефакторинг кода
5.Диаграммы вариантов использования
6.Диаграммы состояний и деятельности.
7.Бизнес-процесс Диаграммы взаимодействий (Sequence & Collaboration)

2.2. Проектирование информационной системы

Целями задания являются:

закрепление студентами теоретических основ проектирования, создания и использования информационных систем;

получение практических навыков создания проектов с помощью CASE средств проектирования;

знакомство с технологиями использования проектирования информационных систем;

оформление отчёта о проделанной работе.

Поставленные цели достигаются студентами путем проектирования и информационных систем, позволяющей автоматизировать отдельные внутренние и внешние бизнес-процессы некоторой организации (согласно выбранной студентами тематике).

Выполнение задания предполагает несколько последовательных шагов, соответствующих основным этапам жизненного цикла разработки информационных систем:

1) проектирование — на данном этапе определяется предметная область (часть реального мира, которая подлежит изучению) и на ее основе строится информационно-логическая модель (таблицы вариантов использования);

2) создание — на основе информационно-логической модели с помощью любого выбранного студентом Case средства проектирования;

Результаты первого, второго и второго этапов заносятся в письменный отчет к заданию, который должен содержать:

Титульный лист с указанием названия проекта, фамилии, имени и учебной группы автора, а также года выполнения.

Постановку задачи.

Инициирование проекта (фаза 0).

Определение ролей (фаза 1). Выделение и описание ролей в виде таблицы.

Построение модели уровня диаграмм (фаза 2). Определение ключевых атрибутов и доменов. Обязательно показать концептуальную схему уровня ключей.

Отчёт должен быть подготовлен в MS Word или Google Docs с использованием стилей, созданием автоматически обновляемого оглавления. В отчёте должны быть представлены таблицы и диаграммы 1) описание ролей; 2) диаграммы Use Cases; 3) диаграммы последовательностей, состояния, активности, классов. В конце отчёта привести список список использованных источников (не менее двух печатных изданий и двух электронных изданий),

К заданию, кроме отчёта прикрепить файл с диаграммами.

При защите проекта обязательным является:

знание и правильное употребление основных терминов, используемых при выполнении задания;

знание потенциальных возможностей UML;

умение объяснять взаимосвязь всех сущностей и отношений, присутствующих в информационно-логической модели, и объектов, созданных на их основе.

Примерный перечень предметных областей для проектирования информационных систем.

1. Страховая компания

Описание предметной области. Вы работаете в страховой компании. Вашей задачей является отслеживание финансовой деятельности компании. Компания имеет различные филиалы по всей стране. Каждый филиал характеризуется названием, адресом и телефоном. Деятельность компании организована следующим образом: к Вам обращаются различные лица с целью заключения договора о страховании. В зависимости от принимаемых на страхование объектов и страхуемых рисков, договор заключается по определенному виду страхования (например, страхование автотранспорта от угона, страхование домашнего имущества, добровольное медицинское страхование). При заключении договора Вы фиксируете дату заключения, страховую сумму, вид страхования, тарифную ставку и филиал, в котором заключался договор.

2. Гостиница

Описание предметной области. Вы работаете в гостинице. Вашей задачей является отслеживание финансовой стороны работы гостиницы. Ваша деятельность организована следующим образом: гостиница предоставляет номера клиентам на определенный срок. Каждый номер характеризуется вместимостью, комфортностью (люкс, полулюкс, обычный) и ценой. Вашими клиентами являются различные лица, о которых Вы собираете определенную информацию (фамилия, имя, отчество и некоторый комментарий). Сдача номера клиенту производится при наличии свободных мест в номерах, подходящих клиенту по указанным выше параметрам. При поселении фиксируется дата поселения. При выезде из гостиницы для каждого места запоминается дата освобождения.

3. Ломбард

Описание предметной области. Вы работаете в ломбарде. Вашей задачей является отслеживание финансовой стороны работы ломбарда. Деятельность

Вашей компании организована следующим образом: к Вам обращаются различные лица с целью получения денежных средств под залог определенных товаров. У каждого из приходящих к Вам клиентов Вы запрашиваете фамилию, имя, отчество и другие паспортные данные. После оценивания стоимости принесенного в качестве залога товара Вы определяете сумму, которую готовы выдать на руки клиенту, а также свои комиссионные. Кроме того, определяете срок возврата денег. Если клиент согласен, то Ваши договоренности фиксируются в виде документа, деньги выдаются клиенту, а товар остается у Вас. В случае если в указанный срок не происходит возврата денег, товар переходит в Вашу собственность.

