Технологии программирования. Курс на базе *Microsoft Solutions Framework*

Лекции 3-4. Визуальное моделирование при анализе и проектировании. Основы Unified Modeling Language (UML)

Содержание

- Вспоминая предыдущую лекцию
- Анализ и проектирование. Некоторые частные вопросы
- Визуальное моделирование. История языка UML
- Структура языка UML
- Учебный пример. Постановка задачи
- Визуальное описание функциональной модели средствами UML
- Структура системы и ее описание средствами UML
- Что дальше?
- Литература

Содержание

- **Вспоминая предыдущую лекцию**
- Анализ и проектирование. Некоторые частные вопросы
- Визуальное моделирование. История языка UML
- Структура языка UML
- Учебный пример. Постановка задачи
- Визуальное описание функциональной модели средствами UML
- Структура системы и ее описание средствами UML
- Что дальше?
- Литература

Вспоминая предыдущую лекцию

- **Программная инженерия, основные понятия**
  - Инженеры и программные инженеры
  - Программная инженерия как инженерная дисциплина
  - Область действия программной инженерии
  - Цели программных инженеров
  - Программные инженеры и научная среда
- **Процесс создания ПО**
  - Понятие процесса. Основные фазы.
  - Модель процесса. Каскадная и эволюционная модель.
  - Итерационный подход. Модель пошаговой разработки и спиральная модель.

Содержание

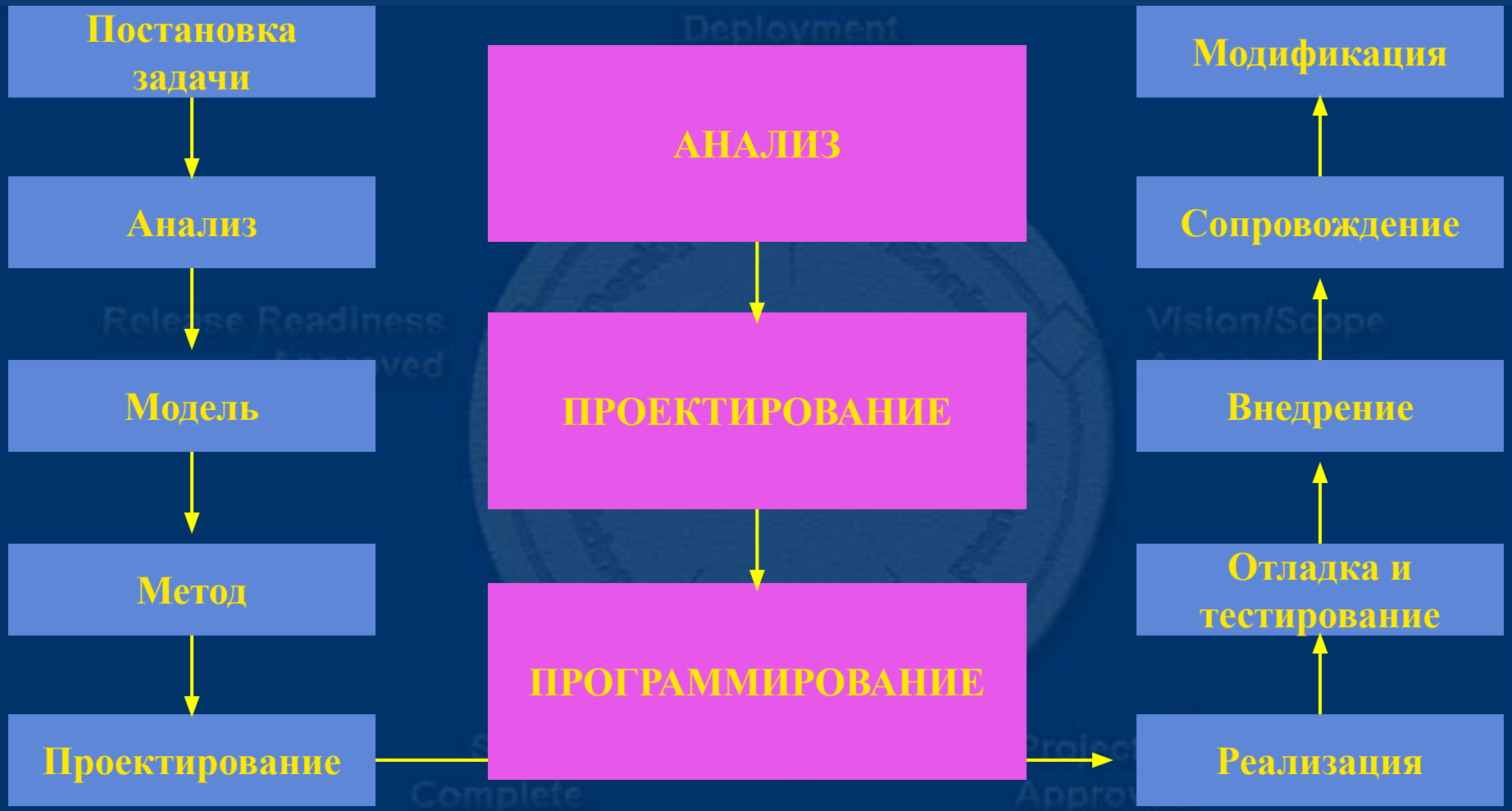
- Вспоминая предыдущую лекцию
- **Анализ и проектирование. Некоторые частные вопросы**
- Визуальное моделирование. История языка UML
- Структура языка UML
- Учебный пример. Постановка задачи
- Визуальное описание функциональной модели средствами UML
- Структура системы и ее описание средствами UML
- Что дальше?
- Литература

Решение задач с использованием ВТ

Как решают задачу с использованием вычислительной техники?

Версии зала...

Типовая схема решения задач с использованием ВТ



Анализ – Проектирование - Программирование

- 3-я часть и элементы 2-ой части этой цепочки изучаются в курсе «Методы программирования».
- 1-я и 2-я части составляют объект изучения отдельного курса «**Анализ и проектирование**».
- В настоящий момент в анализе и проектировании преобладает **объектный подход** (изучен в 1-2 семестрах).
- Вспомним суть объектного подхода.

Анализ предметной области. Декомпозиция

2 вида разбиения предметной области на составляющие:

- **Алгоритмическая декомпозиция**

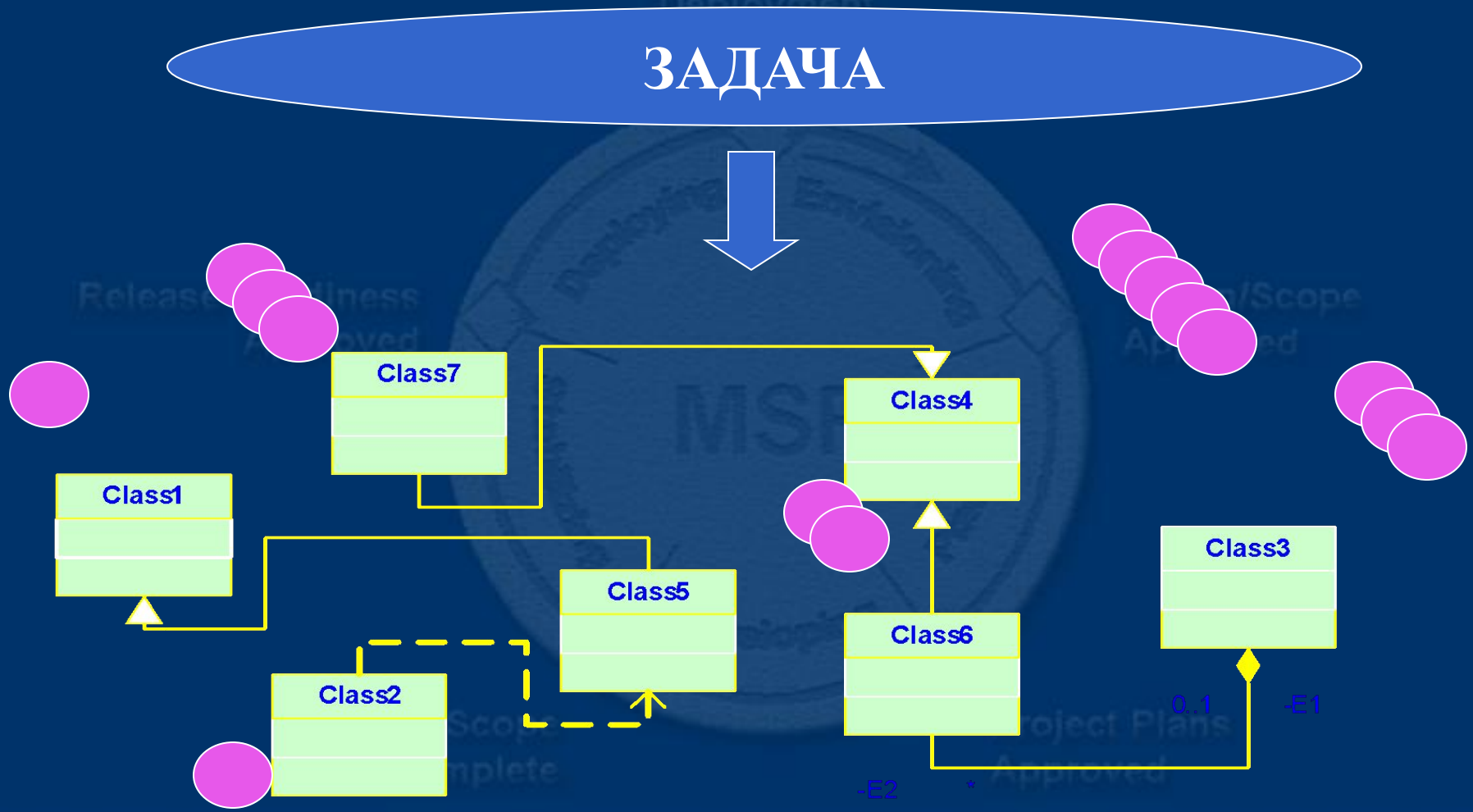
  Основные элементы – **алгоритмы**.

- **Объектная декомпозиция**

  Основные элементы – виды абстракций (**классы**) и представители этих классов (**объекты**)

Алгоритмическая декомпозиция

Объектная декомпозиция



Объектный подход

- OOA (object oriented analysis)
  объектно-ориентированный анализ
- OOD (object oriented design)
  объектно-ориентированное проектирование
- OOP (object oriented programming)
  объектно-ориентированное программирование

Принципы объектного подхода

- **Абстрагирование**.

  выделяем главное, выявляем виды абстракций
- **Инкапсуляция**.

  скрываем детали реализации
- **Иерархия**.

  иерархия помогает разбить задачу на уровни и постепенно ее решать
- **Агрегация и наследование**.

  абстракции можно создавать на основе имеющихся
- **Полиморфизм**.

  полиморфизм позволяет иметь естественные имена и выполнять действия, релевантные ситуации, разбираясь на этапе работы программы

Пример: ООП и структуры хранения. Стек. Постановка задачи

Задача.

Выполнить проектирование и реализацию структуры хранения **стека**.

Примечание.

- Не учитывать необходимость перераспределения памяти.
- Считать, что элементы целого типа.

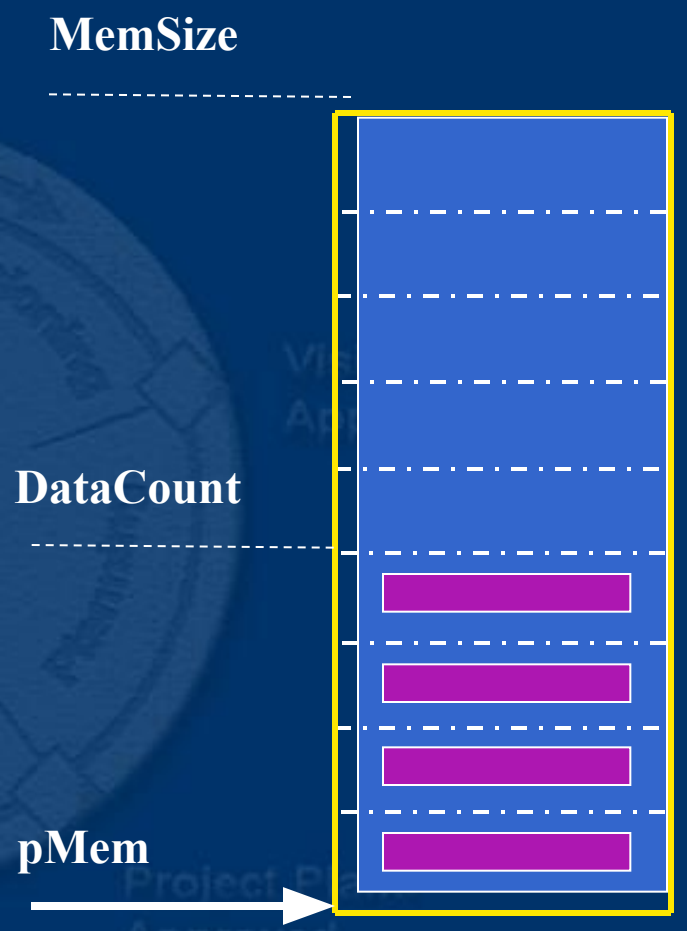
Пример: ООП и структуры хранения. Стек. Анализ и проектирование

Данные:

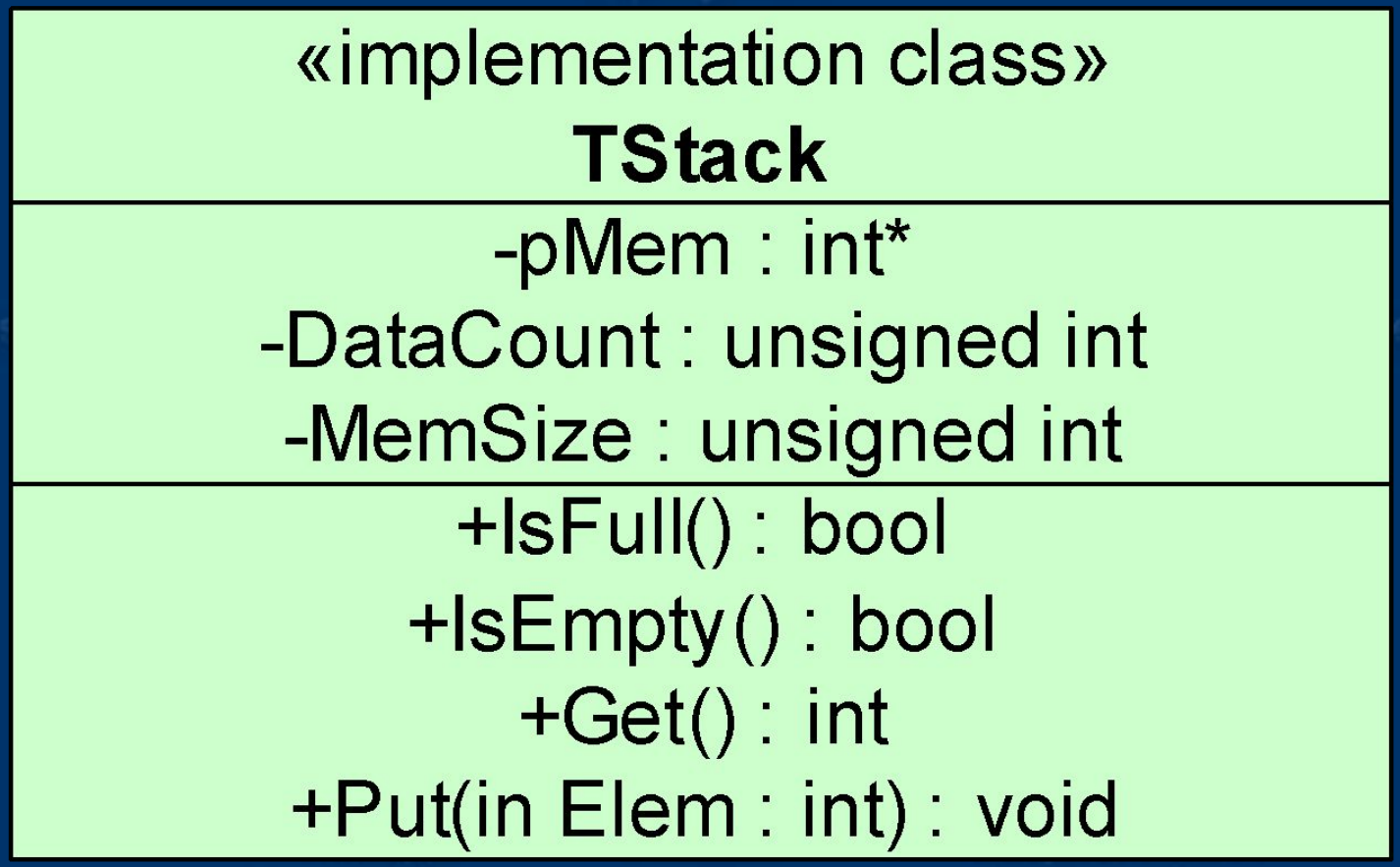
- **MemSize** – максимальное количество элементов.
- **DataCount** – количество элементов в стеке.
- **pMem** – указатель на память для хранения значений.

