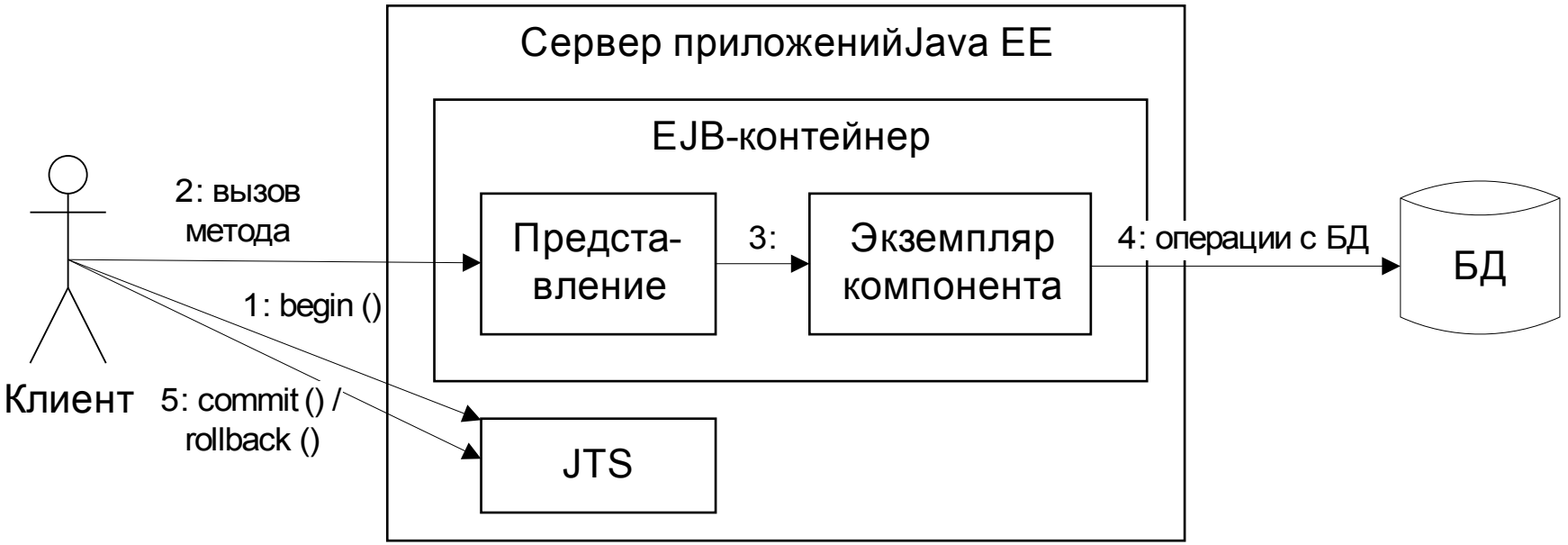
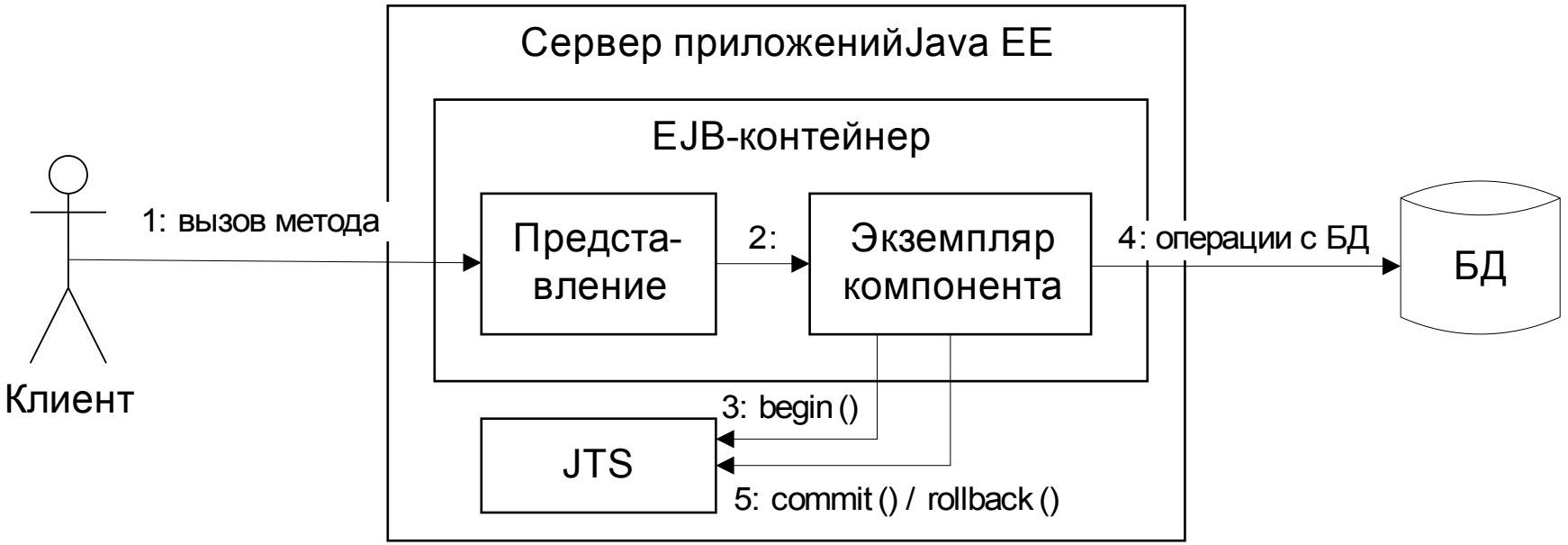
Виды транзакций на платформе Java EE