4. Реализация готовой продукции

Описание предметной области. Вы работаете в компании, занимающейся оптово-розничной продажей различных товаров. Вашей задачей является отслеживание финансовой стороны работы компании. Деятельность Вашей компании организована следующим образом: Ваша компания торгует товарами из определенного спектра. Каждый из этих товаров характеризуется наименованием, оптовой ценой, розничной ценой и справочной информацией. В Вашу компанию обращаются покупатели. Для каждого из них Вы запоминаете в базе данных стандартные данные (наименование, адрес, телефон, контактное лицо) и составляете по каждой сделке документ, запоминая наряду с покупателем количество купленного им товара и дату покупки.

5. Ведение заказов

Описание предметной области. Вы работаете в компании, занимающейся оптовой продажей различных товаров. Вашей задачей является отслеживание финансовой стороны работы компании. Деятельность Вашей компании организована следующим образом: Ваша компания торгует товарами из определенного спектра. Каждый из этих товаров характеризуется ценой, справочной информацией и признаком наличия или отсутствия доставки. В Вашу компанию обращаются заказчики. Для каждого из них Вы запоминаете в базе данных стандартные данные (наименование, адрес, телефон, контактное лицо) и составляете по каждой сделке документ, запоминая наряду с заказчиком количество купленного им товара и дату покупки.

6. Бюро по трудоустройству

Описание предметной области. Вы работаете в бюро по трудоустройству. Вашей задачей является отслеживание финансовой стороны работы компании. Деятельность Вашего бюро организована следующим образом: Ваше бюро готово искать работников для различных работодателей и вакансии для ищущих работу специалистов различного профиля. При обращении к Вам клиента-работодателя, его стандартные данные (название, вид деятельности, адрес, телефон) фиксируются в базе данных. При обращении к Вам клиента-соискателя, его стандартные данные (фамилия, имя, отчество, квалификация, профессия, иные данные) также фиксируются в базе данных. По каждому факту удовлетворения интересов обеих сторон составляется документ. В документе указываются соискатель, работодатель, должность и комиссионные (доход бюро).

7. Нотариальная контора

Описание предметной области. Вы работаете в нотариальной конторе. Вашей задачей является отслеживание финансовой стороны работы компании. Деятельность Вашей нотариальной конторы организована следующим образом: Ваша фирма готова предоставить клиенту определенный комплекс услуг. Для наведения порядка Вы формализовали эти услуги, составив их список с описанием каждой услуги. При обращении к Вам клиента, его стандартные данные (название, вид деятельности, адрес, телефон) фиксируются в базе данных. По каждому факту оказания услуги клиенту составляется документ. В документе указываются услуга, сумма сделки, комиссионные (доход конторы), описание сделки.

8. Фирма по продаже запчастей

Описание предметной области. Вы работаете в фирме, занимающейся продажей запасных частей для автомобилей. Вашей задачей является отслеживание финансовой стороны работы компании. Основная часть деятельности, находящейся в Вашем ведении, связана с работой с поставщиками. Фирма имеет определенный набор поставщиков, по каждому из которых известны название, адрес и телефон. У этих поставщиков Вы приобретаете детали. Каждая деталь наряду с названием характеризуется артикулом и ценой (считаем цену постоянной). Некоторые из поставщиков могут поставлять одинаковые детали (один и тот же артикул). Каждый факт покупки запчастей у поставщика фиксируется в базе данных, причем обязательными для запоминания являются дата покупки и количество приобретенных деталей.

9. Курсы по повышению квалификации

Описание предметной области. Вы работаете в учебном заведении и занимаетесь организацией курсов повышения квалификации. В Вашем распоряжении имеются сведения о сформированных группах студентов. Группы формируются в зависимости от специальности и отделения. В каждой из них включено определенное количество студентов. Проведение занятий обеспечивает штат преподавателей. Для каждого из них у Вас в базе данных зарегистрированы стандартные анкетные данные (фамилия, имя, отчество, телефон) и стаж работы. В результате распределения нагрузки Вы получаете информацию о том, сколько часов занятий проводит каждый преподаватель с соответствующими группами. Кроме того, хранятся также сведения о виде проводимых занятий (лекции, практика), предмете и оплате за 1 час.