Операции:

- **IsFull** – проверка на полноту.
- **IsEmpty** – проверка на пустоту.
- **Get** – взять элемент с вершины.
- **Put** – положить элемент в стек.



Пример: ООП и структуры хранения. Стек. Анализ и проектирование



Повторное использование...

- **Повторное использование** – применение уже существующих наработок в разрабатываемом ПО.
- Повторное использование – важный элемент проектирования.
  - Необходимо проектировать новые элементы системы с тем, чтобы их в последствии можно было использовать.
  - Необходимо при проектировании системы рассматривать возможность использования того, что уже есть и работает.

Повторное использование. Достоинства

- **Девиз:** не надо изобретать велосипед, если он уже изобретен.
- Достоинства повторного использования (по Соммервилю):
  - Повышение надежности.
  - Уменьшение проектных рисков.
  - Эффективное использование специалистов.
  - Соблюдение стандартов (пример: UI).
  - Ускорение разработки.

Повторное использование. Виды

Повторное использование достигается за счет следующих приемов:

- **Компонентная разработка**.

  Часть компонентов уже разработаны ранее, имеют четко описанный интерфейс. Они используются в качестве «кирпичиков» в новой системе.

- **Использование паттернов (шаблонов) проектирования.**

  Применяются известные подходы к решению некоторых встречавшихся ранее проблем.

- **Использование стандартных прикладных (MKL, MFC…) и системных (API) библиотек.**

Содержание

- Вспоминая предыдущую лекцию
- Анализ и проектирование. Некоторые частные вопросы
- **Визуальное моделирование. История языка UML**
- Структура языка UML
- Учебный пример. Постановка задачи
- Визуальное описание функциональной модели средствами UML
- Структура системы и ее описание средствами UML
- Что дальше?
- Литература

Вместо введения

При изучении материалов по визуальному моделированию и языку UML в качестве основного источника рекомендуется классическая книга

Г. Буч, Дж. Рамбо, А. Джекобсон. UML. Руководство пользователя. – ДМК-Пресс, Питер, 2004.

Модель

- Проблема в разработке ПО:

  Проекты не укладываются в сроки, бюджет, не удовлетворяют требованиям.

- Как бороться? На помощь приходит **моделирование**.
- **Модель** – упрощенное представление объектов и явлений реального мира.

Смысл моделирования

- **Модель** строят для того, чтобы лучше понять исследуемую систему.
- **Задачи моделирования** [3]:
  - Визуализация системы в ее некотором состоянии.
  - Определение структуры и поведения системы.
  - Получение шаблона для создания системы.
  - Документирование принятых решений.

Принципы моделирования [3]

- Выбор модели оказывает определяющее влияние на подход к решению проблемы и на то, как будет выглядеть это решение.
- Каждая модель может быть воплощена с разной степенью абстракции.
- Лучшие модели – те, что ближе к реальности.
- Наилучший подход при разработке сложной системы – использовать несколько почти независимых моделей.

Моделирование и объектный подход

- **Объектный подход** – один из ключевых подходов к моделированию.
  В результате OOA & OOD мы получаем «хороший» проект программной системы, прозрачный, удовлетворяющий требованиям, удобный для тестирования и отладки, коллективной разработки, развиваемый, допускающий повторное использование компонентов.
- Вопрос: **все так безоблачно?**

Большие системы

- Вопрос: **все так безоблачно?**
- Ответ: **нет. В больших системах проект слишком велик для восприятия одним человеком.**

Идея визуального моделирования

- Путь к решению проблемы:
  ВИЗУАЛЬНОЕ МОДЕЛИРОВАНИЕ
- В чем смысл?
  - **Визуализация** упрощает понимание проекта в целом.
  - **Визуализация** помогает согласовать терминологию и убедиться, что все одинаково понимают термины.
  - **Визуализация** делает обсуждение конструктивным и понятным.

UML как воплощение идеи визуального моделирования

- Для визуального моделирования нужна специальная нотация или **язык**.
- **UML** (unified modeling language) – это **язык** для
  - визуализации,
  - специфицирования,
  - конструирования,
  - документирования

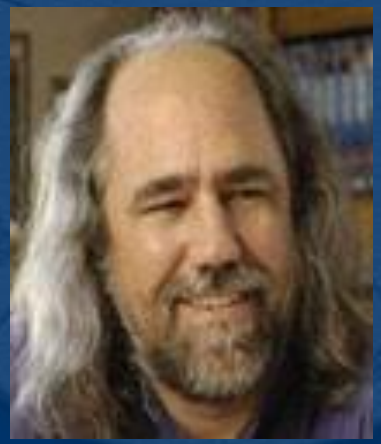
  элементов программных систем [3].
- **UML** – язык общего назначения, предназначенный для объектного моделирования.

История UML.
Этапы большого пути...*

- **1994**: Grady Booch & James Rumbaugh (Rational Software) объединили методы
  Booch (проектирование) и **OMT** (анализ) ->**Unified method**
- **1995:** присоединился Ivar Jacobson (**OOSE** метод)

"Three amigos"

James Rumbaugh



Grady Booch

Ivar Jacobson

•*Источник*: www.wikipedia.org; http://www-306.ibm.com/software/rational/bios; http://www.ivarjacobson.com

История UML. Этапы большого пути...*

- **1996 –** Идея о **Unified Modeling Language** (three amigos)
- **1996** – **UML Partners** консорциум под руководством three amigos
- Июнь, Октябрь 1996 – UML 0.9 & UML 0.91
- **Январь 1997** – спецификации **UML 1.0** предложены OMG (Object Management Group)
- **Август 1997** – спецификации **UML 1.1** предложены OMG
- **Ноябрь 1997** – **UML 1.2** результат адаптации OMG
- **Июнь 1999** – UML 1.3
- **Сентябрь 2001** – UML 1.4
- **Март 2003** – UML 1.5

•*Источник*: www.wikipedia.org; http://www-306.ibm.com/software/rational/bios; http://www.ivarjacobson.com

История UML.
Этапы большого пути*

Принятый стандарт:

- **ISO/IEC 19501:2005** Information technology – Open Distributed Processing – Unified Modeling Language (**UML**) **Version 1.4.2**

- **Октябрь 2004 – UML 2.0.**

Взято с сайта www.uml.org

•*Источник*: www.wikipedia.org; http://www-306.ibm.com/software/rational/bios; http://www.ivarjacobson.com

Содержание

- Вспоминая предыдущую лекцию
- Анализ и проектирование. Некоторые частные вопросы
- Визуальное моделирование. История языка UML
- **Структура языка UML**
- Учебный пример. Постановка задачи
- Визуальное описание функциональной модели средствами UML
- Структура системы и ее описание средствами UML
- Что дальше?
- Литература

Модели UML

UML позволяет описывать систему следующими **моделями**:

- **Модель функционирования**

  Как описывается функциональность системы с точки зрения пользователя.