Транзакции, инициируемые клиентом



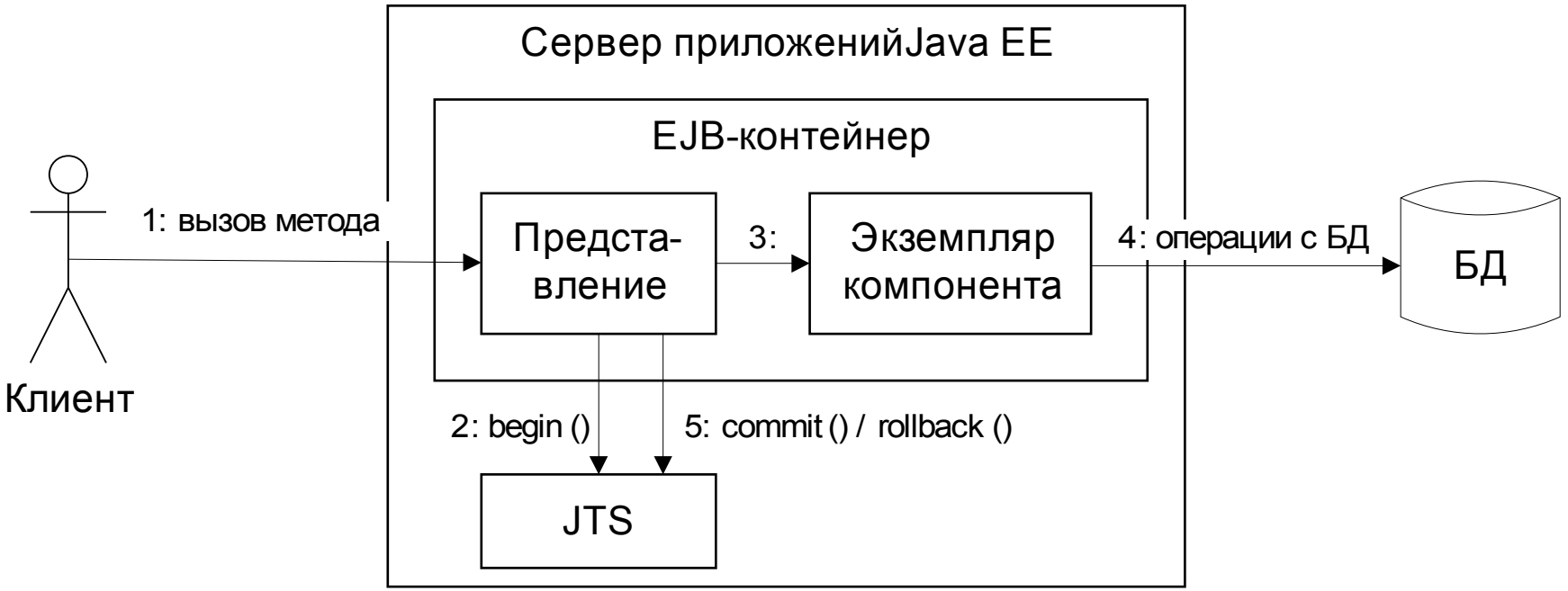
Транзакции, управляемые программно

- **bean** managed transactions



Транзакции, управляемые декларативно

- **container** managed transactions



Выбор вида управления транзакциями для SB

- XML-дескриптор
- Аннотация `@TransactionManagement`
  - **`BEAN`**
  - **`CONTAINER`** (по умолчанию)
- Пример:

```
@Stateless
@TransactionManagement(BEAN)
public class MySessionBean implements MySession {
   ...
}
```

Декларативное управление транзакциями

Транзакционный атрибут

- При декларативном управлении транзакциями разработчик не может явно начать и завершить транзакцию
- Управление осуществляется с помощью транзакционного атрибута
  - Для каждого бизнес-метода компонента
  - Устанавливается аннотацией `@TransactionAttribute` либо через XML-дескриптор
  - Существует 6 значений (по умолчанию — `REQUIRED`), которые определяют:
    - выполняется ли вызванный бизнес-метод в контексте какой-либо транзакции или нет
    - начинается ли транзакция автоматически при вызове этого метода

Поведение контейнера при вызова бизнес-метода компонента

Значение транзакционного атрибута	Клиентская транзакция	Транзакция, связанная с методом компонента
