

# РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Компонентные системы

- ① Что такое CORBA?
- ① Какие преимущества ее применения по сравнению с удаленными вызовами методов вы можете назвать?
- ① Почему CORBA не прижилась на рынке распределенных корпоративных систем?
- ① Каковы достоинства и недостатки использования менеджеров сообщений?
- ① Чем отличается просто «перенос кода» от концепции «мобильных агентов»?

# ОСНОВЫ КОМПОНЕНТНЫХ ПРОГРАММНЫХ СИСТЕМ

# КОМПОНЕНТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД

- ◎ Компонентно-ориентированный подход – развитие объектно-ориентированного. Создан для проектирования и реализации *крупных и распределенных программных систем (корпоративных приложений)*
- ◎ **С точки зрения КОП** программная система – это набор компонентов с четко определенным интерфейсом.
- ◎ Изменения в систему вносятся путем создания новых компонентов или изменения старых.
- ◎ **Наследование реализации запрещено. Наследуется только интерфейс.**

# ПРОГРАММНЫЙ КОМПОНЕНТ

- ◎ **Программный компонент** – это автономный элемент программного обеспечения, предназначенный для *многократного использования*, который может распространяться для использования в других программах в виде скомпилированного кода.
- ◎ Подключение к этим программам осуществляется с помощью интерфейсов компонента.
- ◎ Взаимодействие с программной средой осуществляется посредством инициации и реагирования на события.

# ПРЕИМУЩЕСТВА ИСПОЛЬЗОВАНИЯ ПРОГРАММНЫХ КОМПОНЕНТОВ

- ◎ **Оптимизация стоимости и скорости разработки** программной системы за счет использования готовых блоков
- ◎ Повышение эффективности повторного использования кода
- ◎ Повышение масштабируемости программной системы

# ТРЕБОВАНИЯ К КОМПОНЕНТАМ

К разработке программных компонентов предъявляются серьезные требования:

- ⊙ полная документированность интерфейса;
- ⊙ тщательное тестирование;
- ⊙ тщательный анализ входных значений;
- ⊙ возврат адекватных и понятных сообщений об ошибках;
- ⊙ необходимость предусмотреть возможность неправильного использования.

# РАСПРЕДЕЛЕННЫЕ КОМПОНЕНТНЫЕ ТЕХНОЛОГИИ

- ◎ Цель:
  - ◎ интеграция сервисов для приложений на базе различных платформ;
  - ◎ обеспечение взаимодействия: обеспечить прозрачный механизм общения и обмена данными между элементами РВС.



# РЕАЛИЗОВАННЫЕ ПОДХОДЫ

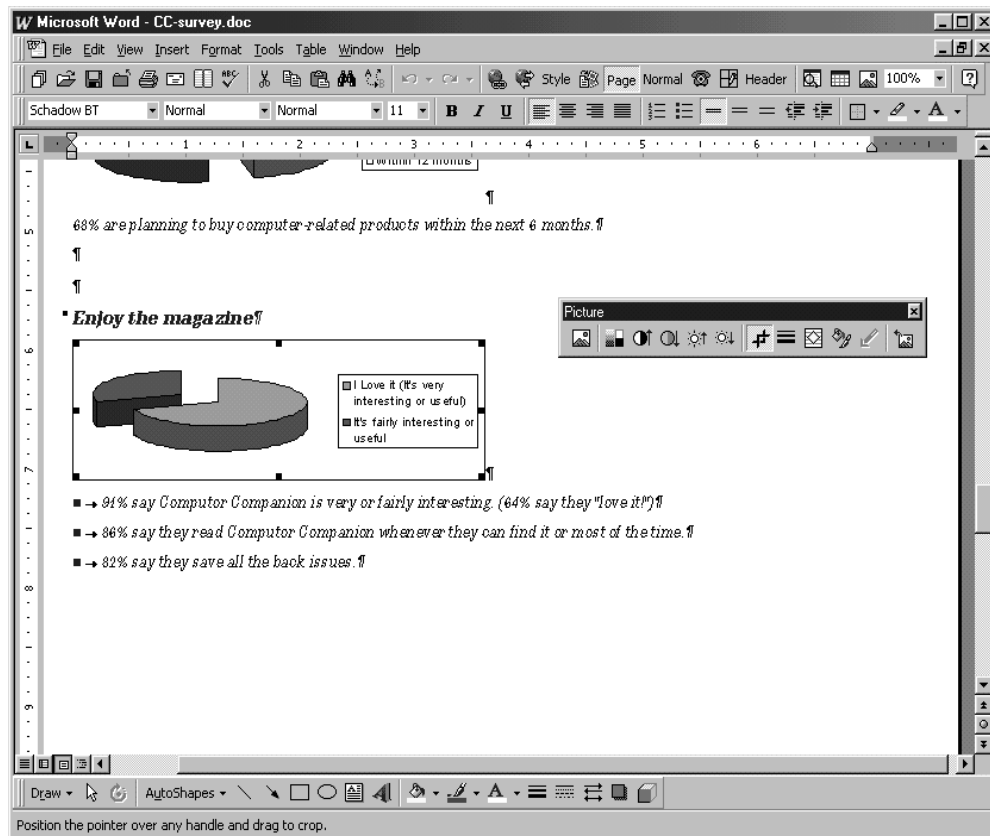
- ◎ **Microsoft:** DDE, COM, OLE, DCOM and ActiveX
- ◎ **Sun:** JavaBeans, Enterprise JavaBeans, JEE