- **Объектная модель**

  Как выглядит проект системы с точки зрения объектного подхода.

- **Динамическая модель**

  Как взаимодействуют друг с другом компоненты системы в динамике, с течением времени. Какие процессы происходят в системе.

Диаграммы UML

- **Диаграммы** UML предназначены для визуального отображения **моделей** и их компонентов.
- **UML 2.0** – **13** типов диаграмм.
  - Структурные диаграммы (6)
  - Диаграммы поведения (3)
  - Диаграммы взаимодействия (4)

Структурные диаграммы

- **Диаграмма классов**
  Показывает классы, их атрибуты и связи между классами.
- **Диаграмма компонентов**
  Показывает компоненты и связи между ними
- **Структурная диаграмма**
  Показывает внутреннюю структуру классов и связи с внешним миром
- **Диаграмма развертывания**
  Показывает, как ПО размещается на аппаратуре (серверах, рабочих станциях...)
- **Диаграмма объектов**
  Показывает структуру системы в конкретный момент времени, объекты, их атрибуты...
- **Диаграмма пакетов**
  Показывает, как система раскладывается на крупные составные части и связи между этими частями

Диаграммы поведения

- **Диаграмма действия**

  Показывает потоки информации в системе.

- **Диаграмма состояния**

  Представляет собой конечный автомат, показывающий функционирование системы.

- **Диаграмма вариантов использования**

  Показывает работу системы с точки зрения пользователей.

Диаграммы взаимодействия

- **Диаграмма кооперации**

  Показывает структурную организацию участвующих во взаимодействии объектов
- **Диаграмма взаимодействия**

  (новация UML 2.0)
- **Диаграмма последовательности**

  Показывает временную упорядоченность событий
- **Временная диаграмма**

  Диаграмма связана с временными рамками

Понятия UML

- **Для описания структуры:**
  Актер, Атрибут, Класс, Компонент, Интерфейс, Объект, Пакет.
- **Для описания поведения:**
  Действие, Событие, Сообщение, Метод, Операция, Состояние, Вариант использования.
- **Для описания связей:**
  Агрегация, Ассоциация, Композиция, Зависимость, Наследование.
- **Некоторые другие понятия:**
  Стереотип, Кратность, Роль.

Содержание

- Вспоминая предыдущую лекцию
- Анализ и проектирование. Некоторые частные вопросы
- Визуальное моделирование. История языка UML
- Структура языка UML
- **Учебный пример. Постановка задачи**
- Визуальное описание функциональной модели средствами UML
- Структура системы и ее описание средствами UML
- Что дальше?
- Литература

Система бронирования билетов для авиакомпании

- **SRS** – Seat reservation system.
- Авиакомпания «GlobalAvia».
- SRS должна содержать 2 части:
  - Занесение информации.
  - Работа с клиентами.
- Дополнительная информация:
  - Рейсы спланированы так, что до пункта назначения можно долететь с пересадками.
  - Система должна помогать покупать билеты в зависимости от пожеланий пользователя.

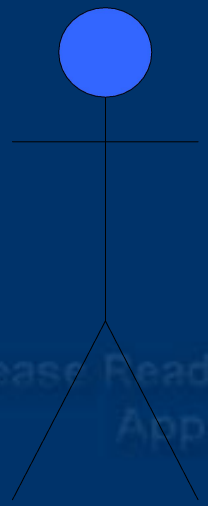
Содержание

- Вспоминая предыдущую лекцию
- Анализ и проектирование. Некоторые частные вопросы
- Визуальное моделирование. История языка UML
- Структура языка UML
- Учебный пример. Постановка задачи
- **Визуальное описание функциональной модели средствами UML**
- Структура системы и ее описание средствами UML
- Что дальше?
- Литература

Как функционирует программная система?

- Программная система **не функционирует сама по себе**.
- Программная система функционирует под воздействием **актеров** – **пользователей**, **машин** и **других программ**.
- **Актер** ожидает, что система ведет себя строго определенным образом.
- **Актер** оказывает **воздействие** – система выдает ожидаемый **результат**.
- Модель того, как воздействие приводит к результату – **Вариант использования**.

Актеры и Варианты использования в UML



Актер в UML – человек, машина или программа, воздействует на систему, является внешним по отношению к ней.

Вариант использования в UML – описание последовательности действий – (часто с вариантами – сценариями).