10. Определение факультативов для студентов

Описание предметной области. Вы работаете в высшем учебном заведении и занимаетесь организацией факультативов. В Вашем распоряжении имеются сведения о студентах, включающие стандартные анкетные данные (фамилия, имя, отчество, адрес, телефон). Преподаватели Вашей кафедры должны обеспечить проведение факультативных занятий по некоторым предметам. По каждому факультативу существует определенное количество часов и вид проводимых занятий (лекции, практика, лабораторные работы). В результате работы со студентами у Вас появляется информация о том, кто из них записался на какие факультативы. Существует некоторый минимальный объем

факультативных предметов, которые должен прослушать каждый студент. По окончанию семестра Вы заносите информацию об оценках, полученных студентами на экзаменах.

11. Распределение учебной нагрузки

Описание предметной области. Вы работаете в высшем учебном заведении и занимаетесь распределением нагрузки между преподавателями кафедры. В Вашем распоряжении имеются сведения о преподавателях кафедры, включающие наряду с анкетными данными сведения об их ученой степени, занимаемой административной должности и стаже работы. Преподаватели Вашей кафедры должны обеспечить проведение занятий по некоторым предметам. По каждому из них существует определенное количество часов. В результате распределения нагрузки у Вас должна получится информация следующего рода: «Такой-то преподаватель проводит занятия по такому-то предмету с такой-то группой».

12. Распределение дополнительных обязанностей

Описание предметной области. Вы работаете в коммерческой компании и занимаетесь распределением дополнительных разовых работ. Вашей задачей является отслеживание хода выполнения дополнительных работ. Компания имеет определенный штат сотрудников, каждый из которых получает определенный оклад. Время от времени, возникает потребность в выполнении некоторой дополнительной работы, не входящей в круг основных должностных обязанностей сотрудников. Для наведения порядка в этой сфере деятельности Вы проклассифицировали все виды дополнительных работ, определившись с суммой оплаты по факту их выполнения. При возникновении дополнительной работы определенного вида Вы назначаете ответственного, фиксируя дату начала. По факту окончания Вы фиксируете дату и выплачиваете дополнительную сумму к зарплате с учетом Вашей классификации.

13. Техническое обслуживание станков

Описание предметной области. Ваше предприятие занимается ремонтом станков и другого промышленного оборудования. Вашей задачей является отслеживание финансовой стороны деятельности предприятия. Клиентами Вашей компании являются промышленные предприятия, оснащенные различным сложным оборудованием. В случае поломок оборудования они обращаются к Вам. Ремонтные работы в Вашей компании организованы следующим образом: все станки проклассифицированы по странам-производителям, годам выпуска и маркам. Все виды ремонта отличаются названием, продолжительностью в днях, стоимостью. Исходя из этих данных, по каждому факту ремонта Вы фиксируете вид станка и дату начала ремонта.

14. Туристическая фирма

Описание предметной области. Вы работаете в туристической компании. Ваша компания работает с клиентами, продавая им путевки. Вашей задачей является отслеживание финансовой стороны деятельности фирмы. Работа с клиентами в Вашей компании организована следующим образом: у каждого клиента, пришедшего к Вам, собираются некоторые стандартные данные – фамилия, имя, отчество, адрес, телефон. После этого Ваши сотрудники выясняют

у клиента, куда он хотел бы поехать отдыхать. При этом ему демонстрируются различные варианты, включающие страну проживания, особенности местного климата, имеющиеся отели разного класса. Наряду с этим, обсуждается возможная длительность пребывания и стоимость путевки. В случае если удалось договориться, и найти для клиента приемлемый вариант, Вы регистрируете факт продажи путевки (или путевок, если клиент покупает сразу несколько путевок), фиксируя дату отправления. Иногда Вы решаете предоставить клиенту некоторую скидку.

15. Грузовые перевозки

Описание предметной области. Вы работаете в компании, занимающейся перевозками грузов. Вашей задачей является отслеживание стоимости перевозок с учетом заработной платы водителей. Ваша компания осуществляет перевозки по различным маршрутам. Для каждого маршрута Вы определили некоторое название, вычислили примерное расстояние и установили некоторую оплату для водителя. Информация о водителях включает фамилию, имя, отчество и стаж. Для проведения расчетов Вы храните полную информацию о перевозках (маршрут, водитель, даты отправки и прибытия). По факту некоторых перевозок водителям выплачивается премия.