# ПОДХОД MICROSOFT

COM, DCOM, ActiveX

# СРЕДА MICROSOFT (СЕРЕДИНА 90-Х)

- 🎯 Возможность интегрировать данные одних приложений в другие (в Word вставляется диаграмма из Excel)



# DDE (DYNAMIC DATA EXCHANGE)

- ◎ Первое решение обмена данными между приложениями:
  - ◎ Объединение этих приложений в единый программный комплекс
  - ◎ Пример - *Microsoft Works*
  
- ◎ Альтернатива : Dynamic Data Exchange (DDE)
  - ◎ Различные приложения под Windows смогли обмениваться данными посредством ссылок
  - ◎ **Ограничение:** не возможно изменить данные в «дочернем» приложении, необходимо вызывать «родительское»
  - ◎ Если данные перемещаются в файловой системе – все ссылки нарушаются.

# OLE (1991)

- ◎ Технология связывания и внедрения объектов (**Object Linking and Embedding**)
  - ◎ *Связывание* – на основе DDE
  - ◎ *Внедрение* позволяет пользователям вставить элемент данных в Word и сохранить его там
  - ◎ *Связывание* дешевле, если объем данных большой
  - ◎ *Внедрение* поддерживает составные документы (“документно-ориентированные” вычисления)
- ◎ Компонентные контейнеры могут быть повторно использованы многими приложениями
- ◎ Но компоненты не являются программно-независимыми  
=> OLE – это решение только для Windows.

# Технология COM (1993)

- ◎ Технология OLE – это набор API для создания и отображения документа
  - ◎ Обеспечивает совместное использование не только данных, но и кода
  
- ◎ Объектная Модель Компонентов (Component Object Model - COM)
  - ◎ Протоколы COM позволяют объединить множество компонентов для создания единого приложения:
  - ◎ Например, текстовый редактор может сказать электронной таблице: *“пользователь только что нажал на таблицу, поэтому необходимо запуститься, найти данные, и, как только закончишь работу – дай мне знать.”*
  
- ◎ COM в настоящее время включает OLE как часть более широкой концепции
  - ◎ OLE становится блоком стандартных интерфейсов COM

# ACTIVEХ (1996)

- ◎ В 1996 Microsoft перевыпускает OLE и COM под именем ActiveХ
- ◎ ActiveХ – это ответ Microsoft на технологию Java
  - ◎ Элемент ActiveХ – эквивалент Java-апплета
- ◎ Приложение, написанное на ActiveХ может работать в любом месте, поддерживающим данную технологию
- ◎ Данный компонент называется «Элемент ActiveХ» и часто используется для привязки программы к определенной веб-странице

# РЕАЛИЗАЦИЯ ACTIVEX

- ◎ Элемент ActiveX может быть создан на базе различных языков, включая C++ и Visual Basic.
- ◎ Изначально, ActiveX был привязан к Windows
  - ◎ Позднее появились реализации под Mac и Unix/Linux
- ◎ Проблемы безопасности: ActiveX получает полный доступ к системе (нет песочницы)
  - ◎ Но могут быть подписаны для авторизации



# ENTERPRISE JAVA BEANS

JavaBeans, J2EE

# JAVABEANS API

- ◎ **“Java Bean – это многократно используемый программный компонент, которым можно визуально манипулировать IDE”**
- ◎ **Компоненты** являются автономными единицами программного обеспечения, поддерживающими многократное использование
  - ◎ которые могут быть объединены в составные компоненты, апплеты, приложения и сервлеты.
- ◎ компоненты JavaBean известны как *Beans*.