NOT_SUPPORTED	Нет	нет
	T1	нет
REQUIRED	Нет	T1
	T1	T1
SUPPORTS	Нет	нет
	T1	T1
REQUIRES_NEW	Нет	T1
	T1	T2
MANDATORY	Нет	ошибка
	T1	T1
NEVER	Нет	нет
	T1	ошибка

Возможное применение значений транзакционного атрибута

- **REQUIRES_NEW** – изменения в источнике данных должны быть произведены независимо от других действий с источниками данных в рамках цепочки вызовов
  - Запись в журнал действий пользователя или аналогичных операций.
- **NOT_SUPPORTED** – аналогично **REQUIRES_NEW**, но при этом контейнер не обеспечивает выполнение транзакционных свойств для этих изменений
  - Если данные только добавляются, то конфликты при подобных действиях с БД не возникают, и нет накладных расходов на организацию транзакции

Возможное применение значений транзакционного атрибута

- **SUPPORTS** – используется в первую очередь для методов, не выполняющих операции с источником данных
  - Например, set-методы для полей сессионного компонента с состоянием
  - Отсутствуют накладные расходы, вызываемые как приостановлением транзакции в режиме **NOT_SUPPORTED**, так и ее началом в режиме **REQUIRED**
- **NOT_SUPPORTED**, **SUPPORTS** и **NEVER** используются также для работы с источниками данных, не поддерживающими транзакции.