16. Учет телефонных переговоров

Описание предметной области. Вы работаете в коммерческой службе телефонной компании. Компания предоставляет абонентам телефонные линии для междугородних переговоров. Вашей задачей является отслеживание стоимости междугородних телефонных переговоров. Абонентами компании являются юридические лица, имеющие телефонную точку, ИНН, расчетный счет в банке. Стоимость переговоров зависит от города, в который осуществляется звонок, и времени суток (день, ночь). Каждый звонок абонента автоматически фиксируется в базе данных. При этом запоминаются город, дата, длительность разговора и время суток.

17. Учет внутриофисных расходов

Описание предметной области. Вы работаете в бухгалтерии частной фирмы. Сотрудники фирмы имеют возможность осуществлять мелкие покупки для нужд фирмы, предоставляя в бухгалтерию товарный чек. Вашей задачей является отслеживание внутриофисных расходов. Ваша фирма состоит из отделов. Каждый отдел имеет название. В каждом отделе работает определенное количество сотрудников. Сотрудники могут осуществлять покупки в соответствии с видами расходов. Каждый вид расходов имеет название, некоторое описание и предельную сумму средств, которые могут быть потрачены по данному виду расходов в месяц. При каждой покупке сотрудник оформляет документ, где указывает вид расхода, дату, сумму и отдел.

18. Библиотека

Описание предметной области. Вы являетесь руководителем библиотеки. Ваша библиотека решила зарабатывать деньги, выдавая напрокат некоторые книги, имеющиеся в небольшом количестве экземпляров. Вашей задачей является отслеживание финансовых показателей работы библиотеки. У каждой книги, выдаваемой в прокат, есть название, автор, жанр. В зависимости от

ценности книги Вы определили для каждой из них залоговую стоимость (сумма, вносимая клиентом при взятии книги напрокат) и стоимость проката (сумма, которую клиент платит при возврате книги, получая назад залог). В библиотеку обращаются читатели. Все читатели регистрируются в картотеке, которая содержит стандартные анкетные данные (фамилия, имя, отчество, адрес, телефон). Каждый читатель может обращаться в библиотеку несколько раз. Все обращения читателей фиксируются, при этом по каждому факту выдачи книги запоминаются дата выдачи и ожидаемая дата возврата.

19. Прокат автомобилей

Описание предметной области. Вы являетесь руководителем коммерческой службы в фирме, занимающейся прокатом автомобилей. Вашей задачей является отслеживание финансовых показателей работы пункта проката. В Ваш автопарк входит некоторое количество автомобилей различных марок, стоимостей и типов. Каждый автомобиль имеет свою стоимость проката. В пункт проката обращаются клиенты. Все клиенты проходят обязательную регистрацию, при которой о них собирается стандартная информация (фамилия, имя, отчество, адрес, телефон). Каждый клиент может обращаться в пункт проката несколько раз. Все обращения клиентов фиксируются, при этом по каждой сделке запоминаются дата выдачи и ожидаемая дата возврата.

20. Выдача банком кредитов

Описание предметной области. Вы являетесь руководителем информационно-аналитического центра коммерческого банка. Одним из существенных видов деятельности Вашего банка является выдача кредитов юридическим лицам. Вашей задачей является отслеживание динамики работы кредитного отдела. В зависимости от условий получения кредита, процентной ставки и срока возврата все кредитные операции делятся на несколько основных видов. Каждый из этих видов имеет свое название. Кредит может получить юридическое лицо (клиент), при регистрации предоставивший следующие сведения: название, вид собственности, адрес, телефон, контактное лицо. Каждый факт выдачи кредита регистрируется банком, при этом фиксируются сумма кредита, клиент и дата выдачи.

21. Инвестирование свободных средств

Описание предметной области. Вы являетесь руководителем аналитического центра инвестиционной компании. Ваша компания занимается вложением денежных средств в ценные бумаги. Ваши клиенты – предприятия, которые доверяют Вам управлять их свободными денежными средства на определенный период. Вам необходимо выбрать вид ценных бумаг, которые позволят получить прибыль и Вам и Вашему клиенту. При работе с клиентом для Вас весьма существенной является информация о предприятии – название, вид собственности, адрес и телефон.