# КОМПОНЕНТЫ И АРХИТЕКТУРА ПО

## ◎ Классы vs. компоненты:

- Иерархии классов + кооперация объектов = детализированный дизайн
- компоненты + кооперация = архитектура

## ◎ Класс vs. JavaBean:

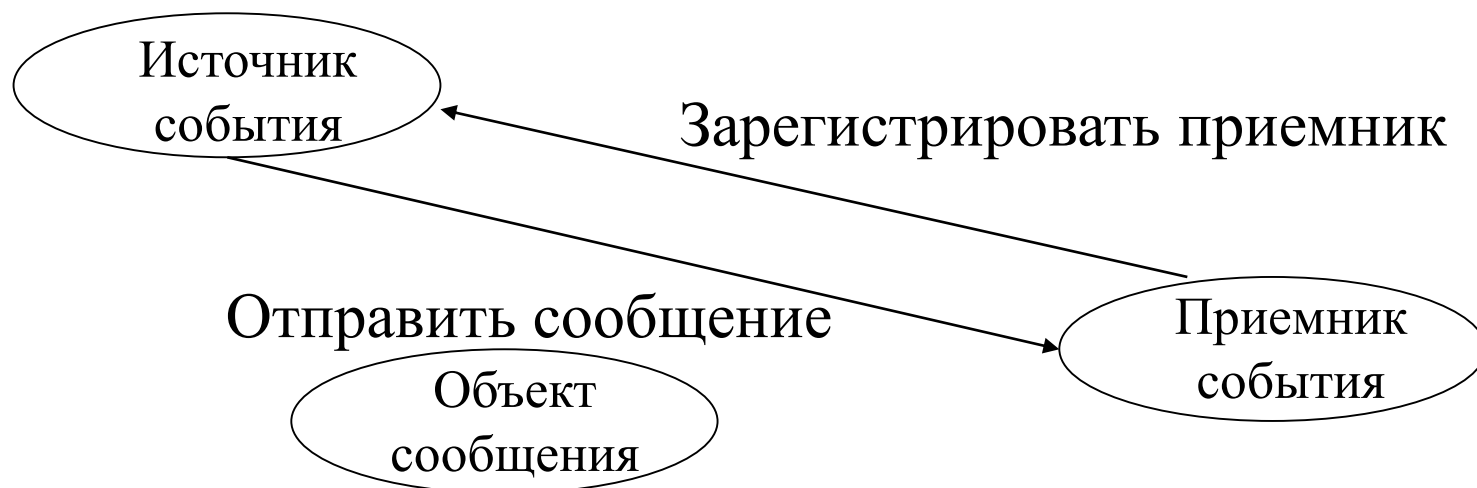
- Класс == кирпич, гвоздь
- JavaBean = блок стены, крыша, комната
- Клиентское приложение == здание
- Архитектура не должна концентрироваться на гвоздях и кирпичах!

# ЭЛЕМЕНТЫ JAVA BEAN

- ⊙ **Свойства:** Beans можете описывать атрибуты, которые определяют их внешний вид и поведение
- ⊙ **Сохранность** (сериализация): можно сохранять и извлекать данные JavaBeans из внешних источников и по сети
  - ⊙ `java.io.Serializable` поддерживает чтение/запись состояния в/из потока
  - ⊙ сохранять значения переменных экземпляра
  - ⊙ сохранять версии класса (хэш для имени класса, поля, методы)
- ⊙ **События:** Beans могут обмениваться сообщениями о событиях
- ⊙ **Самоописание** (рефлексия) система может самостоятельно понять методы работы Bean
  - ⊙ Reflection API (`java.lang.reflect`)
  - ⊙ Поддержка получения информации о классе, методах, конструкторах, полях во время выполнения приложения

# ОБЩЕНИЕ ЧЕРЕЗ СОБЫТИЯ

- ◎ Сообщение отправляется с одного объекта на другой.
- ◎ Может быть несколько приемников одного сообщения



# ENTERPRISE JAVA BEANS (EJB)

- ◎ **Серверный компонент**
  - ◎ Содержит **бизнес-логику** приложения
  - ◎ Программные клиенты реализуют бизнес-логику посредством вызова методов EJB
- ◎ Такой подход освобождает разработчика клиентского ПО от операций с системой на уровне приложения
- ◎ Позволяет разработчику JavaBean сконцентрироваться на логике приложения.