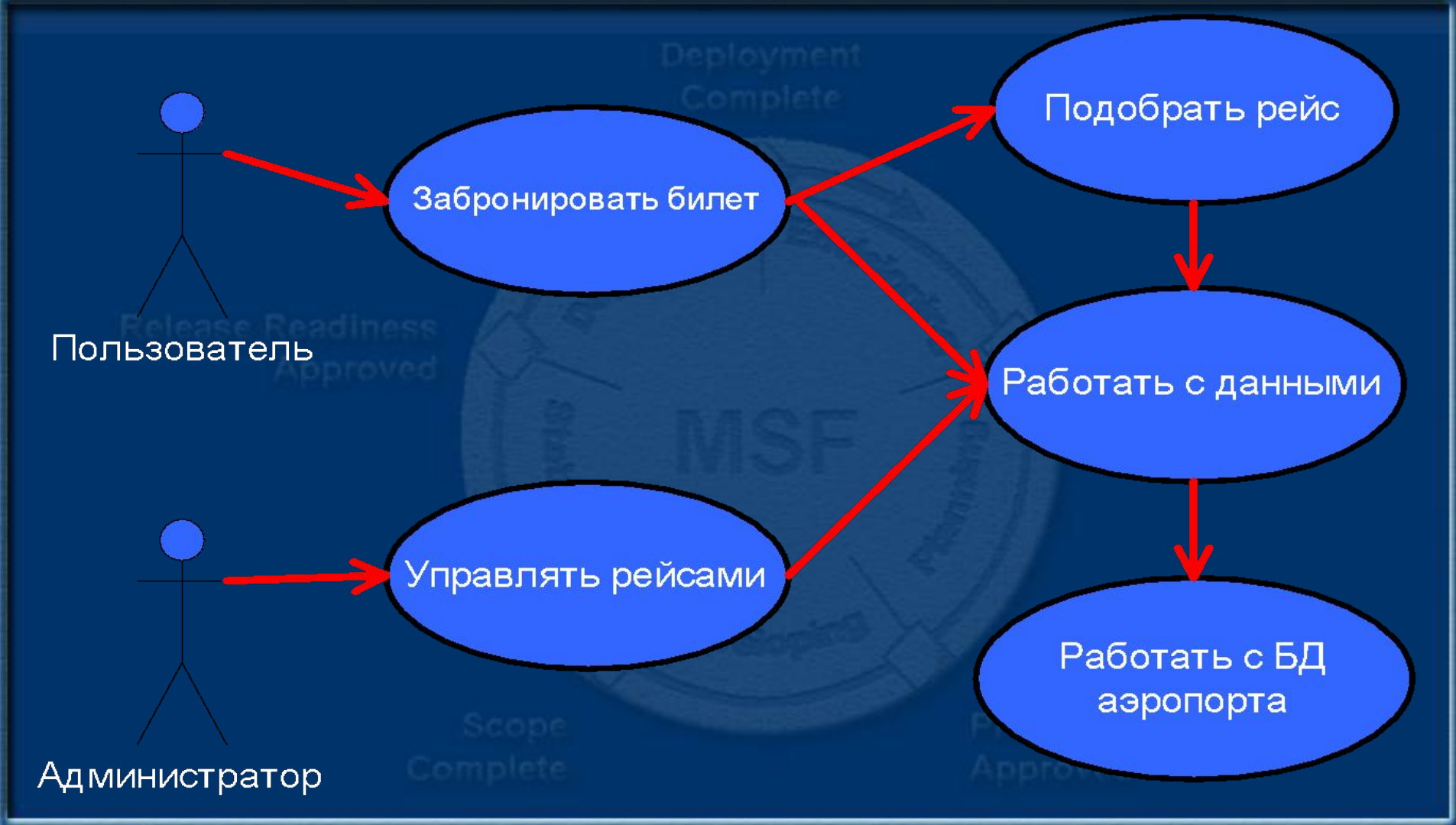
Связь актеров и вариантов использования

- Актеры и варианты использования общаются посредством посылки сообщений.
- Сообщения могут идти в обе стороны.
- Стрелка показывает инициатора общения (актер на рисунке) и может быть опущена.

Актеры и Варианты использования в SRS

- **Актеры**:
  - Пользователь.
  - Администратор.
- **Варианты использования**:
  - Забронировать билет.
  - Подобрать рейс.
  - Работать с данными.
  - Управлять рейсами.
  - Работать с БД аэропорта.

SRS – Диаграмма вариантов использования



Некоторые соображения... [3]

- При таком моделировании обращают внимание на **поведение** системы, а не на ее реализацию.
- Хорошая модель описывает **основное поведение** системы, не являясь слишком подробным.
- Подобная модель позволяет проверить, удовлетворит ли система требования заказчика.

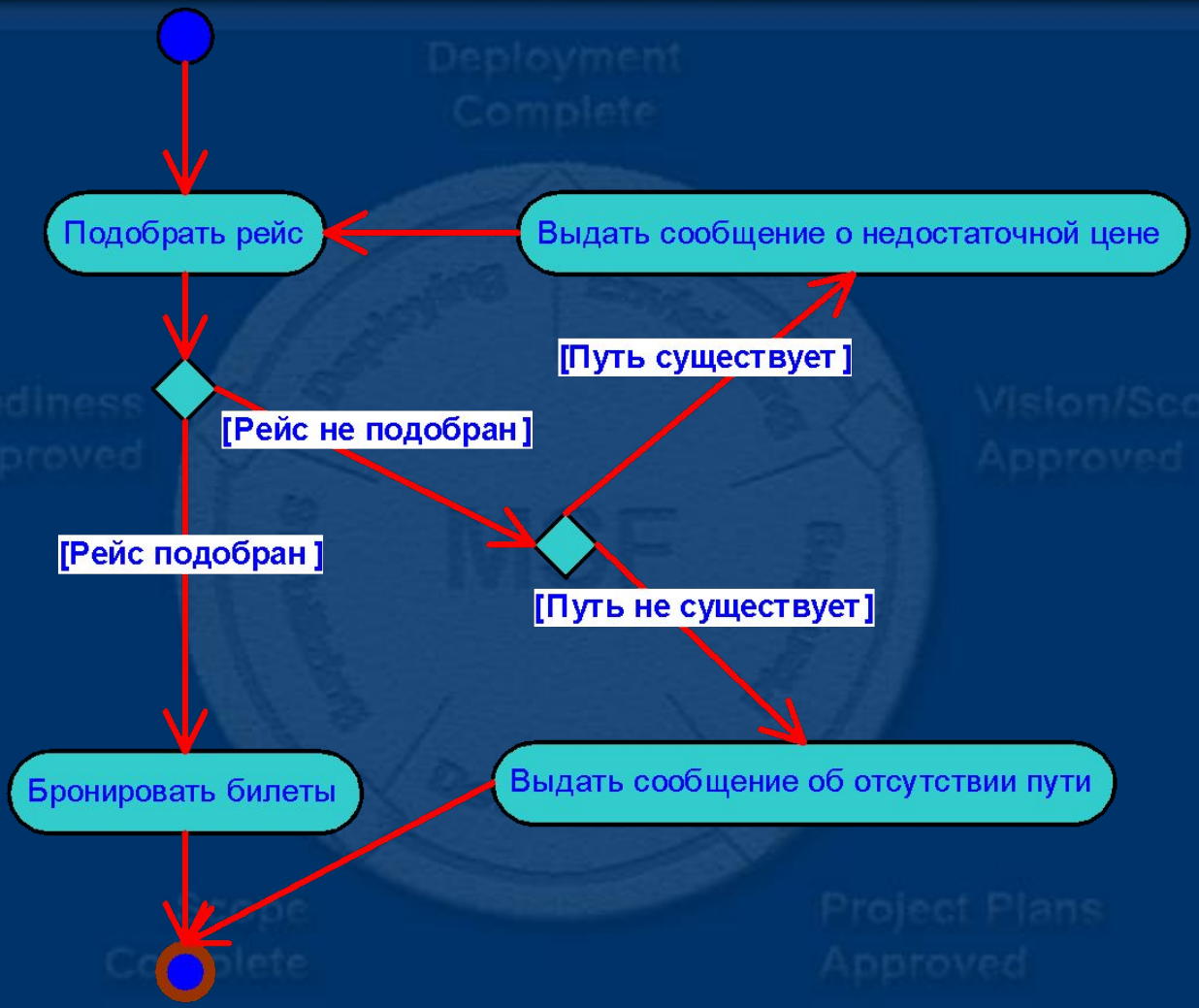
Некоторые соображения [3]

- Система средних размеров может быть описана большим количеством вариантов использования.
- Варианты использования могут описываться разными **сценариями**.

Сценарии варианта использования

- Для описания **сценариев Варианта использования** используется **Диаграмма действия**.
- Диаграмма действия это блок-схема, которая отображает динамику в поведении системы.
- Может использоваться **не только** для описания сценариев Варианта использования.

Диаграммы действия в SRS

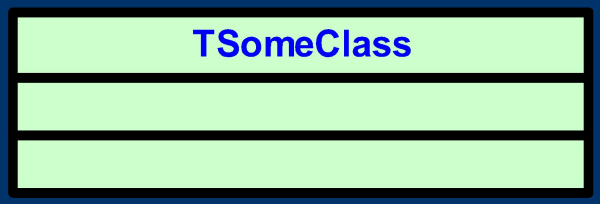


Содержание

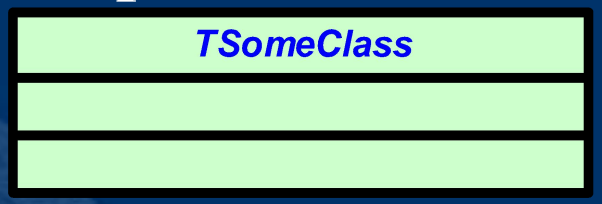
- Вспоминая предыдущую лекцию
- Анализ и проектирование. Некоторые частные вопросы
- Визуальное моделирование. История языка UML
- Структура языка UML
- Учебный пример. Постановка задачи
- Визуальное описание функциональной модели средствами UML
- **Структура системы и ее описание средствами UML**
- Что дальше?
- Литература

