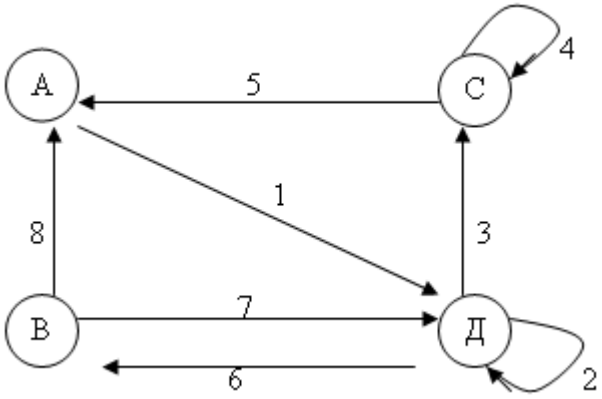
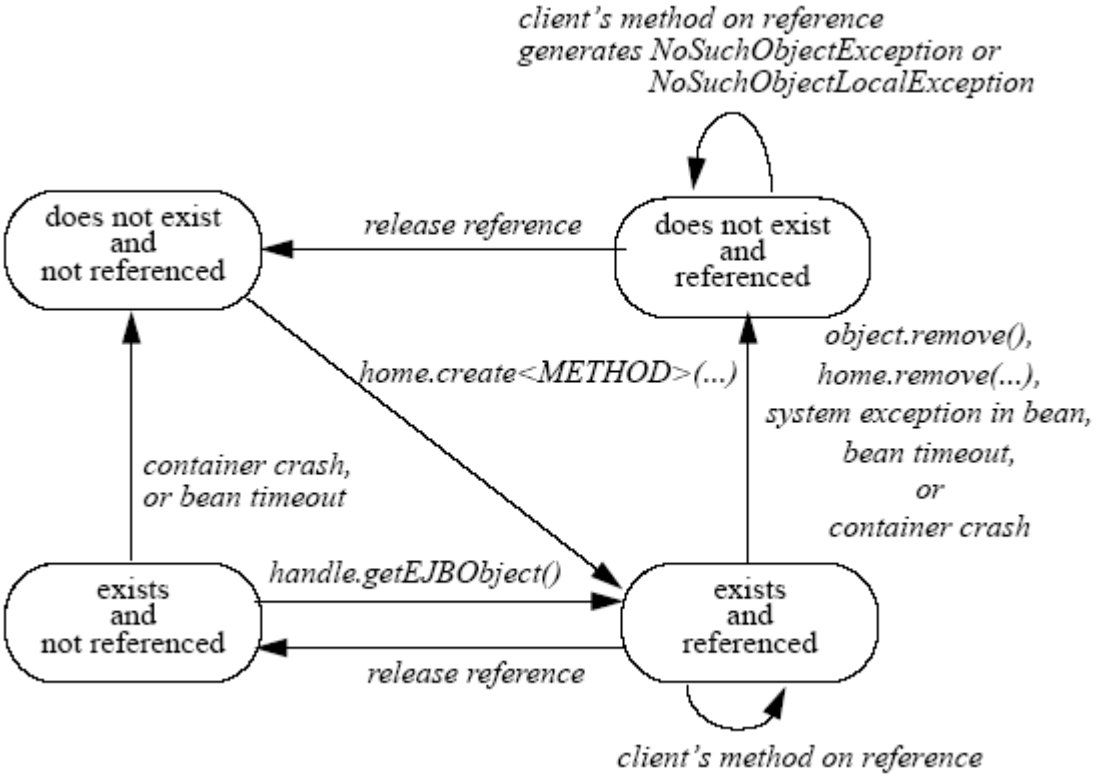
Session Beans