# ЧТО ТАКОЕ EJB?

- ◎ EJB – специализированный, не визуальный JavaBean, который работает на сервере.
- ◎ Технология EJB поддерживает разработку приложений на основе **распределенной объектной архитектуры**, в основе которой большая часть логики приложения *перемещается от клиента на сервер*.
- ◎ Является частью спецификации JEE (Java Platform, Enterprise Edition), обеспечивающей:
  - ◎ масштабируемость корпоративных систем;
  - ◎ целостность данных во время работы

# ПОРТИРУЕМОСТЬ КОМПОНЕНТОВ

- ◎ Технология EJB обеспечивает поддержку подхода WORA («Write Once, Run Anywhere») – написано единожды, исполняется везде.
- ◎ Компоненты EJB **полностью переносимы** между EJB-совместимыми серверами приложений от любого поставщика.
- ◎ Среда EJB автоматически сопоставляет новые компоненты для работы с сервисами **любого** сервера приложений.



# ОБЗОР ТЕХНОЛОГИИ EJB

- ◎ **Модель компонентов EJB** расширяет модель JavaBeans для поддержки серверных компонентов.
- ◎ **Серверные компоненты** – это поддерживающие многократное использование блоки, содержащие основные функциональные возможности конкретного приложения.
- ◎ EJB могут быть **собраны и настроены** во время развертывания использованием средств, предоставляемых EJB-совместимом сервере приложений.

# Возможности EJB

- ◎ EJB позволяет из любого Java-класса сделать **распределенный, безопасный, транзакционный класс**
- ◎ Возможно взять любой **источник данных** и **представить его в виде коллекции Java-объектов**
- ◎ Устраняет различие между данными из базы данных и любыми другими источниками
- ◎ Доступ ко всей информации осуществляется посредством Java-объектов

# ПРЕИМУЩЕСТВА EJB

- ◎ Разработчики **могут сосредоточиться на написании бизнес-логики**, а не на низкоуровневой инфраструктуре (доступ к данным, параллелизм, операции, многопоточное программирование и т.д.)
  - ◎ Уменьшается время разработки
  - ◎ Уменьшается сложность систем
  - ◎ Улучшается качество
- ◎ Код находится в разделяемых, серверных объектах, что позволяет значительно повысить **повторное использование кода**.

# АРХИТЕКТУРА ЕЈВ

# СПЕЦИФИКАЦИЯ EJB

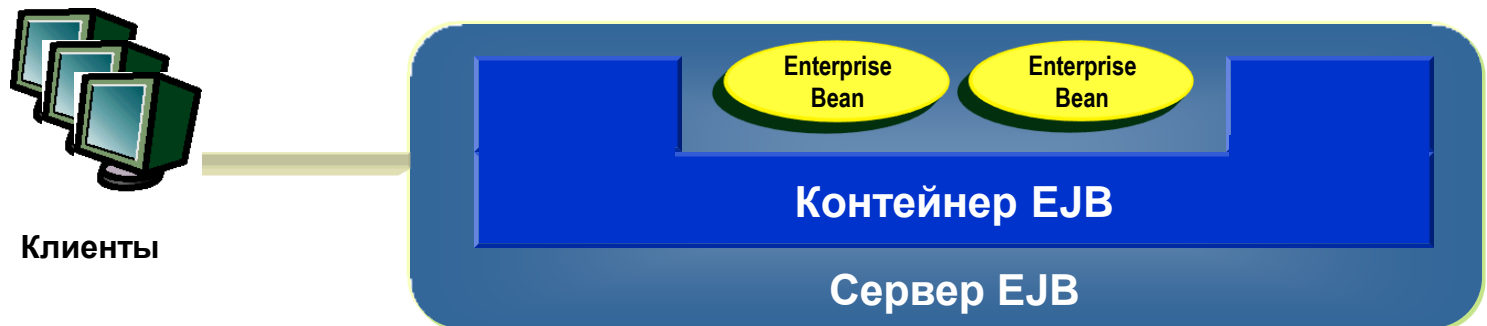
- ◎ EJB определяет модель серверных компонентов для разработки и развертывания Java-приложений на основе многоуровневой архитектуры распределенных объектов
- ◎ Спецификация Enterprise JavaBeans определяет:
  - ◎ Модель контейнера
  - ◎ Определение сервисов, которые контейнер должен предоставлять EJB, и наоборот
  - ◎ Как контейнер должен управлять EJB

