**Жизненный цикл сущности в Hibernate**

**Java** объекты — сущности(классы с аннотацией **@Entity**) проходят через несколько состояний когда мы используем **Hibernate**:

**1.** Transient

**2.** Persistent (Managed)

**3.** Detached

**4.** Removed

**Transient**   
Когда мы создаем наш объект, наш объект попадает в **Transient** состояние, тоесть этот объект пока никак не связан с **Hibernate** и **Hibernate** этот объект не отслеживает.



* Объект пока никак не связан с **Hibernate** и не отслеживается (не находится в так называекмом Persistance context)
* Вызов сеттеров на объекте никак не будет влиять на состояние БД.
* Можно воспринимать объект, находящийся в **Transient** состоянии как обычный **Java** объект.

**Persistent(Managed)**

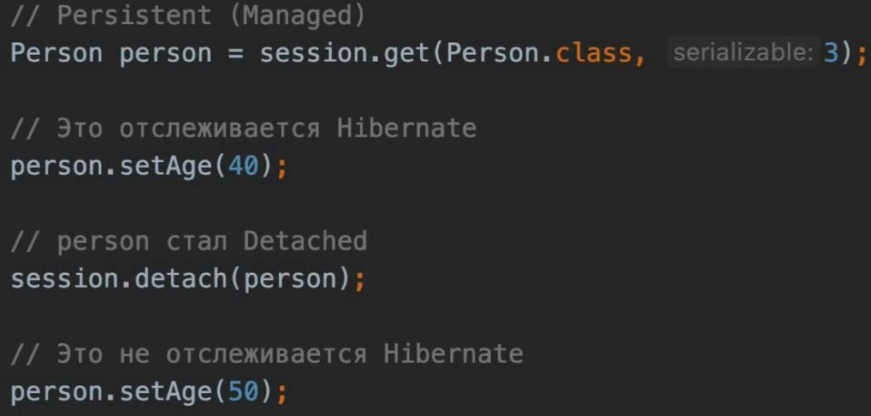
Если мы наш объект сохраним с помощью метода ***save()***, то он перейдет в другое состояние которое называется **Persistent.**



* Java объект становится **Persistent** тогда, когда сохраняется с помощью **Hibernate** (следующий вызов ***commit()*** сохранит объект в БД).
* Еще один слцучай когда Java объект является **Persistent** – когода мы получаем его из БД с помощью **Hibernate.**
* Объект находится в **Persistence context** и отслеживается **Hibernate`**ом. **Persistent context** это такое специальное пространство, где находятся объекты которые отслеживаются **Hibernate`**ом.
* Объект находится в этом состоянии до тех пор, пока работает **Hibernate** сессия или пока мы сами не выкинем его из этого состояния.
* Если на объекте который находится в состоянии **Persistent** мы вызовем сеттер, то **Hibernate** так как он отслеживает эти объекты сгенерирует **sql** запрос.
* Способов получить объект с таким статусом ровно два:  
  Загрузить объект из **Hibernate**.  
  Сохранить объект в **Hibernate**.

**Detached**

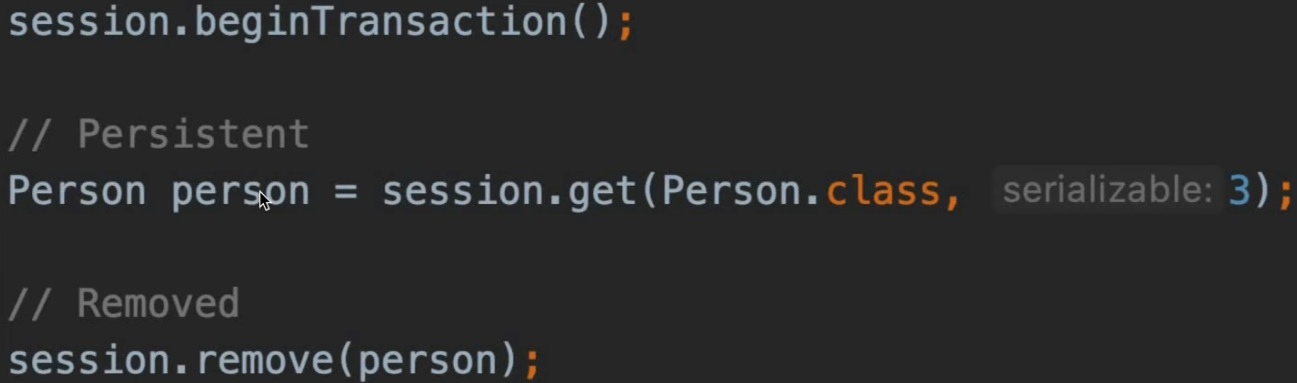
Это состояние характеризуется тем, что объект перестёт быть в состоянии **Persistent.**