Сессионные компоненты делятся на два подвида:
1) сессионные компоненты без состояния (stateless session beans);
2) сессионные компоненты с состоянием (stateful session beans).
Эти компоненты сильно отличаются по жизненному циклу (с точки зрения контейнера) и назначению, но их сближают требования, предъявляемые к разработчику компонента.
SSB предназначены для реализации некоторого функционального API приложения. Между вызовами методов компонента внутри него не должна сохраняться никакая информация о состоянии. Таким образом, каждый конкретный экземпляр SSB сам по себе ничего не «помнит», а для реализации сохраняемого состояния он использует другие средства (компоненты соответствующего вида, БД, файлы, JNDI-каталог). SSB предназначены для моделирования процессов или бизнес-операций. Как правило, они реализуют так называемый «фасад» приложения, с которым взаимодействуют клиенты и который скрывает от них внутренние слои middleware-приложения.
SFSB используются для хранения данных в рамках сеанса работы пользователя с приложением и организации работы с этими данными. При этом данные сеанса не требуется сохранять в постоянном хранилище, так как они актуальны и используются исключительно в рамках сеанса. Данные сеанса реализуются в виде полей класса компонента. Бизнес-методы могут рассчитывать на то, что эти данные будут сохраняться между последовательными вызовами, сделанными в рамках сеанса. Более того, в отличие от SSB вызовы, сделанные клиентом на одном и том же экземпляре компонента, обязательно будут обработаны одним и тем же экземпляром класса SFSB-компонента.
SFSB имеют два преимущества перед использованием веб-сессии для хранения данных сеанса:
1) Для SFSB контейнер поддерживает сервис управления памятью, механизм работы которого подобен механизму виртуальной памяти: при повышенной нагрузке на сервер приложений экземпляры SFSB-компонентов в соответствии с той или иной политикой управления памятью могут удаляться из памяти с сохранением их состояния в сериализованной форме в некотором постоянном хранилище. Компонент в этом случае называется пассивным. При обращении к пассивному компоненту со стороны клиента, он автоматически восстанавливается в оперативной памяти. Данный сервис управления памятью для веб-сессии на уровне спецификации не предусмотрен, хотя может быть реализован специфическими для контейнера средствами. Приложение, использующее эти средства, непереносимо.
2) Механизм веб-сессии специфичен для веб-контейнера и может быть использован лишь в веб-приложении и веб-сервисах. SFSB являются EJB-компонентами и могут быть использованы как веб-, так и GUI-клиентами, совместимыми с платформами Java или CORBA.
Спецификация не закрепляет тип постоянного хранилища, используемого для хранения пассивных SFSB-компонентов. Это может быть как файловое хранилище, так и объектная СУБД или XML-хранилище. На практике используется объектная СУБД. Не регламентирована также и политика управления памятью, в соответствии с которой осуществляется перевод того или иного экземпляра компонента в пассивное состояние. Эти вопросы отданы на откуп разработчикам контейнеров, настройки выполняются средствами специфичных для контейнера дескрипторов.

С точки зрения клиента жизненный цикл SSB и SFSB совпадает и имеет следующий вид.





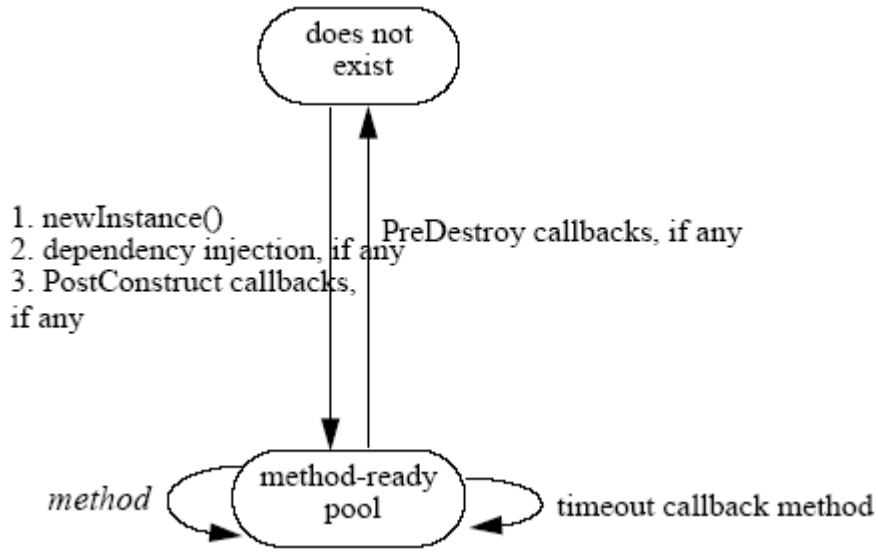
Состояния:
A – экземпляр компонента не существует, и у клиента нет на него ссылки;
B – экземпляр компонента существует, но у клиента нет на него ссылки;
C – экземпляр компонента не существует а у клиента осталась на него ссылка;
D – экземпляр компонента существует у клиента есть на него ссылка.
Переходы:

1. происходит при вызове метода create домашнего интерфейса;
2. клиент вызывает бизнес-метод компонента;
3. происходит в одном из четырех случаев:
    a. клиент вызвал метод remove экземпляра компонента;
    b. вызов бизнес-метода привел к возникновению исключения;
    c. экземпляр компонента был уничтожен контейнером по таймауту (для SFSB);
    d. системный сбой контейнера;
4. соответствует вызову бизнес-метода по ссылке, которая не соответствует никакому реально существующему экземпляру компонента, при этом выбрасывается исключение NoSuchObjectException или NoSuchLocalObjectException в зависимости от вида представления;

5. ссылка, полученная клиентом на экземпляр компонента, вышла из области видимости;
6. клиент сохраняет ссылку на экземпляр компонента во внешнем хранилище и теряет ее;
7. клиент восстанавливает ссылку из внешнего хранилища;
8. контейнер уничтожает экземпляр компонента по тайм-ауту (SFSB) либо контейнер завершает свое исполнение.

Особенности ЖЦ сессионных компонентов, созданных с помощью упрощенного API EJB 3.0: Компонент существует с момента получения ссылки на его бизнес-интерфейс либо при помощи инъекции зависимости (dependency injection), либо через поиск бизнес-интерфейса в JNDI. Клиент, обладающий ссылкой на бизнес-интерфейс сессионного компонента может вызывать методы данного интерфейса и передавать ссылку в качестве параметра или результата метода бизнес-интерфейса. Клиент может удалить сессионный компонент с состоянием, вызвав метод бизнес-интерфейса, отмеченный (аннотированный) как Remove-метод. ЖЦ сессионного компонента без состояния не требует его явного удаления клиентом. Удаление экземпляра такого компонента выполняется контейнером прозрачно для клиента.

Жизненный цикл Stateless Session Bean с точки зрения контейнера



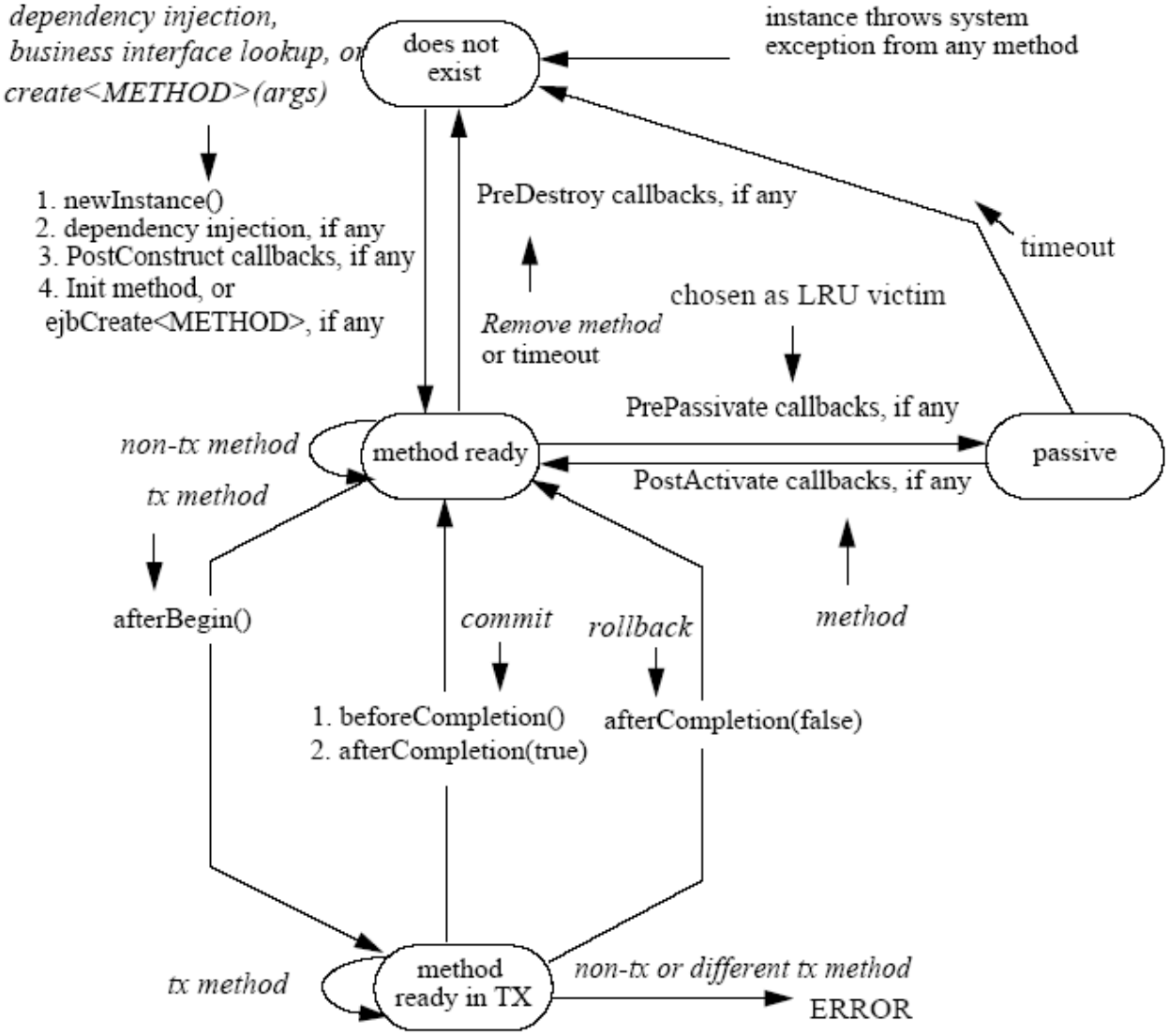
Переходы:

1. Выполняется, когда контейнер принимает решение расширить пул экземпляров компонента, при этом выполняется три действия:
    a. создание экземпляра и вызов конструктора по умолчанию;
    b. инъекция зависимостей;
    c. вызов определенных для данного компонента PostConstruct-методов, если они есть;
2. Происходит, когда контейнер принимает решение сократить пул экземпляров компонента. Перед удалением экземпляра вызываются определенные для данного компонента PreDestroy-методы, если они есть;
3. Выполняется, когда контейнер выбирает экземпляр компонента из пула для обработки вызова бизнес-метода, сделанного клиентом.

Между жизненным циклом SSB с точек зрения клиента и контейнера есть некоторое соответствие: расширение пула экземпляров компонента может быть инициировано клиентом путем вызова метода create домашнего интерфейса, при этом отличительной особенностью SSB является то, что между вызовом клиентом метода create домашнего интерфейса и созданием контейнером нового экземпляра SSB нет жесткой связи. Сокращение пула может быть вызвано истечением настраиваемого тайм-аута. Здесь нет

жесткой связи между вызовом клиентом метода remove компонентного интерфейса и уничтожением экземпляра SSB.
Так как SSB не обладает состоянием, то есть с экземпляром не связано никаких данных, которые должны сохраняться между последовательными вызовами бизнес-методов одним и тем же клиентом, то с точки зрения контейнера все экземпляры одного и того же SSB равны между собой в том плане, что любой из них может быть использован для обработки вызова бизнес-метода, сделанным любым из клиентов.

Жизненный цикл Stateful Session Bean с точки зрения контейнера.



Состояния:
A – компонент не существует;
B – компонент существует и находится вне контекста транзакции;
C – компонент существует и находится в контексте какой-либо транзакции;
D – компонент существует, но находится в пассивном состоянии (данные сброшены в хранилище).

Переходы:

1. Выполняется контейнером, когда клиент вызывает метод create домашнего интерфейса или получает ссылку на его бизнес-интерфейс либо при помощи инъекции зависимости (dependency injection), либо через поиск бизнес-интерфейса в JNDI. Контейнер создает экземпляр класса компонента (при этом вызывается конструктор без параметров), выполняет инъекцию зависимостей, вызывает определенные для

данного компонента PostConstruct-методы, если они есть, и выполняет соответствующий Init-метод или метод ejbCreate<METHOD>(). В отличие от SSB-компонентов SFSB-компоненты могут иметь несколько create-методов, которые отличаются параметрами и суффиксом в имени метода.