# АРХИТЕКТУРА ENTERPRISE JAVABEANS

30

**Архитектура EJB определяет обязанности  
и методы взаимодействия между  
сущностями EJB**

- ◆ Серверы EJB
- ◆ Контейнеры EJB
- ◆ Корпоративные Beans
- ◆ Клиенты EJB



## Обеспечивает Среду исполнения

- ◎ Сервер EJB обеспечивает системные службы и управляет ресурсами
  - ◎ Управление процессами и потоками
  - ◎ Управление системными ресурсами
  - ◎ Обработка соединений с базами данных (+кеширование)
  - ◎ API управления

# КОНТЕЙНЕР EJB

**Предоставляет среду выполнения для Корпоративных Bean**

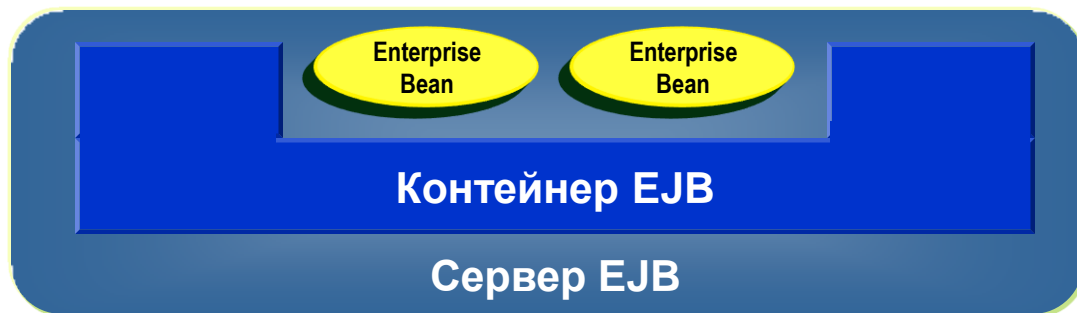
- ◎ Обеспечивает размещение EJB
- ◎ Обеспечивает базовые сервисы для EJB
  - ◎ Именованное
  - ◎ управление жизненным циклом
  - ◎ Сохранность (управление состоянием)
  - ◎ Управление транзакциями
  - ◎ Безопасность
- ◎ Обычно реализован поставщиком сервера EJB





# КОРПОРАТИВНЫЕ BEAN

- ◎ Специальный Java-класс, в котором живет бизнес-логика
  - ◎ Может быть сгенерирован либо написан самостоятельно
- ◎ Распределен в сети
- ◎ Обеспечивает работу с транзакциями
- ◎ Обеспечивает безопасность
- ◎ Поставщики сервера EJB предоставляют инструменты, которые автоматически обеспечивают распределение и безопасность



# КЛИЕНТЫ EJB

- ⊙ Доступ клиентов контролируется контейнером, в котором развернут корпоративный Bean
- ⊙ Клиенты находят EJB посредством Java Naming and Directory Interface (JNDI)
- ⊙ RMI является стандартным методом для доступа к Bean по сети



# УНИКАЛЬНОСТЬ ПОДХОДА EJB

## Декларативная Модель Программирования

- ⊙ Спецификация EJB определяет модель контейнера, в которой общие службы объявляются, не программируются
  - ⊙ Во время **развертывания**, контейнер просматривает атрибуты EJB и выявляет сервисы, необходимые ему для работы, и обеспечивает Bean необходимой функциональностью.
  - ⊙ Во время **работы**, контейнер перехватывает все обращения к объекту
    - ⊙ Обеспечивает обработку транзакций, потоков и безопасности до вызова метода
    - ⊙ Вызывает метод объекта
    - ⊙ Производит зачистку после вызова