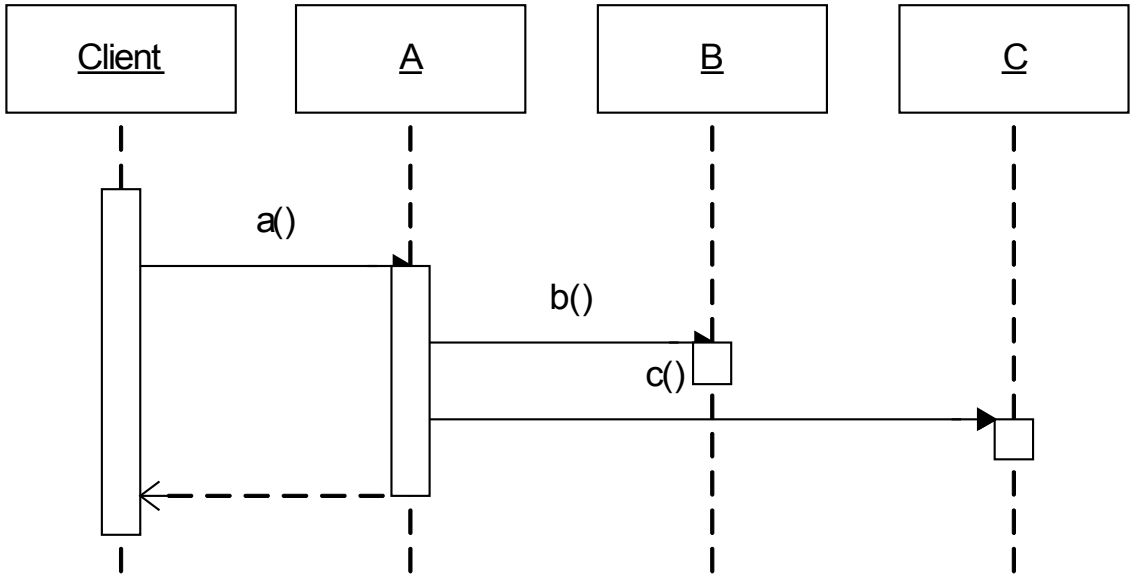
Демаркация транзакций, управляемых декларативно

- Границы транзакции совпадают с границами бизнес-метода
- Если транзакция была автоматически начата при вызове метода, то при успешном возврате из метода она будет зафиксирована
- Откат транзакции:
  - Автоматический, при возникновении исключений
  - Программный, с помощью метода `EJBContext.setRollbackOnly()`

Ограничения для транзакций, управляемых декларативно

- Запрещается использовать любые API управления транзакциями, специфичные для менеджера ресурсов
- Если в качестве менеджера ресурсов используется источник данных JDBC, то запрещено использование методов `commit()`, `rollback()`, `setAutoCommit()` интерфейса `java.sql.Connection`.

Пример декларативного управления транзакциями



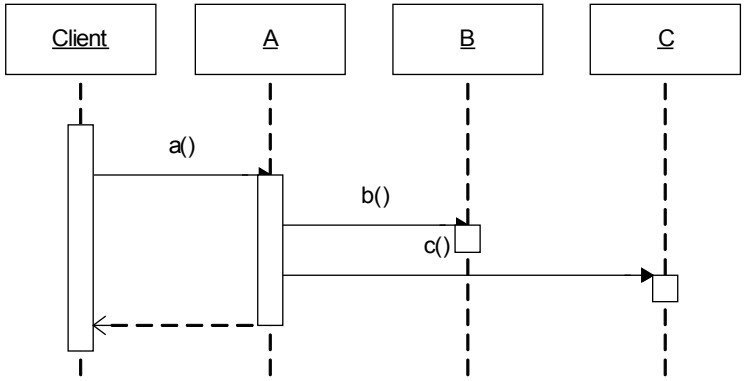
Client TX	A		B		C	
	TX Attr	a()	TX Attr	b()	TX Attr	c()
T1	SUPPORTS	–	NOT_SUPPORTED	–	REQUIRES_NEW	setRollbackOnly()

	Client	A.a()	B.b()	C.c()
Контекст транзакции	T1	T1	–	T2

Поведение контейнера при вызова бизнес-метода компонента

Значение транзакционного атрибута	Клиентская транзакция	Транзакция, связанная с методом компонента
NOT_SUPPORTED	Нет	нет
	T1	нет
REQUIRED	Нет	T1
	T1	T1
SUPPORTS	Нет	нет
	T1	T1
REQUIRES_NEW	Нет	T1
	T1	T2
MANDATORY	Нет	ошибка
	T1	T1
NEVER	Нет	нет
	T1	ошибка

Пример декларативного управления транзакциями (прод.)