2. Выполняется, когда клиент вызывает метод бизнес-интерфейса, помеченный аннотацией Remove, или метод remove домашнего или компонентного интерфейса, либо когда вышел тайм-аут времени жизни сессионного компонента. Контейнер вызывает определенные для компонента PreDestroy-методы, а потом удаляет его. Чтобы обеспечить гарантированное уничтожение данных сессии в случае, когда этого не может сделать клиент путем явного вызова метода remove, для SFSB предусмотрено автоматическое удаление экземпляра по некоторому настраиваемому тайм-ауту, в течение которого клиент не обращался к компоненту. Невозможность явного уничтожения компонента клиентом может быть вызвана разрывом соединения (при взаимодействии через удаленное представление), ошибкой при исполнении клиента и т.д. Автоматическое уничтожение по тайм-ауту возможно только тогда, когда компонент находится вне транзакции (состояния B и D).
3. Выполняется, когда клиент вызывает бизнес-метод экземпляра компонента. Вызов бизнес-метода делегируется контейнером экземпляру класса компонента, при этом выполнение бизнес-метода происходит вне контекста транзакции.
4. Выполняется, когда клиент вызывает бизнес-метод, который должен выполняться в контексте транзакции. При этом перед делегированием вызова метода экземпляру класса компонента контейнером выполняется обращение к службе управления транзакциями (JTS – Java Transaction Service) для начала новой транзакции.
5. Транзакция, в рамках которой шло выполнение бизнес-методов, завершается подтверждением.
6. Выполняется, если транзакция, в рамках которой шло выполнение бизнес-методов, завершается откатом.
7. Выполняется, если клиент вызывает бизнес-метод, исполняющийся в рамках начавшейся ранее транзакции.
8. Выполняется, когда контейнер выбирает экземпляр компонента для удаления из памяти и сохранения его состояния во внешнем хранилище. Этот переход определяется политикой управления памятью сервера приложений и не инициируется явно клиентом, так как механизм управления памятью для него прозрачен. Не все поля, формирующие состояние компонента, могут быть сохранены в сериализованном виде, что вызывает необходимость предварительной подготовки экземпляра к сериализации. Она должна выполняться в рамках определенного для компонента обработчика состояния ЖЦ PrePassivate, который вызывается непосредственно перед сериализацией данных.
9. Выполняется контейнером, когда вызванный клиентом бизнес-метод относится к экземпляру компонента, который находится в пассивном состоянии. Состояние экземпляр при этом восстанавливается в памяти, а для инициализации полей, которые не были сохранены, контейнер вызывает обработчики состояния ЖЦ PostActivate, определенные для восстановленного экземпляра компонента. Инициализацию соответствующих полей компонент должен выполнить сам.
10. Выполняется, когда выходит тайм-аут экземпляра, при этом обработчик состояния ЖЦ PreDestroy не вызывается. (Пример с сохранением корзины заказа в БД – периодическая очистка БД от «покинутых» корзин)
11. Выполняется, когда клиент вызывает бизнес-метод, который должен выполняться вне контекста транзакции или в контексте другой транзакции. При этом возникает исключение.

В состояние A экземпляр переходит при возникновении системного исключения в любом из методов.

Требования к полям, формирующим Stateful SB.

По окончании работы обработчика состояния ЖЦ PrePassivate и перед сериализацией состояния сессионного компонента вкупе с состоянием всех определенных для него перехватчиков (interceptors) любое не помеченное ключевым словом transient поле класса компонента или класса его перехватчика, должно содержать одно из следующих значений:

1. сериализуемый объект, либо значение простого типа;
2. null;
3. ссылка на бизнес-интерфейс, удаленный интерфейс, удаленный домашний интерфейс, локальный интерфейс или локальный домашний интерфейс EJB-компонента;
4. ссылка на объект типа SessionContext;
5. ссылка на контекст имен java:comp/env или любой из его подконтекстов;
6. ссылку на объект типа javax.transaction.UserTransaction;
7. ссылка на фабрику соединения с менеджером ресурсов (например, источником данных javax.sql.DataSource);
8. ссылка на управляемый контейнером менеджер сущностей (объект типа javax.persistence.EntityManager);
9. ссылка на фабрику менеджеров сущностей (объект типа javax.persistence.EntityManagerFactory), полученная через инъекцию или поиск в JNDI;
10. ссылка на объект типа javax.ejb.Timer;
11. ссылка на объект, становящийся сериализуемым при замене содержащихся в нем объектов типов, указанных в пп. 3-9, на сериализуемые (например, коллекция ссылок на компонентные интерфейсы).

Не следует хранить в полях, помеченных модификатором transient, ссылки на объекты типов, указанных выше в пп. 3-9.

Важным практическим следствием является то, что при обработке события ЖЦ PrePassivate следует закрывать все активные соединения с менеджерами ресурсов (например, JDBC-соединения) и присваивать соответствующим полям значение null, а при обработке события ЖЦ PostActivate необходимо восстанавливать данные соединения.

Вопросы реализации Session Beans.

С точки зрения разработчика SSB – это класс компонента и, как минимум, одно из представлений: локальное или удаленное. Допускается обеспечивать оба представления. Каждое представление состоит из бизнес-интерфейса или из пары интерфейсов: домашнего и компонентного. Спецификация предъявляет требования ко всем из них.