# КОМПОНЕНТЫ EJB

# СПЕЦИФИКАЦИЯ ЕЈВ

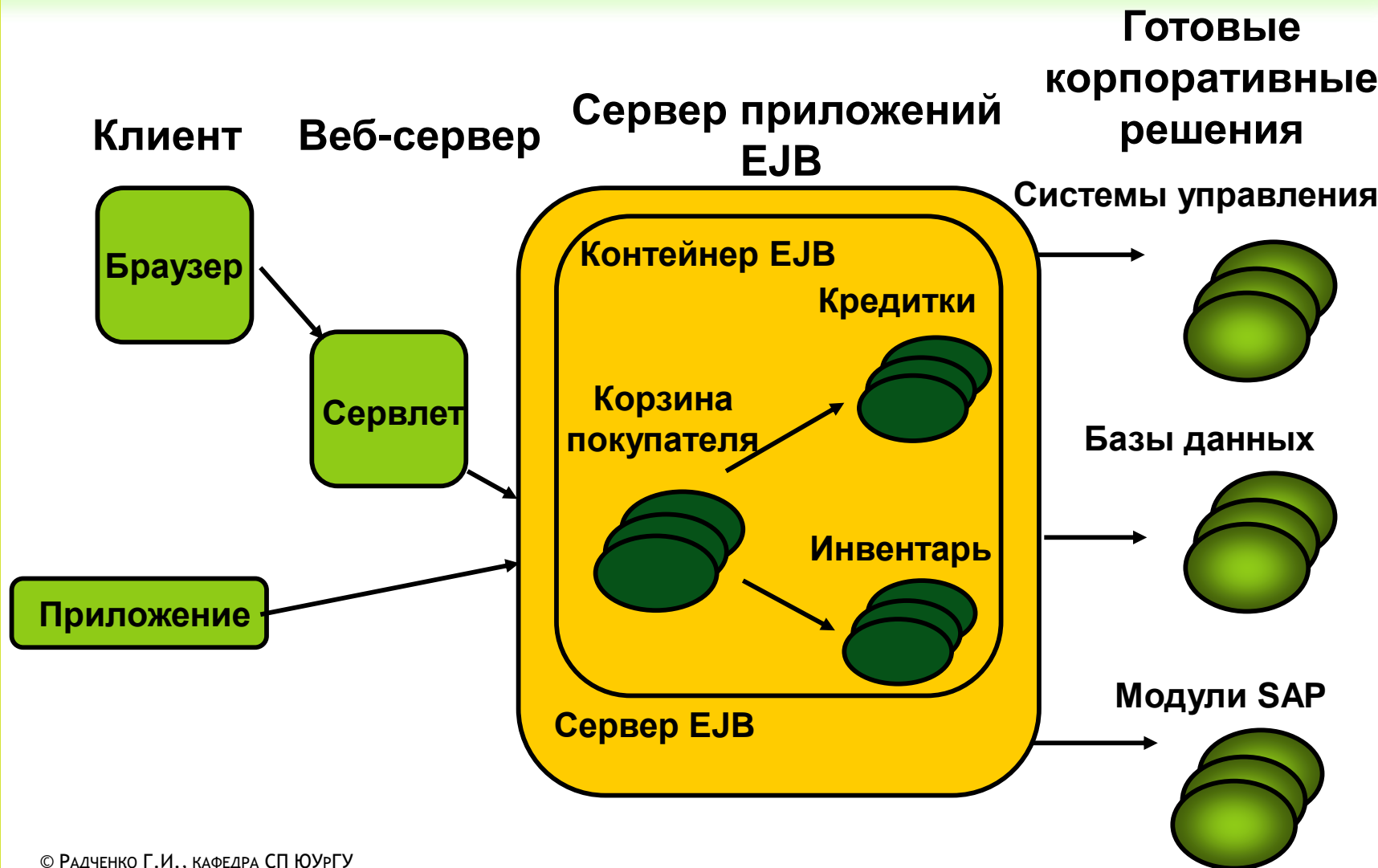
~200 Страниц технических материалов  
для поставщиков ЕЈВ