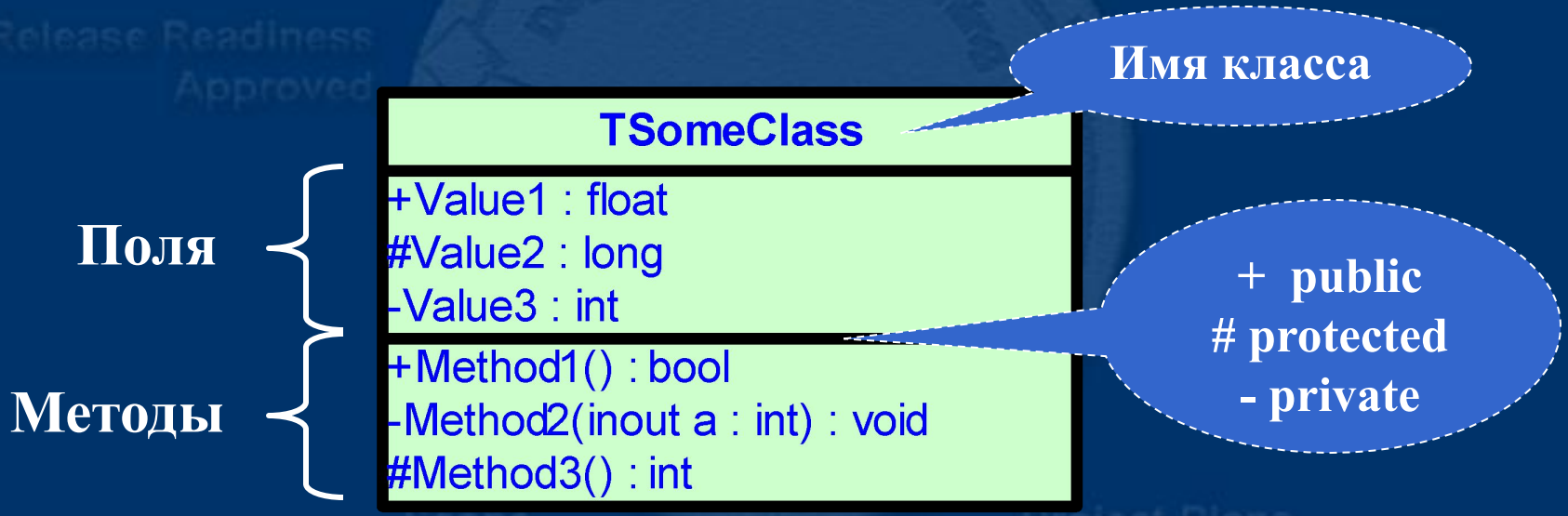
Классы в UML

Класс

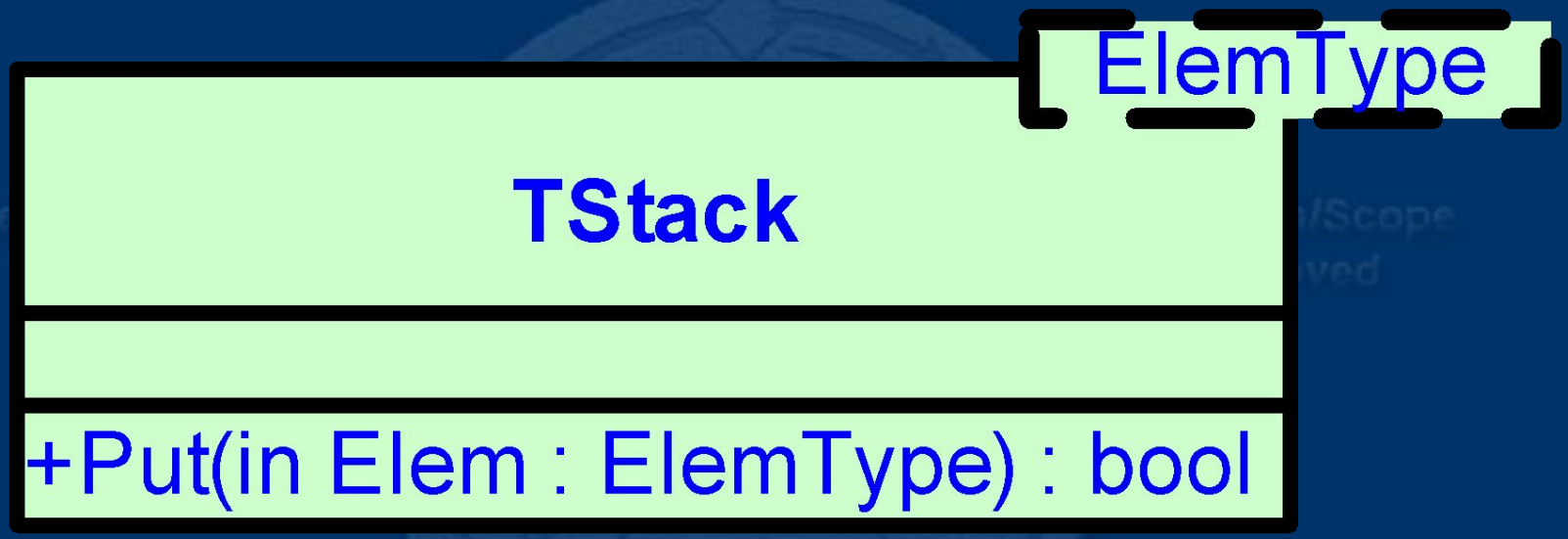


Абстрактный класс

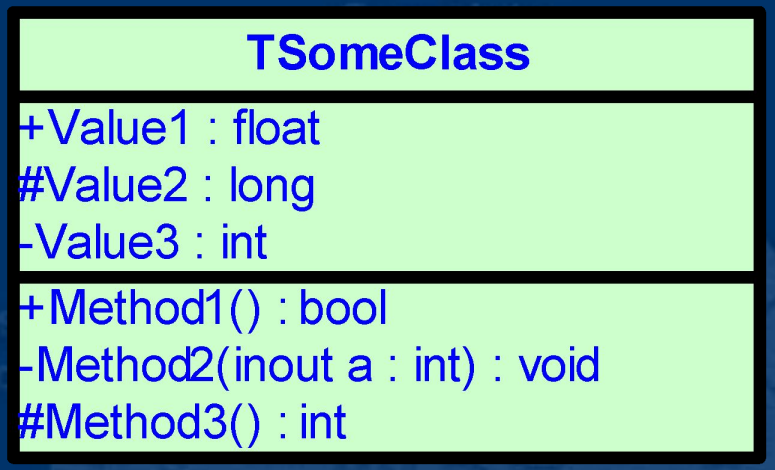




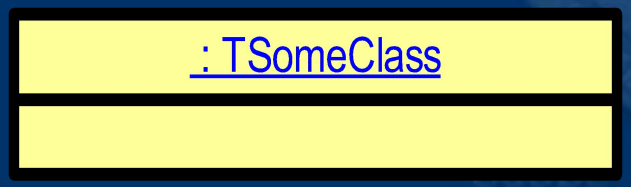
Шаблоны классов в UML



Объекты в UML



Объект

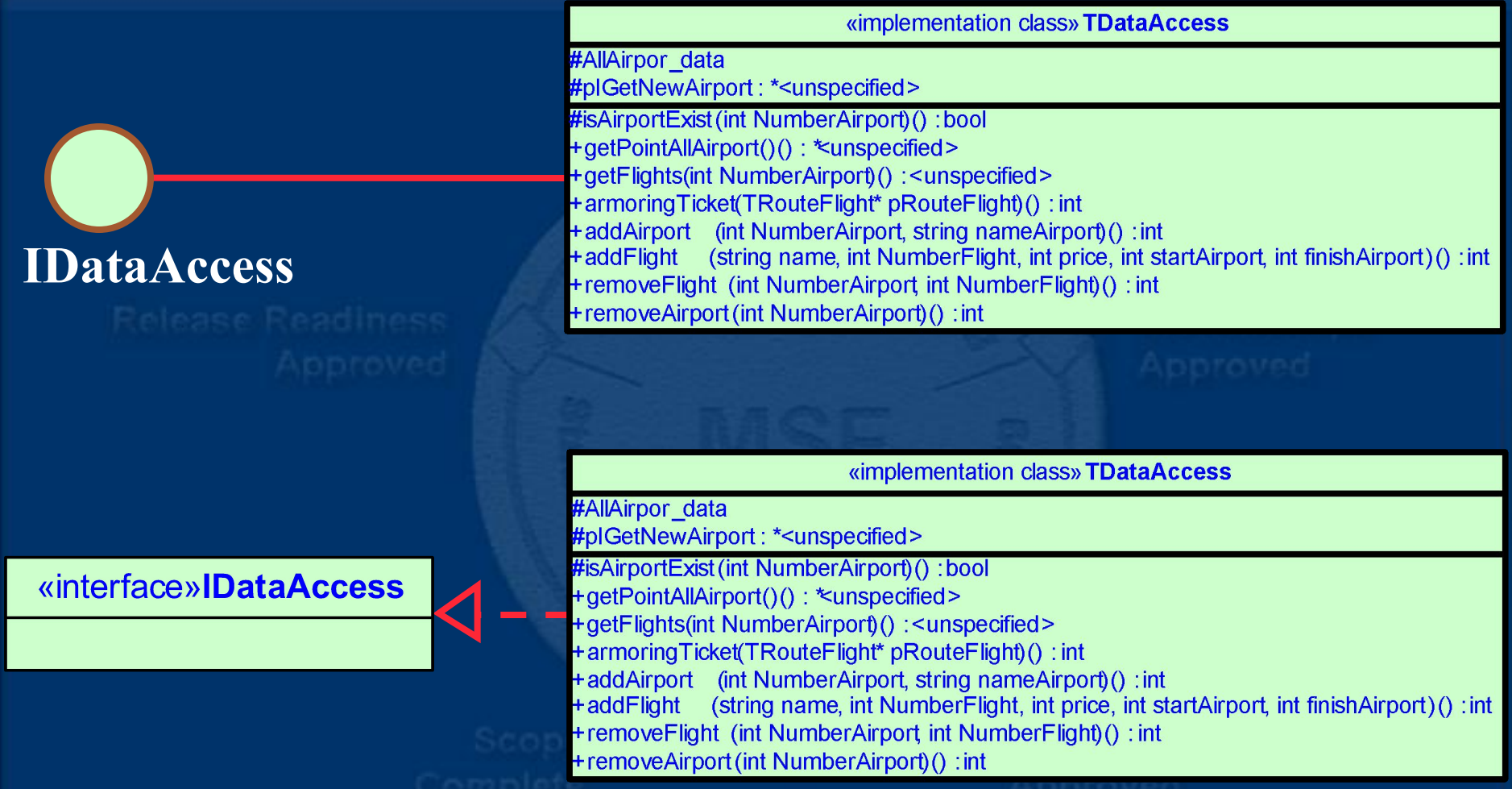


Именованный объект

Интерфейсы [3]

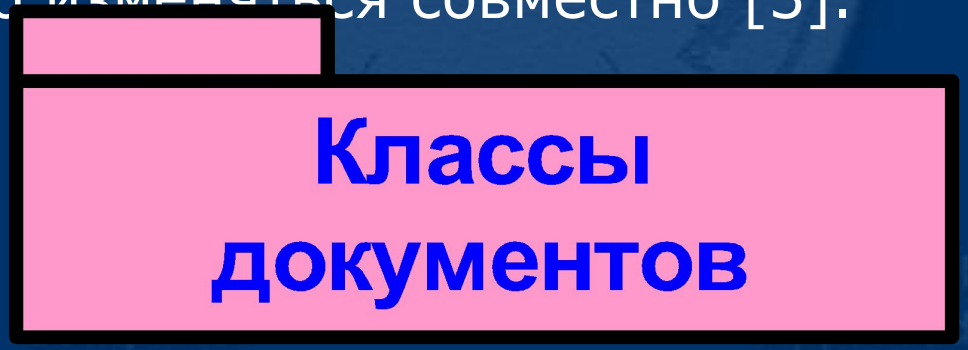
- **Интерфейс** определяет границу между спецификацией того, что делает абстракция, и реализацией того, как она это делает [3].
- **Интерфейс** – это набор операций, используемых для специфицирования услуг, предоставляемых классом или компонентом [3].
- Смысл использования: отделить детали реализации от функциональности. «Внешние» методы выносятся в **Интерфейс**.

Интерфейсы в UML



Пакеты в UML