22. Занятость актеров театра

Описание предметной области. Вы являетесь коммерческим директором театра, и в Ваши обязанности входит вся организационно-финансовая работа, связанная с привлечением актеров и заключением контрактов. Вы поставили дело следующим образом: каждый год театр осуществляет постановку

различных спектаклей. Каждый спектакль имеет определенный бюджет. Для участия в конкретных постановках в определенных ролях Вы привлекаете актеров. С каждым из актеров Вы заключаете персональный контракт на определенную сумму. Каждый из актеров имеет некоторый стаж работы, некоторые из них удостоены различных наград и званий.

23. Платная поликлиника

Описание предметной области. Вы являетесь руководителем службы планирования платной поликлиники. Вашей задачей является отслеживание финансовых показателей работы поликлиники. В поликлинике работают врачи различных специальностей, имеющие разную квалификацию. Каждый день в поликлинику обращаются больные. Все больные проходят обязательную регистрацию, при которой в базу данных заносятся стандартные анкетные данные (фамилия, имя, отчество, год рождения). Каждый больной может обращаться в поликлинику несколько раз, нуждаясь в различной медицинской помощи. Все обращения больных фиксируются, при этом устанавливается диагноз, определяется стоимость лечения, запоминается дата обращения.

24. Анализ динамики показателей финансовой отчетности различных предприятий

Описание предметной области. Вы являетесь руководителем информационно-аналитического центра крупного холдинга. Вашей задачей является отслеживание динамики показателей для предприятий Вашего холдинга. В структуру холдинга входят несколько предприятий. Каждое предприятие имеет стандартные характеристики (название, реквизиты, телефон, контактное лицо). Работа предприятия может быть оценена следующим образом: в начале каждого отчетного периода на основе финансовой отчетности вычисляется по неким формулам определенный набор показателей. Принять, что важность показателей характеризуется некоторыми числовыми константами. Значение каждого показателя измеряется в некоторой системе единиц.

25. Учет телекомпанией стоимости прошедшей в эфире рекламы

Описание предметной области. Вы являетесь руководителем коммерческой службы телевизионной компании. Вашей задачей является отслеживание расчетов, связанных с прохождением рекламы в телеэфире. Работа построена следующим образом: заказчики просят поместить свою рекламу в определенной передаче в определенный день. Каждый рекламный ролик имеет определенную продолжительность. Для каждой организации-заказчика известны банковские реквизиты, телефон и контактное лицо для проведения переговоров. Передачи имеют определенный рейтинг. Стоимость минуты рекламы в каждой конкретной передаче известна (определяется коммерческой службой, исходя из рейтинга передачи и прочих соображений).

26. Интернет-магазин

Описание предметной области. Вы являетесь сотрудником коммерческого отдела компании, продающей различные товары через Интернет. Вашей задачей является отслеживание финансовой составляющей работы компании. Работа Вашей компании организована следующим образом: на Интернет-сайте компании представлены (выставлены на продажу) некоторые товары. Каждый из

них имеет некоторое название, цену и единицу измерения (штуки, килограммы, литры). Для проведения исследований и оптимизации работы магазина Вы пытаетесь собирать данные с Ваших клиентов. При этом для Вас определяющее значение имеют стандартные анкетные данные, а также телефон и адрес электронной почты для связи. В случае приобретения товаров на сумму свыше 5000 р. клиент переходит в категорию «постоянных клиентов» и получает скидку на каждую покупку в размере 2%. По каждому факту продажи Вы автоматически фиксируете клиента, товары, количество, дату продажи, дату доставки.

27. Ювелирная мастерская

Описание предметной области. Вы работаете в ювелирной мастерской. Ваша мастерская осуществляет изготовление ювелирных изделий для частных лиц на заказ. Вы работаете с определенными материалами (платина, золото, серебро, различные драгоценные камни и т. д.). При обращении к Вам потенциального клиента Вы определяетесь с тем, какое именно изделие ему необходимо. Все изготавливаемые Вами изделия принадлежат к некоторому типу (серьги, кольца, броши, браслеты), бывают выполнены из определенного материала, имеют некоторый вес и цену (включающую стоимость материалов и работы).

28. Парикмахерская

Описание предметной области. Вы работаете в парикмахерской. Ваша парикмахерская стрижет клиентов в соответствии с их пожеланиями и некоторым каталогом различных видов стрижки. Так, для каждой стрижки определены название, принадлежность полу (мужская, женская), стоимость работы. Для наведения порядка Вы, по мере возможности, составляете базу данных клиентов, запоминая их анкетные данные (фамилия, имя, отчество). Начиная с 5-й стрижки, клиент переходит в категорию постоянных и получает скидку в 3% при каждой последующей стрижке. После того, как закончена очередная работа, в кассе фиксируются стрижка, клиент и дата производства работ.