- Основные цели резила
- Базовые роли и сценарии
- Базовые возможности
- Сессии и сущности
- Транзакции, Исключения, Распределение
- Обязанности EJB Bean и контейнера
- Описание API



**Поставщики ЕЈВ  
Должны сделать все САМИ**

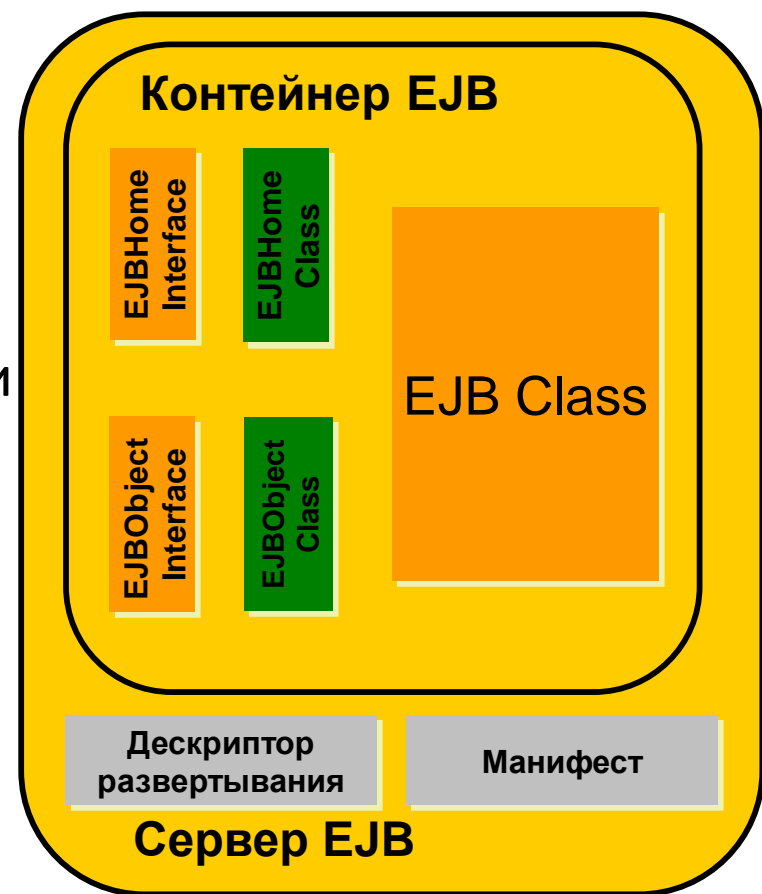
# СЦЕНАРИЙ EJB



# КОМПОНЕНТЫ EJB

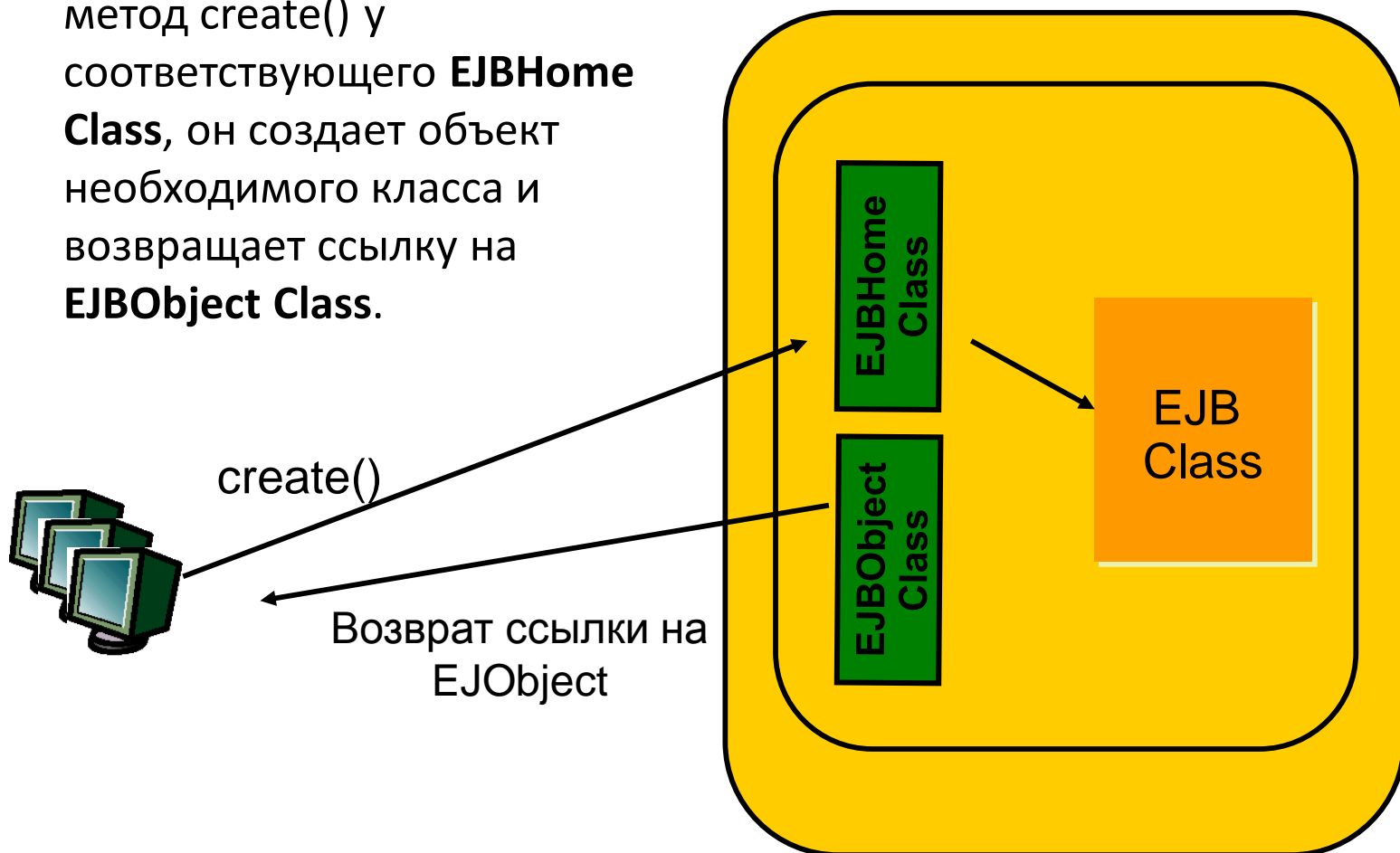
- ◎ Класс-Bean создается разработчиком
- ◎ Интерфейсы EJBHome и EJBObject обеспечивают доступ к классу EJB
- ◎ Дескриптор развертывания и манифест описывают безопасность Bean-а и его метод работы с транзакциями

- Создается разработчиком
- Генерируется при разработке
- Генерируется при развертывании



# ИНСТАНЦИРОВАНИЕ EJB

- ⊙ Когда клиент вызывает метод `create()` у соответствующего **EJBHome Class**, он создает объект необходимого класса и возвращает ссылку на **EJBObject Class**.





# Типы EJB-компонент

- ⊙ Выделяют 2 основных типа EJB-компонент:
  - ⊙ **Компоненты данных** (сущностные компоненты, entity beans) - представляют данные приложения и основные методы работы с ними.