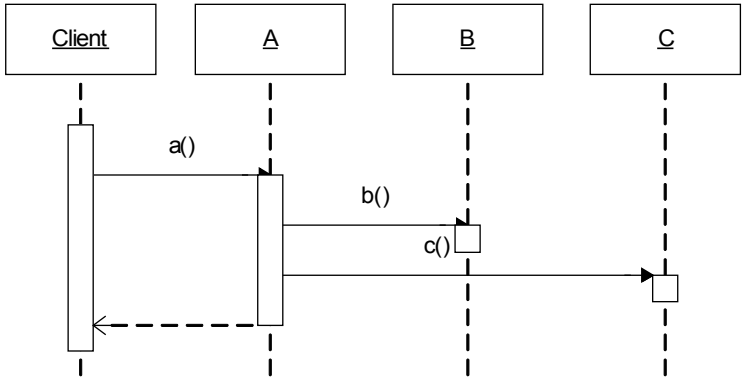


Client TX	A		B		C	
	TX Attr	a()	TX Attr	b()	TX Attr	c()
T1	SUPPORTS	–	NOT_SUPPORTED	–	REQUIRES_NEW	setRollbackOnly()

	Client	A.a()	B.b()	C.c()
Контекст транзакции	T1	T1	–	T2
Исход транзакции	?	Зависит от клиента	–	Откат

Неизвестно, как закончится вызывающий метод на клиенте

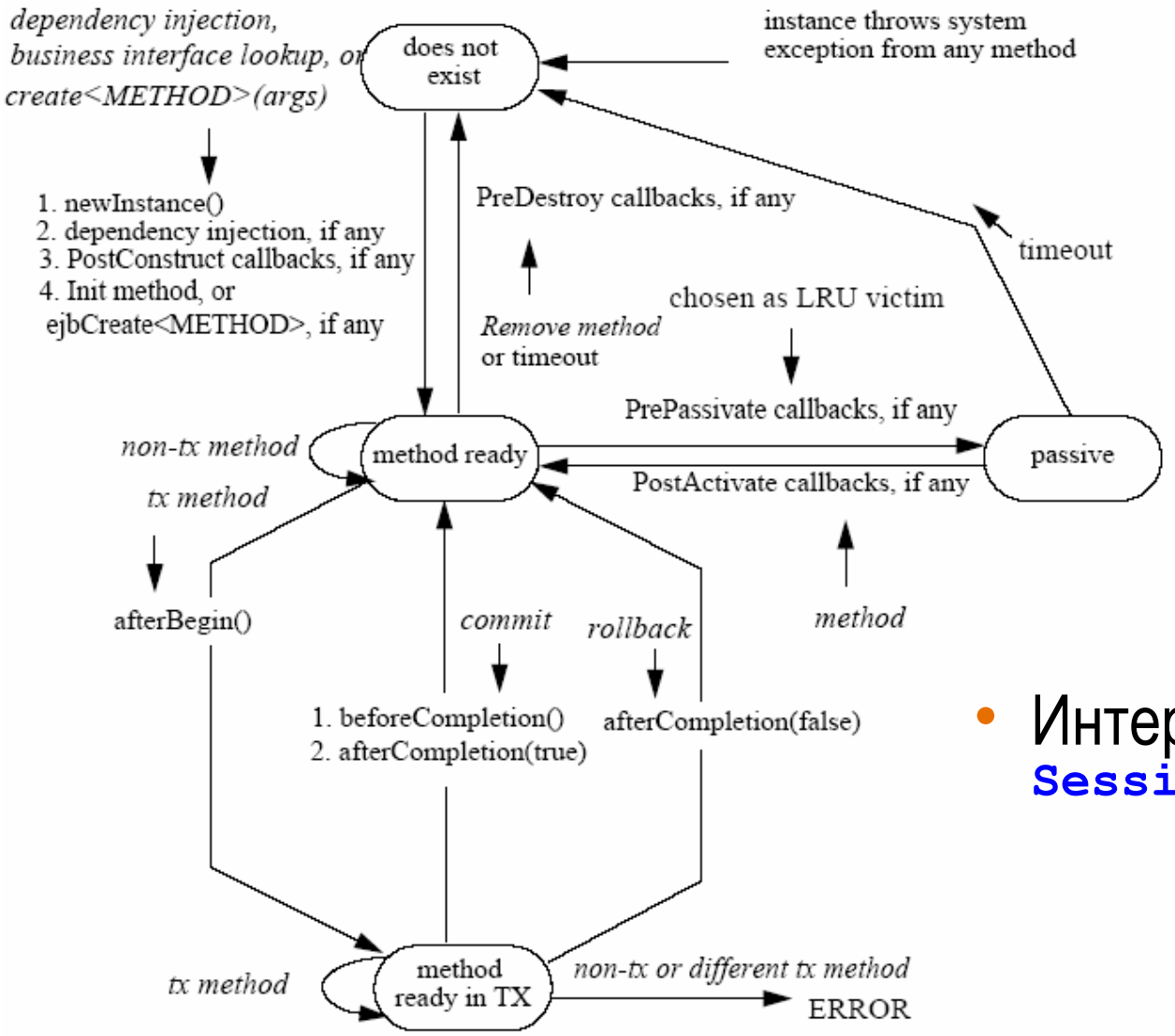
Пример декларативного управления транзакциями (прод.)