- **Пакет** – структурная единица для группировки элементов модели, в частности, классов.
- **Пакет** – это способ организации элементов модели в более крупные блоки, которыми впоследствии позволяется манипулировать как единым целым [3].
- Хорошо спроектированный пакет группирует семантически близкие элементы, которые имеют тенденцию изменяться совместно [3].

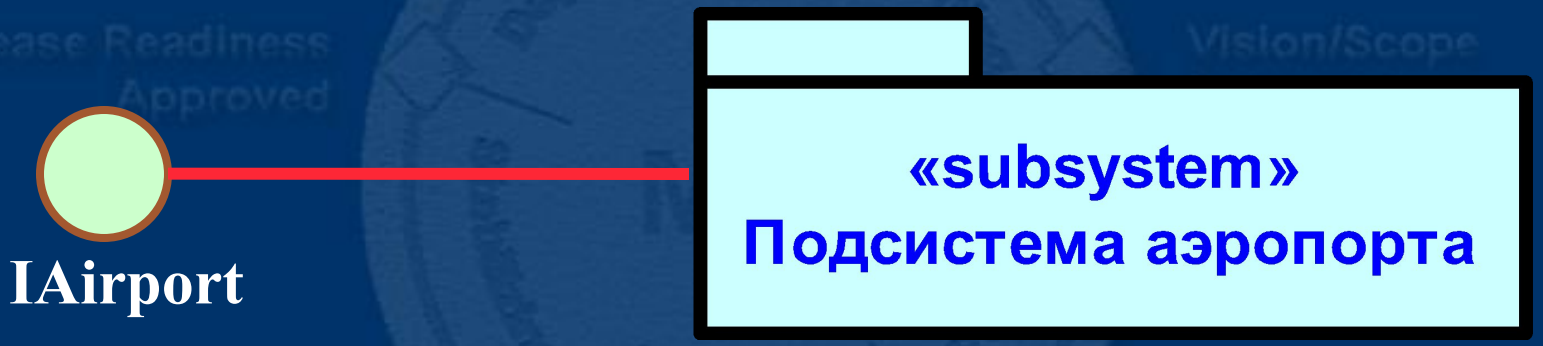


Подсистемы

- **На этапе проектирования** системы классы и пакеты могут объединяться в **подсистемы**.
- Подсистема – структурная единица.
- Каждая подсистема имеют свою **область ответственности** и реализует некоторую **функциональность**.
- Подсистема реализует **Интерфейс**, который описывает ее поведение.
- Примеры: подсистема бронирования билетов; подсистема доступа к данным...