  - ⊙ **Сеансовые компоненты** (session beans) - представляют независимую от пользовательского интерфейса и конкретных типов данных логику работы приложения (бизнес-логику)

# СРАВНЕНИЕ КОМПОНЕНТОВ ДАННЫХ И СЕАНСОВЫХ КОМПОНЕНТОВ

## Сеансовые компоненты

- ⊙ Одна сущность на одного клиента
- ⊙ С коротким сроком службы
- ⊙ Может быть любого класса Java
- ⊙ Могут обеспечивать обработку транзакций

## Компоненты данных

- ⊙ Представляет базовый объект данных или контекст (Единый для множества клиентов)
- ⊙ Долгоживущие, сохраняют свое состояние
- ⊙ Может быть классом, отображаемым на данные (база данных)
- ⊙ Всегда должны обрабатывать транзакции

# РЕАЛИЗАЦИЯ ЕЈВ

Реализация EJB входит в базовые возможности, предоставляемые JEE-серверами

- ◎ Open Source:
  - ◎ Apache Geronimo (использует Apache Tomcat или Jetty)
  - ◎ GlassFish (использует Apache Tomcat)
  - ◎ JBoss Application Server
  - ◎ Resin
- ◎ Коммерческие системы:
  - ◎ IBM WebSphere
  - ◎ Oracle WebLogic