29. Химчистка

Описание предметной области. Вы работаете в химчистке. Ваша химчистка осуществляет прием у населения вещей для выведения пятен. Для наведения порядка Вы, по мере возможности, составляете базу данных клиентов, запоминая их анкетные данные (фамилия, имя, отчество). Начиная с 3-го обращения, клиент переходит в категорию постоянных клиентов и получает скидку в 3% при чистке каждой последующей вещи. Все оказываемые Вами услуги подразделяются на виды, имеющие название, тип и стоимость, зависящую от сложности работ. Работа с клиентом первоначально состоит в определении объема работ, вида услуги и, соответственно, ее стоимости. Если клиент согласен, он оставляет вещь (при этом фиксируется услуга, клиент и дата приема) и забирает ее после обработки (при этом фиксируется дата возврата).

30. Сдача в аренду торговых площадей

Описание предметной области. Вы работаете в крупном торговом центре, сдающим в аренду коммерсантам свои торговые площади. Вашей задачей является наведение порядка в финансовой стороне работы торгового центра.

Работы Вашего торгового центра построена следующим образом: в результате планирования Вы определили некоторое количество торговых точек в пределах Вашего здания, которые могут сдаваться в аренду. Для каждой из торговых точек важными данными являются этаж, площадь, наличие кондиционера и стоимость аренды в день. Со всех потенциальных клиентов Вы собираете стандартные данные (название, адрес, телефон, реквизиты, контактное лицо). При появлении потенциального клиента Вы показываете ему имеющиеся свободные площади. При достижении соглашения Вы оформляете договор, фиксируя в базе данных торговую точку, клиента, период (срок) аренды.

3. РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

3.1. Перечень рекомендуемых средств диагностики

Диагностика результатов учебной деятельности по дисциплине «Анализ и проектирование информационных систем» проводится, как правило, во время аудиторных занятий. Для диагностики используются:

– устный опрос;

– отчет по лабораторным и домашним заданиям.

Контроль УСР проводится преподавателем с использованием ИКТ в форме опроса и проверки результатов выполнения работы.

Формой промежуточной аттестации по дисциплине «Анализ и проектирование информационных систем» учебным планом предусмотрен **зачет.**

3.2. Примерный перечень заданий для управляемой самостоятельной работы студентов

Установка и конфигурирование Enterprise Architect. (2ч)

Задание 1. Загрузка и установка Enterprise Architect

Форма контроля – опрос, отчет по лабораторным / домашним заданиям.

3.3. Описание инновационных подходов и методов к преподаванию учебной дисциплины

При организации образовательного процесса используются

- ***методы и приемы развития критического мышления,*** которые представляют собой систему, формирующую навыки работы с информацией в процессе чтения и письма; понимании информации как отправного, а не конечного пункта критического мышления;
- ***практико-ориентированный подход,*** который предполагает освоение содержание образования через решения практических задач.

3.4. Методические рекомендации по организации самостоятельной работы обучающихся

При изучении учебной дисциплины рекомендуется использовать следующие формы самостоятельной работы:

- изучение литературы и материалов электронных источников по проблемам дисциплины;
- выполнение домашнего задания;
- подготовка к лабораторным занятиям;
- курсовые, дипломные и научно-исследовательские работы, связанные с тематикой дисциплины;

- подготовка к участию в конференциях с докладами по проблемам дисциплины.

Для организации самостоятельной работы студентов по учебной дисциплине рекомендуется использовать современные информационные ресурсы, размещенные на образовательном портале смешанного и дистанционного обучения БГУ, содержащие учебные материалы (курс лекций, задания к лабораторным и домашним работам, перечень вопросов к зачёту и т. д.)