Client TX	A		B		C	
	TX Attr	a()	TX Attr	b()	TX Attr	c()
T1	SUPPORTS	–	NOT_SUPPORTED	–	REQUIRES_NEW	setRollbackOnly()

	Client	A.a()	B.b()	C.c()
Контекст транзакции	T1	T1	–	T2
Исход транзакции	?	Зависит от клиента	–	Откат
Изменения в БД	?	Зависит от клиента	Сохранение	Откат

Синхронизация с транзакциями в SFSB



- Интерфейс `SessionSynchronization`

Программное управление транзакциями

Программное управление транзакциями

- Для управления транзакциями используется интерфейс **`javax.transaction.UserTransaction`**
  - `@Resource javax.transaction.UserTransaction ut;`
  - `SessionContext.getUserTransaction()`
  - `new InitialContext()`
    `.lookup("java:comp/UserTransaction")`
- Использование методов управления транзакциями, специфичных для менеджеров ресурсов, **запрещено**
  - например, запрещены методы `commit()` и `rollback()` интерфейса **`java.sql.Connection`** для источников данных JDBC

Методы UserTransaction

- begin() - начало новой транзакции
- commit() - подтверждение транзакции
- rollback() - откат транзакции
- setRollbackOnly() - пометка к откату транзакции, связанной с потоком выполнения
- setTransactionTimeout() - установка интервала времени (в секундах), по истечении которого транзакция автоматически завершается
- getStatus() - получение состояния транзакции, связанной с данным потоком выполнения

Состояние глобальной транзакции

- `STATUS_ACTIVE` – транзакция была начата, не помечена к откату и выполняется;
- `STATUS_MARKED_ROLLBACK` – транзакция была начата, была помечена для отката, но все еще выполняется;
- `STATUS_NO_TRANSACTION` – с данным потоком выполнения не связано никакой транзакции, т.е. транзакция никогда не начиналась, либо успешно завершена подтверждением или откатом, в ходе которых не было обнаружено эвристических решений;
- `STATUS_UNKNOWN` – означает, что состояние транзакции не может быть определено, потому что оно в данный момент изменяется (переходное состояние);
- `STATUS_PREPARING` – транзакция, связанная с потоком выполнения, подготавливается к подтверждению (то есть выполняется фаза подготовки протокола 2PC);
- `STATUS_PREPARED` – фаза подготовки протокола 2PC завершена и ожидается начало выполнения второй фазы;
- `STATUS_COMMITTING` – выполняется фаза подтверждения протокола 2PC;
- `STATUS_ROLLING_BACK` – выполняется откат транзакции, связанной с потоком выполнения;
- `STATUS_COMMITTED` – транзакция завершилась подтверждением, однако в процессе подтверждения было обнаружено, что часть интерфейсов данных приняли эвристическое решение;
- `STATUS_ROLLEDBACK` – транзакция завершилась откатом, в процессе выполнения отката было обнаружено, что часть источников данных приняли эвристическое решение.