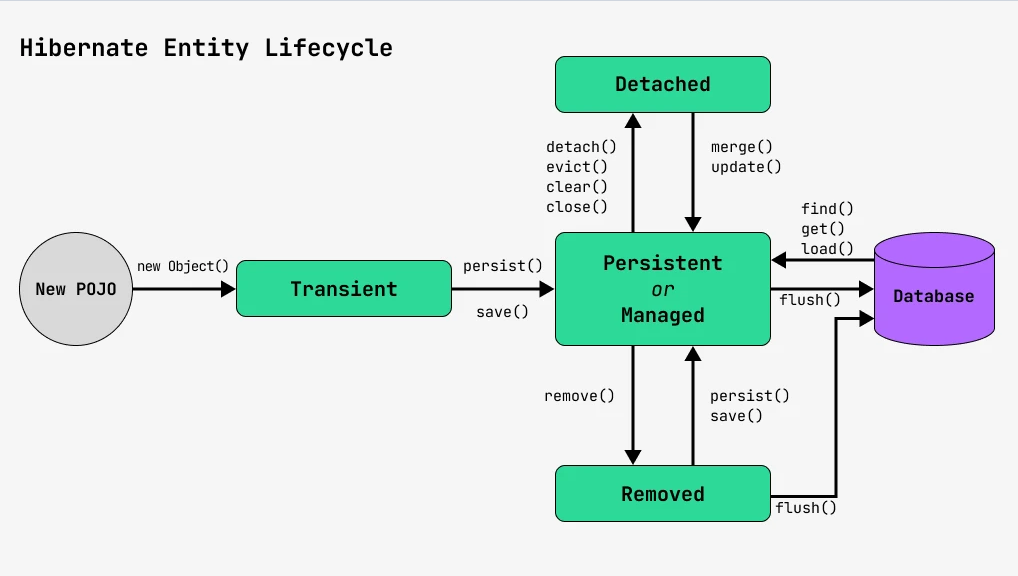


* Объект перестаёт быть **Persistent** и покидает **Persistence context –** становится опять обычным Java объектом, который **Hibernate** не отслеживает.
* Похоже на **Transient** состояние.
* Это состояние достигается с помощью вызова метода ***detach()*** или когда закрывается **Hibernate** сессия.
* Можно присоединить объект обратно к **Persistence context** если вызвать метод ***merge()***. После этого объект опять станет отслеживаемым.

**Removed**

Это состояние возникает тогда когда мы удаляем наш объект и когда на сессии вызываетcя метод ***remove()***, после этого объект помечается для удаления и следующий ***commit()*** в этой транзакции удалит соответствующую строку из таблицы. Наш объект будет помечен как **removed** и этот объект будет вскоре удален из **Hibernate** кэша.





**Нюансы работы метода persist()**

При сохранении объекта в базу данных тебе нужно помнить две вещи.

Во-первых, сохранение объекта в итоге приведет к выполнению одной из команд SQL: **INSERT** или **UPDATE**.

Во-вторых, эти действия произойдут не сразу после вызова метода объекта session, а только после закрытия транзакции.

Задача этого метода сохранить в базу данных новый объект. Если такого объекта еще нет, то он будет добавлен в базу через вызов SQL-метода **INSERT**.

Если же объект уже есть в базе, то ничего не произойдет. Однако возможен еще и третий случай – попытка сохранить в базу объект со статусом **Detached**. В этом случае кинется исключение. 

**Нюансы работы метода save()**

Метод ***save()*** достался нынешнему **Hibernate** от его предыдущих версий. По своей сути он очень похож на метод ***persist()***, он тоже добавляет новую запись в таблицу с помощью метода **INSERT**. Однако у него есть несколько интересных нюансов.

Во-первых, этот метод **возвращает значение – новое ID объекта**. Как ты уже знаешь, обычно до добавления в базу у объектов нет ID и оно присваивается уже базой данных. Так вот, метод ***save()*** объекта session возвращает ID, которая была присвоена сохраняемому объекту.

**Важно!** Согласно спецификации **Hibernate** ID может быть любой сериализуемый объект, не только число. Это может быть строка, число, enum, вообще что угодно, что можно целиком положить в одну колонку таблицы в базе.

Также у метода ***save()***другое поведение в случае с **Detached** состоянием объекта. Такой объект он рассматривает как новый и просто добавляет еще одну запись.

**Нюансы работы метода merge()**

Если ты хочешь с помощью **Hibernate** изменить объект, который уже был сохранен в базе, то для этого тоже есть несколько методов.