3.5. Примерный перечень вопросов к зачету

1. Что такое UML? Какие основные виды диаграмм предоставляет UML?
2. Какие основные элементы классов в UML? Как они связаны между собой?
3. Что такое ассоциация, агрегация и композиция в UML? В чем разница между ними?
4. Что такое диаграмма случаев использования (Use Case Diagram) в UML? Какие элементы она содержит и как они взаимодействуют?
5. Что такое диаграмма классов (Class Diagram) в UML? Какие элементы она содержит и как они связаны друг с другом?
6. Какие основные элементы диаграммы последовательности (Sequence Diagram) в UML? Как они отображают взаимодействие между объектами?
7. Что такое диаграмма состояний (State Diagram) в UML? Какие элементы она содержит и как они отображают переходы состояний объекта?
8. Что такое Rational Unified Process (RUP)? Какие фазы и основные документы входят в RUP?
9. Какие основные роли определены в RUP? Какие обязанности каждой роли в процессе разработки?
10. Как происходит итеративное и инкрементное развитие в RUP? Какие преимущества это дает при разработке информационных систем?
11. Что такое диаграмма компонентов (Component Diagram) в UML? Какие элементы она содержит и как они связаны друг с другом?
12. Что такое диаграмма развертывания (Deployment Diagram) в UML? Какие элементы она содержит и как они связаны с аппаратным и программным обеспечением?
13. Что такое диаграмма пакетов (Package Diagram) в UML? Каким образом она организует модули и компоненты системы?
14. Какие основные этапы жизненного цикла разработки программного обеспечения определены в RUP? Что происходит на каждом этапе?
15. Что такое прецедент (Use Case) в контексте диаграммы случаев использования? Как они связаны с актерами и как они помогают в анализе системы?
16. Какие основные виды связей (Relationships) между классами в диаграмме классов? Как они используются для описания отношений между классами?

17. Что такое диаграмма активности (Activity Diagram) в UML? Какие элементы она содержит и как они отображают поток управления и действий в системе?

18. Что такое диаграмма коммуникации (Communication Diagram) в UML? Как она отображает взаимодействие между объектами и передачу сообщений?

19. Какие основные элементы диаграммы композитного структуры (Composite Structure Diagram) в UML? Как они отображают вложенные структуры и связи между элементами?

20. Что такое артефакт (Artifact) в контексте диаграммы развертывания? Какие артефакты могут быть связаны с различными компонентами системы?

21. Какие основные принципы объектно-ориентированного программирования (ООП) применяются при разработке информационных систем с использованием UML?

22. Что такое диаграмма временной последовательности (Timing Diagram) в UML? Какие элементы она содержит и как они отображают взаимодействие между объектами во времени?

23. Каким образом диаграмма развертывания помогает в планировании и развертывании системы на физическом оборудовании?

24. Что такое диаграмма составного состояния (Composite State Diagram) в UML? Как она помогает моделировать сложные состояния объектов?

25. Каким образом диаграмма деятельности (Activity Diagram) помогает в моделировании бизнес-процессов и логики работы системы?

26. Что такое рефакторинг (Refactoring) в разработке программного обеспечения? Как он связан с процессом анализа и проектирования?

27. Какие основные принципы проектирования (Design Principles) применяются при разработке информационных систем? Как они связаны с архитектурой и модульностью системы?

28. Что такое диаграмма таймлайна (Timeline Diagram) в UML? Как она помогает визуализировать ?

29. Что такое диаграмма таймлайна (Timeline Diagram) в UML? Как она помогает визуализировать события и их последовательность в системе?

30. Какие основные этапы моделирования системы с использованием UML и RUP? Как они связаны с процессом разработки?

31. Какие основные преимущества использования языка UML и методологии RUP при анализе и проектировании информационных систем?

4. ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

4.1. Список рекомендуемой литературы

4.2. Основная

1. Маран, М. М. Программная инженерия: учебное пособие для вузов / М. М. Маран. — 3-е изд., стер. — Санкт-Петербург: Лань, 2022. — 196 с. — Текст : электронный // Лань : электронно-библиотечная система. — [Электронный ресурс]. – Режим доступа: https://e.lanbook.com/book/189470. – Дата доступа : 29.01.2024.
2. Блинов, И. Н. Рациональный унифицированный процесс и язык UML: пособие по курсу "Анализ и проектирование распределенных систем" для студ. фак. прикладной математики и информатики / И. Н. Блинов. – Минск: БГУ, 2008. – 92 с. – [Электронный ресурс]. – Режим доступа: http://elib.bsu.by/handle/123456789/1847. – Дата доступа : 29.01.2024.
3. Мартин Роберт. Чистый Agile. Основы гибкости. — (Серия «Библиотека программиста»). — Санкт-Петербург: Питер, 2021. — 352 с. — [Электронный ресурс]. – Режим доступа: https://ibooks.ru/bookshelf/371720. – Дата доступа : 29.01.2024.
4. Мартин, Р. Идеальная работа. Программирование без прикрас\ Р. Мартин. — Санкт-Петербург: Питер, 2023. – 384 с.

