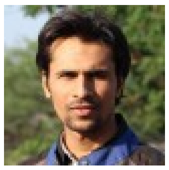
JAVA EE, JPA 2

# JPA Lifecycle callback methods

by Prasad Kharkar • May 23, 2014 • 1 Comment

Bio | Latest Posts



**Prasad Kharkar**

Prasad Kharkar is a java enthusiast and always keen to explore and learn java technologies. He is SCJP,OCPWCD, OCEJPAD and aspires to be java architect.

As of now we have learned about entities, relationships, inheritance and how entity manager performs operations on them. Till now, entities played a passive part. What if we want entities to perform something when some operations are performed on them? How can we listen to the events performed on entities? We can write JPA lifecycle callback methods by making use of annotations.

## JPA Lifecycle Callback methods:

Here are JPA lifecycle callback methods:

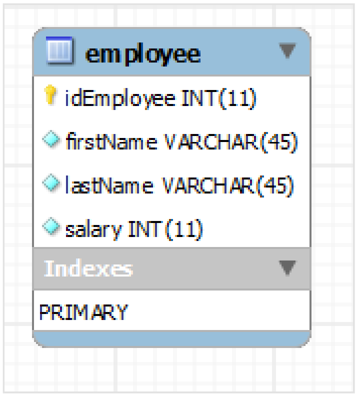
- @PrePersist :- Entity is notified when em.persist() is successfully invoked on entity.
- @PostPersist :- Notified when entity is persted in database.
- @PreUpdate :- Notified when entity is modified.
- @PostUpdate :- Notified when updated state of entity is inserted in database
- @PreRemove :- Notified when remove operation is called for entity by entity manage.r
- @PostRemove:- Notified when entity is deleted from database.

Some rules need to be followed when annotating a method as jpa lifecycle callback methods.

- They can have any name
- The method signature should be such that return type is **void** and there are no parameters.
- They can have any access i.e. they can be **private**, **default**, **protected** or **public**.
- They cannot throw any checked exceptions.

We've seen a little description about jpa lifecycle callback methods, let us directly test them out.

We have employee table as below.



Corresponding Employee entity

```java
package com.thejavageek.jpa.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.PostLoad;
import javax.persistence.PostPersist;
import javax.persistence.PostRemove;
import javax.persistence.PostUpdate;
import javax.persistence.PrePersist;
import javax.persistence.PreRemove;
import javax.persistence.PreUpdate;
import javax.persistence.TableGenerator;

@Entity
public class Employee {

    @TableGenerator(name = "employee_gen", pkColumnName = "gen_name", valueColumnName = "gen_
    @Id
    @GeneratedValue(generator = "employee_gen", strategy = GenerationType.TABLE)
    private int idEmployee;

    private String firstName;
    private String lastName;
    private int salary;

    public int getIdEmployee() {
        return idEmployee;
    }

    public void setIdEmployee(int idEmployee) {
        this.idEmployee = idEmployee;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public int getSalary() {
        return salary;
    }

    public void setSalary(int salary) {
        this.salary = salary;
    }

    @PrePersist
    public void methodInvokedBeforePersist() {
        System.out.println("Invoked before persisting employee");
    }

    @PostPersist
    public void methodInvokedAfterPersist() {
        System.out.println("Invoked after persisting employee");
    }

    @PreUpdate
    public void methodInvokedBeforeUpdate() {
        System.out.println("Invoked before updating employee");
    }

    @PostUpdate
    public void methodInvokedAfterUpdate() {
        System.out.println("Invoked after updating employee");
    }

    @PreRemove
    public void methodInvokedBeforeRemove() {
        System.out.println("Invoked before removing employee");
    }

    @PostRemove
    public void methodInvokedAfterRemove() {
        System.out.println("Invoked after removing employee");
    }

}
```

Run this using code below.

```java
Employee employee = new Employee();
employee.setFirstName("prasad");
employee.setLastName("kharkar");
employee.setSalary(100000);

em.persist(employee);
int idEmployee = employee.getIdEmployee();
```

```java
transaction.commit();

transaction.begin();

employee = (Employee) em.find(Employee.class, idEmployee);
employee.setSalary(employee.getSalary() + 10000);
transaction.commit();

transaction.begin();
em.persist(employee);
em.remove(employee);
transaction.commit();
```

This gives output as follows.

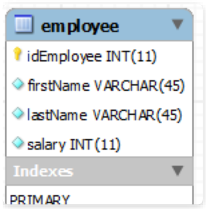
```
Invoked before persisting employee
Invoked after persisting employee
Invoked before updating employee
Invoked after updating employee
Invoked before removing employee
Invoked after removing employee
```

I hope this example helped understand jpa lifecycle callback methods, in next article, we will see how we can create entity listeners such that we do not need to modify entities

## References for further study :

- Apress: Mastering Java Persistence API 2.o book.
- JPA Tutorial
- JPA javadoc

### Related Posts

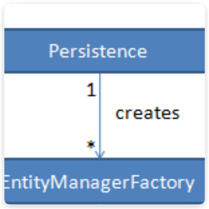

employee
idEmployee INT(11)
firstName VARCHAR(45)
lastName VARCHAR(45)
salary INT(11)
Indexes
PRIMARY

JPA EntityListeners

Entities in JPA

Application Managed EntityManager

JPA EntityManager Operations



jpa architecture

Tags: entities javaee jpa2

← JPA table per concrete class example

JPA EntityListeners →

## 1 comment for “JPA Lifecycle callback methods”

Pingback: JPA EntityListeners | theJavaGeek

## Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

Bank of America

You have tried to visit a site that has been blocked for your security

**<http://tpc.googlesyndi
0-
16/html/container.html
n=1>**

This website is categorized as: **Web Ads/Analytics**

If you have a critical business need to access this website, please request an exception via

Search …