Во-первых, это метод ***merge()***, который обновляет информацию в базе на основе переданного объекта. При этом будет вызван SQL-запрос **UPDATE.**

Тут есть несколько важных нюансов:

Во-первых, метод ***merge()*** возвращает результат – обновленный объект. Этот объект имеет состояние **Persist** и присоединен к объекту session. Объект передаваемый в метод ***merge()***при этом не меняется.

В метод ***merge()*** можно передать POJO объект, а в качестве результата метод может вернуть proxy (зависит от настроек Hibernate). Поэтому просто запомни, что метод ***merge()*** не меняет передаваемый объект.

Во-вторых, если объект передаваемый в ***merge()***имеет статус Transient (и у него нет ID), то для него создастся отдельная строчка в базе данных. Другими словами будет выполнена команда ***persist()***.

В-третьих, если в метод ***merge()*** передать объект уже присоединенный к сессии (со статусом Persist), то ничего не произойдет – метод просто вернет этот же объект. Почему? А все потому, что при коммите транзакции данные и так запишутся в базу.

Не нужно сохранять каждый раз объект после любых его изменений. Если этот объект в статусе Persist, то Hibernate все сделает сам. Если вы меняете объект, который “присоединен к базе”, то все его изменения будут записаны в базу.

**Нюансы работы метода update()**

Также у **Hibernate** есть метод ***update()***, который как и метод ***save()***достался ему от предыдущих версий. С помощью этого метода можно только обновить данные уже сохраненного объекта. При этом будет вызван SQL-запрос UPDATE.

Этот метод ничего не возвращает и не меняет существующий объект.

Если вызвать этот метод для нового объекта, то просто кинется исключение.

**Метод saveOrUpdate()**

До появления JPA функцию метода ***persist()***выполнял метод ***saveOrUpdate().*** Его задачей было обновить в базе информацию по существующему объекту, а если такового нет, то создать его. Его почти всегда используют вместо методов ***save()*** и ***update().***

В отличии от метода ***update()***, он может менять передаваемый ему объект. Например, установить ему **ID**, который был присвоен при сохранении в базу данных.

Как это работает:

* если у передаваемого объекта установлен ID, то вызывается SQL-метод UPDATE.
* если у передаваемого объекта ID не установлен, то вызывается SQL-метод INSERT.

**Разные способы получения объекта**

Если ты хочешь получить объект по его **ID** (или Primary Key), то для этого у **Hibernate** есть аж целых три метода:

* load()
* get()
* find()

Делают они одно и то же, однако есть нюансы.

**Метод get()**

Метод ***get()*** в качестве второго параметра принимает **ID** (Primary Key) объекта, который ему нужно вернуть. Затем загружает этот объект из базы и возвращает его.

Если запись с таким ID не будет найдена в базе, то метод вернет null.

**Метод load()**

Поведение отличается от метода ***get()***.

Во-первых, этот метод возвращает не реальный объект, а **proxy**: виртуальную заглушку.

Во-вторых, при использовании метода ***load()***не происходит проверка, есть ли такая запись в базе. Вместо этого **Hibernate** сразу создает **proxy**-объект с переданным **ID** и возвращает его.

В-третьих, вся работа с базой данных будет происходить при вызове методов **proxy**-объекта. Если ты попытаешься вызвать, например, метод ***getName()***, тогда и произойдет первое обращение к базе.

Метод ***load()***не стоит использовать для проверки наличия объектов в базе – он этого просто не покажет. Кроме того, если в него передать невалидный **ID**, например null, он просто вернет null.

**Метод find()**

Метод ***find()***достался интерфейсу Session от стандарта JPA. А как ты знаешь, этот стандарт описывает не просто сигнатуру методов, но и регламентирует поведение.

Этот метод работает точно так же, как и метод ***get()***. Если объект по переданному ключу не был найден, то метод просто вернет null.

**Метод refresh()**

Еще один полезный метод, который имеет отношение к загрузке объекта из базы – это метод ***refresh()***.

Помнишь метод ***persist()***, который обновлял данные в базе на основе переданного объекта? Так вот, метод ***refresh()*** работает с точностью да наоборот: он обновляет существующий объект на основе данных из базы.

Такое поведение бывает нужно, если, например, при записи объекта в базу там вызываются различные хранимые процедуры, которые корректируют записанные данные.

В таких случаях бывает полезно перечитать объект из базы, если есть вероятность, что он изменился.

## **Удаление с помощью метода remove()**

Удалять объекты из базы очень просто, но как говориться, есть нюансы. И таких нюансов аж шесть:

* Удаление с помощью метода remove()
* Удаление за компанию
* Удаление по Orphan
* Удаление с помощью JPQL
* Удаление через NativeQuery
* softDeleted()