4.3. Дополнительная

1. Г.Буч, Д. Рамбо, А. Джекобсон. Язык UML. – М: “ДМК”, 2000. – 430стр.
2. У.Богге, М. Богге. UML и Rational Rose – М: “Лори”, 2000. – 376 стр.
3. Ф.Крачтен. Введение в Rational Unifird Process – М.: “Вильямс”, 2002. – 240 стр.
4. Х.Ахмед, К.Амриш. Разработка корпоративных Java приложений с помощью J2EE™ и UML – М.: “Вильямс”, 2002. – 268 стр.
5. К. Ларманн. Применение UML и шаблонов проектирования. – М.: “Вильямс”, 2000. – 490 стр.
6. Дж. Коналлен. Разработка Web приложений с использованием UML М.: “Вильямс”, 2001. – 286 стр.
7. Т.Кватрани. Rational Rose2000 и UML – М: “ДМК”, 2001. – 176 с.
8. М.Фаулер, К.Скотт.UML основы Second Edition – СПб.: Симбо, 2002. – 186 стр.
9. Дж. Рамбо, М. Блаха. UML 2.0. Объектно-ориентированное моделирование и разработка. 2-е изд. – СПб.: Питер, 2007. – 544 стр.: ил.

10. С. Макконнелл. Совершенный код. Мастер-класс / Пер. с англ. – М.: Издательско-торговый дом “Русская редакция”; СПб.: Питер, 2005. – 896 стр.: ил.
11. С.Макконнелл. Профессиональная разработка программного. Изд.: Символ., 2007. – 240 стр.
12. С.Макконнелл. Остаться в живых! Руководство для менеджера программных проектов. Изд. : Питер. Серия: Библиотека программиста, 2006. – 240 стр.

4.4. Электронные ресурсы

1. Электронная библиотека БГУ [Электронный ресурс]. – Режим доступа: https://library.bsu.by/MegaPro/Web/SearchResult/ToPage/1. – Дата доступа : 29.01.2024.
2. Электронная библиотека БГУ [Электронный ресурс]. – Режим доступа: https://library.bsu.by/MegaPro/Web/SearchResult/ToPage/1. – Дата доступа : 29.01.2024.
3. Электронная библиотека БГУ [Электронный ресурс]. – Режим доступа: https://library.bsu.by/MegaPro/Web/SearchResult/ToPage/1. – Дата доступа : 29.01.2024.

4.5. Учебно-методическая карта учебной дисциплины

Дневная форма получения образования с применением электронных средств обучения (ДО)

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов					
		Лекции	Практические занятия	Лабораторные занятия	Иное	Количество часов УСР	Форма контроля знаний
	Общие понятия проектирования	4		4			опрос
	Проектирование на основе анализа требований и тестирования	4		4			опрос

	Принципы построения проекта	4		4		2	Опрос, отчет по лабораторным / домашним работам
	Перепроектирование или рефакторинг кода	4		2			Опрос, отчет по лабораторным / домашним работам
	Варианты использования	6		4			опрос
	Диаграммы вариантов использования	4		4		2	Опрос, отчет по лабораторным / домашним работам
	Диаграммы состояний и деятельности. Бизнес-процесс	4		4			опрос
	Диаграммы взаимодействий (Sequence & Collaboration)	4		4			Опрос, отчет по лабораторным / домашним работам
	ВСЕГО ЧАСОВ	**34**		**30**		**4**	**Зачет**

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА УЧЕБНОЙ ДИСЦИПЛИНЫ

Заочная форма получения образования

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов					
		Лекции	Практические	Лабораторные занятия	Иное	Количество часов в УСР	Форма контроля знаний
1	Общие понятия проектирования	1					опрос
2	Проектирование на основе анализа требований и тестирования	1		2			опрос
3	Принципы построения проекта	1		1			опрос
4	Перепроектирование или рефакторинг кода	1		1			Опрос, отчет по лабораторным / домашним работам
5	Варианты использования	1		1			опрос

6	Диаграммы вариантов использования	1		1			опрос
7	Диаграммы состояний и деятельности. Бизнес-процесс	1		1			опрос
8	Диаграммы взаимодействий (Sequence & Collaboration)	1		1			Опрос, отчет по лабораторным / домашним работам
	ВСЕГО ЧАСОВ	**8**		**8**			**Зачет**