Требования к бизнес-интерфейсу.

1) Тип представления, обеспечивающего данный интерфейс, можно указать с помощью аннотации Remote (удаленное представление) или Local (локальное представление).
2) Интерфейс не должен расширять интерфейс javax.ejb.EJBObject или javax.ejb.EJBLocalObject.
3) Если бизнес-интерфейс является удаленным, то он не обязан расширять java.rmi.Remote, и его методы в этом случае не должны выбрасывать исключение java.rmi.RemoteException.

С практической точки зрения из возможных представлений сессионных компонентов EJB 2.1 наибольший интерес представляет удаленное, так как создание удаленного адаптированного представления для компонента EJB 3.0 позволяет находить его в удаленном EJB-контейнере по протоколу CosNaming.

Требования к удаленному интерфейсу.

1) Интерфейс должен прямо или косвенно расширять интерфейс javax.ejb.EJBObject.

2) Интерфейс должен содержать объявления бизнес-методов, реализованных в классе компонента, которые разработчик хочет предоставить для использования через удаленное представление. Объявления методов должны соответствовать их объявлениям в классе компонента, а предложение throws должно включать исключение java.rmi.RemoteException.
3) Параметры и возвращаемые значения бизнес-методов, доступных через удаленное представление, должны иметь сериализуемый тип, то есть любой простой тип или тип класса, реализующий интерфейс java.io.Serializable. Это требование связано с семантикой передачи параметров и результатов через удаленное представление.

Требования к удаленному домашнему интерфейсу.

1) Интерфейс должен прямо или косвенно расширять интерфейс javax.ejb.EJBHome.
2) Интерфейс сессионного компонента без состояния должен содержать ровно один метод с именем create без параметров, которому может соответствовать (по параметрам, модификаторам и выбрасываемым исключениям) метод ejbCreate класса компонента.

   Интерфейс сессионного компонента с состоянием должен содержать один или несколько методов create<METHOD>, которым в классе компонента соответствуют методы с именами ejbCreate<METHOD> или методы, помеченные аннотацией Init. Все create-методы в домашнем удаленном интерфейсе возвращают значения типа удаленного интерфейса данного компонента, а соответствующие им методы в классе компонента не возвращают результата.
3) Все методы интерфейса должны соответствовать их объявлению в классе компонента и содержать в предложении throws исключение java.rmi.RemoteException.

Требования к классу компонента.

1) Класс должен быть объявлен как public и не может быть final или abstract.
2) Класс компонента должен предоставлять публичный конструктор без параметров. Именно он используется контейнером для создания экземпляра класса компонента.
3) Тип компонента (с состоянием или без) можно указать с помощью аннотации к классу компонента: Stateful – для компонентов с состоянием и Stateless – для компонентов без состояния.
4) Класс компонента должен реализовать бизнес-интерфейс(ы) компонента или его (их) методы и бизнес-методы, объявленные в удаленном или локальном интерфейсе компонента, если указанные интерфейсы определены. Если класс компонента не реализует явно бизнес-интерфейс или компонентные интерфейсы, то для определения клиентского представления компонента можно использовать следующие аннотации: для локального представления – Local, для удаленного представления – Remote, для адаптированного локального представления – LocalHome, для адаптированного удаленного представления – RemoteHome.

   К бизнес-методам предъявляются следующие требования:
   a. название метода может быть произвольным, но не должно начинаться с «ejb»;
   b. метод должен быть public и не может быть final или static;
   c. параметры и возвращаемое значение должны быть сериализуемыми, если метод соответствует бизнес-методу удаленного бизнес-интерфейса или удаленного компонентного интерфейса;
   d. метод может выбрасывать произвольные исключения, определенные в приложении (не системные).
5) Класс компонента может прямо или косвенно реализовать интерфейс javax.ejb.SessionBean.
6) Класс компонента может содержать ejbCreate-метод(ы), к которому предъявляются следующие требования:

   a. метод ejbCreate должен быть public и не может быть final или static;
   b. тип возвращаемого значения должен быть void;
   c. предложение throws может включать исключение javax.ejb.CreateException;
   d. параметры метода должны соответствовать параметрам одного из create-методов, объявленных в домашнем интерфейсе.
7) Класс компонента может иметь родительский класс, который в свою очередь не может быть классом сессионного компонента.
8) Кроме этого класс компонента может содержать любые вспомогательные методы.