Вызов бизнес-метода компонента

№ п/п	Транзакция клиента	Транзакции, связанная с экземпляром компонента	Транзакция, связанная с вызванным методом
1	Нет	Нет	Нет
2	T1	Нет	Нет
3	Нет	T2	T2
4	T1	T2	T2

} Только SFSB

- Бизнес-метод **SFSB**, начавший транзакцию, может завершиться без подтверждения или отката транзакции
  - > Контейнер устанавливает связь между экземпляром компонента (а точнее потоком, в котором выполняются его методы) и транзакцией
  - > Последующие вызовы бизнес-методов этого экземпляра компонента выполняются в контексте той же транзакции
  - > Связь между экземпляром компонента и транзакцией поддерживается до завершения транзакции

Пример программного управления транзакциями

```
@Stateful
@TransactionManagement(BEAN)
public class MySessionBean implements MySession {
    @Resource javax.transaction.UserTransaction ut;
    @Resource javax.sql.DataSource database1;

    public void method1(...) {
        ut.begin(); // начало транзакции
    }
    public void method2(...) {
        java.sql.Connection con = database1.getConnection();
        java.sql.Statement stmt = con.createStatement();
        stmt.executeUpdate(...);
        con.close();
    }

    public void method3(...) {
        ut.commit(); // подтверждение транзакции
    }
}
```

Принципы обработки ошибок в EJB-компонентах

Группы исключений

- Прикладные (Application Exception)
  - Сообщают клиенту о проблемах, специфических для предметной области (например, некорректное значение входного параметра)
  - Клиент может преодолеть эти проблемы
- Системные (System Exceptions)
  - Ошибки, преодолеть которые невозможно
    - Невозможность получить соединение с БД или найти имя в JNDI-контексте
    - Исключение `RemoteException` при удаленном вызове
    - Ошибки при работе виртуальной машины JVM (потомки `java.lang.Error`)
    - ...

Прикладные исключения

- Прикладное исключение передается клиенту точно в том виде, в каком оно выбрасывается методом компонента
- Классы прикладных исключений определяются разработчиком
  - Класс исключения не должен расширять `java.rmi.RemoteException`
- Прикладные исключения могут быть как контролируемыми, так и неконтролируемыми
- Класс исключения может быть помечен аннотацией `@ApplicationException`
  - Для неконтролируемых исключений — должен

Прикладные исключения

- При возникновении прикладного исключения контейнер не выполняет автоматический откат транзакции, позволяя клиенту преодолеть ошибочную ситуацию
- Иногда для обеспечения целостности данных следует помечать транзакцию для отката при выбросе прикладного исключения
  - Явно - вызовом метода `EJBContext.setRollbackOnly()` перед выбросом прикладного исключения
  - Неявно — на уровне класса исключения

```
@ApplicationException(rollback=true)
public class RollingBackAppException { ... }
```

Системные исключения

- Неконтролируемые исключения
  - Не помеченные как прикладные
- Исключения, расширяющие `java.rmi.RemoteException`

Системные исключения

- При возникновении системного исключения компонент должен выбросить неприкладное исключение:
  - если при исполнении метода возникло неконтролируемое исключение, то необходимо просто передать его контейнеру
  - если при выполнении операции метода возникло контролируемое исключение, которое не может быть преодолено, следует выбросить `javax.ejb.EJBException`, которое содержит в себе оригинальный объект исключения;
  - прочие ошибочные ситуации должны завершаться выбросом `javax.ejb.EJBException`

Системные исключения

- Когда контейнер отлавливает системные исключения, выброшенные методами компонентов, он:
  - сохраняет информацию о нем в журнале;
  - выбрасывает `javax.ejb.EJBException` или `java.rmi.RemoteException`
    - Если компонент используется в контексте клиентской транзакции, то выбрасывается `javax.transaction.TransactionRollbackException` или `javax.transaction.TransactionRollbackLocalException`
- При этом контейнер гарантирует следующее:
  - транзакция, в которой участвовал метод, выбросивший системное исключение, заканчивается откатом;
  - экземпляр компонента, выбросивший исключение, будет удален