Подсистемы в UML

Подсистема реализует интерфейс

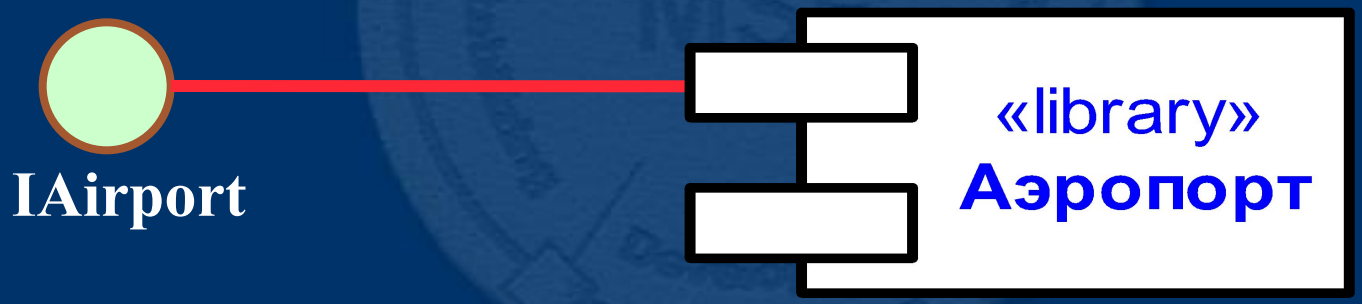


Компоненты

- **Компонент** – физическая заменяемая часть системы, совместимая с одним набором интерфейсов и обеспечивающая реализацию какого-либо другого [3].
- Компонент может разрабатываться и тестироваться независимо от системы.
- Виды компонентов:
  - Исходные файлы (.cpp, .h, .java…).
  - Бинарные файлы (.dll, .ocx…).
  - Исполняемые файлы (.exe).

Компоненты в UML

- По смыслу компонент – реализация подсистемы.
- На этапе проектирования – подсистемы. На этапе реализации – компоненты.



Комментарии (заметки) в UML

Отношения между элементами модели в UML

- **Отношения**:
  - **Зависимость**;
  - **Ассоциация**;
  - **Обобщение** (наследование);
  - **Реализация** (для Интерфейса).
- Отношения показывают наличие связей между элементами модели и семантику этих связей.

Зависимость в UML

- **Зависимость** – связь между сущностями (классами, объектами).
- **Зависимость** показывает, что изменения в одной сущности могут повлиять на другую сущность.

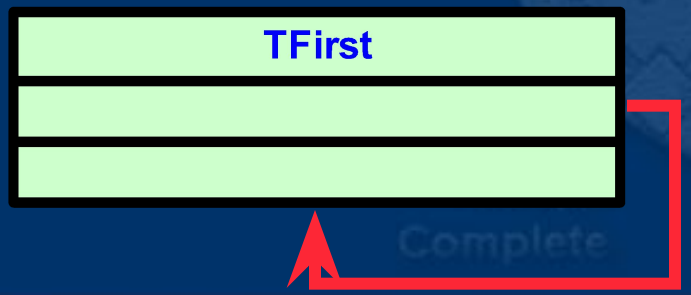
TFirst зависит от TSecond

- **Зависимость** – не структурная связь. Возникает через локальную, глобальную переменные или параметр метода.

Ассоциация в UML

- **Ассоциация** – связь между сущностями (классами, объектами).
- **Ассоциация** показывает наличие структурной связи между экземплярами (объектами).
- Связь через поле класса. Направление может быть не указано (двусторонняя связь).

TFirst содержит поле, связанное с TSecond



TFirst содержит поле, связанное с TFirst

Направление и навигация

- Заметим, что наличие направления связано с понятием **Навигация**.
- Навигация означает, что в направлении стрелки один объект «**видит**» другой, в то время как обратное не выполняется.

TFirst видит TSecond

Кратности в UML

Кратность – способ конкретизации характера отношения. Показывает тип отношения 1:1, 1:M, N:1, N:M.

Каждому контейнеру соответствует M элементов.
Каждому элементу соответствует 1 контейнер.

Таблица кратностей в UML

Вид кратности	Значение
* или **0..***	**≥0**
1..*	**≥1**
	обычно 0 или 1
1	**Ровно 1**
3,5..6	**{3,5,6}**

Частные случаи ассоциаций: агрегация и композиция

- **Агрегация** предполагает, что 0 или более объектов одного типа включены в 1 или более объектов другого типа.
- **Композиция** – вариант агрегации, в котором каждый объект второго типа может быть включен ровно в 1 объект первого типа.

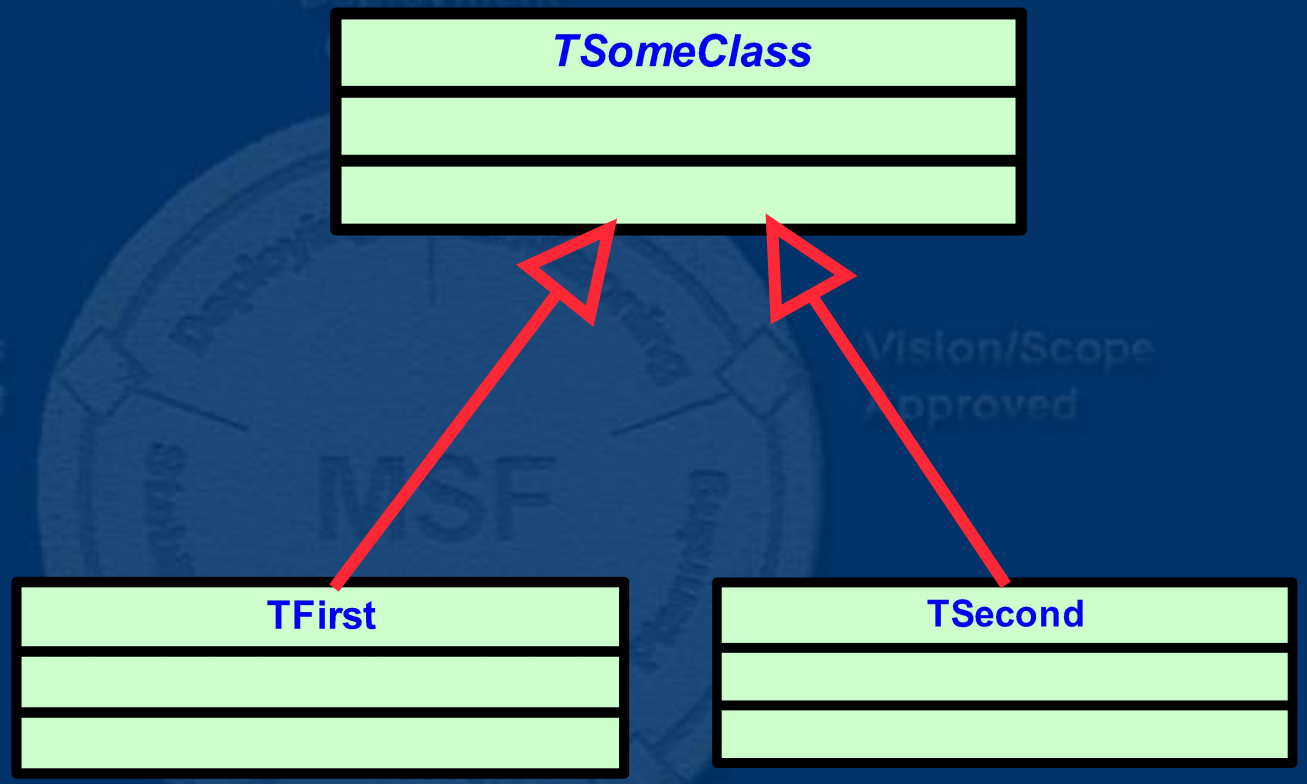
Композиция

Агрегация

Обобщение (наследование)

Предок

Потомки



Что дальше?

Следующая тема:

Microsoft Solutions Framework

Литература к лекции

1. **И. Соммервиль**. Инженерия программного обеспечения, 6 изд. – И.д. "Вильямс", 2002.
2. **Г. Буч**. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. Второе издание. – Бином, 1998.
3. **Г. Буч, Дж. Рамбо, А. Джекобсон.** UML. Руководство пользователя. – ДМК-Пресс, Питер, 2004.
4. **G. Booch, J. Rumbaugh, I. Jacobson**. The Unified Modeling Language Reference Manual – Second Edition, Addison-Wesley, 2004.
5. www.uml.org
6. www.wikipedia.org