Entity Lifecycle, Callbacks, and Listeners

Antonio Goncalves
JAVA CHAMPION

@agoncal www.antoniogoncalves.org

Previous Module

Java Persistent Query Language

Manipulate entities and attributes

Dynamic queries

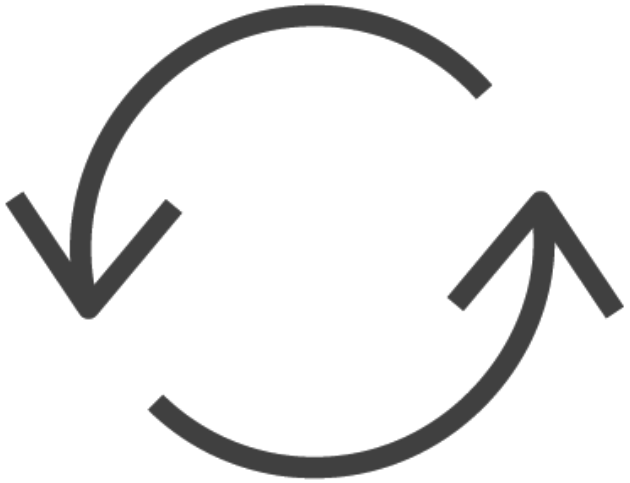
Named queries

Bind parameters or paginate

Overview

- Lifecycle
- Managed
- Detached
- Callback annotations
- Business logic
- Entity listeners

Entity Lifecycle



Entities are POJOs

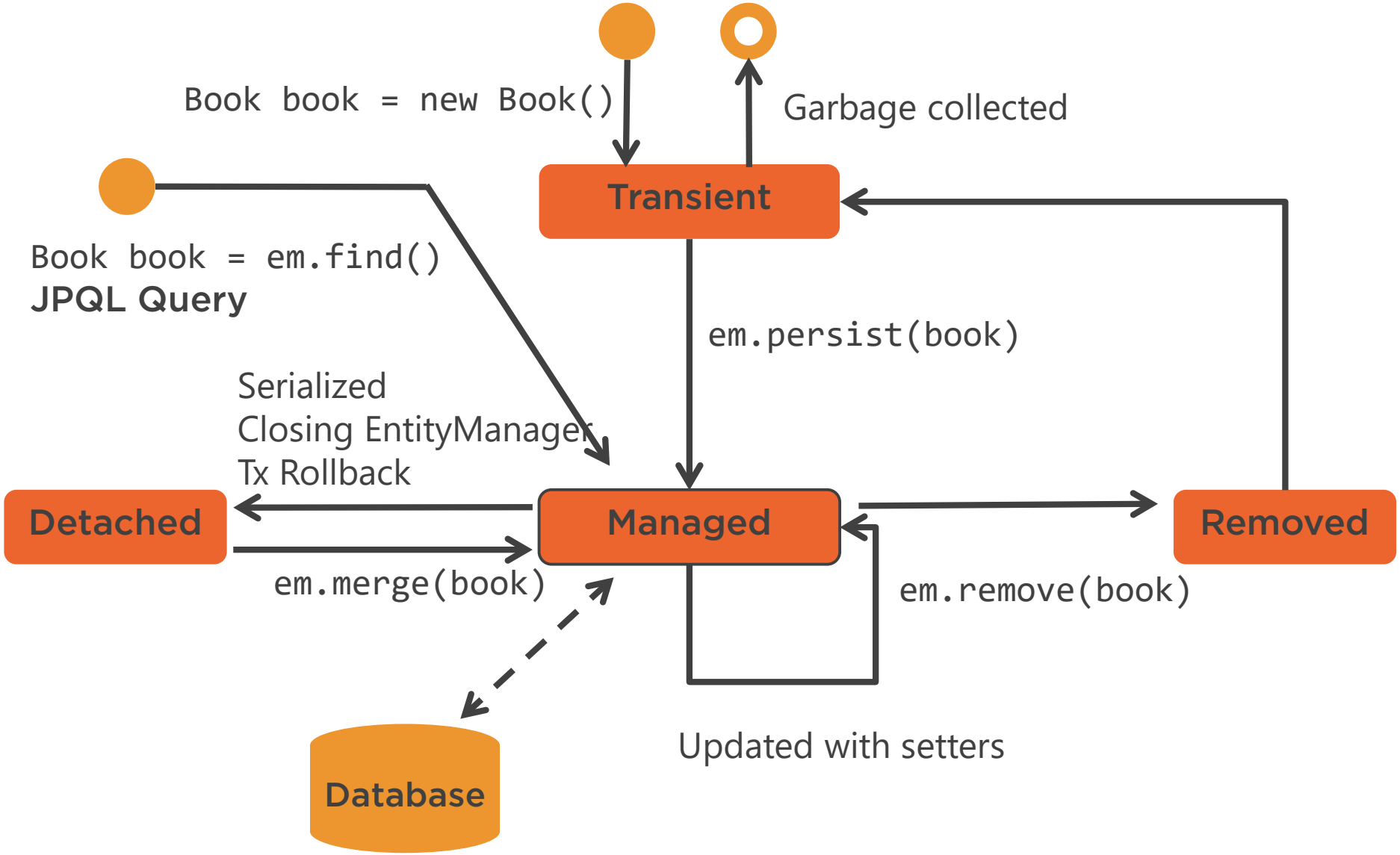
Events occur depending on operation

- Persisting
- Updating
- Removing
- Loading

Managed

Detached

Entity Lifecycle



Managed vs. Detached: Persisting

```
public class BookService {

  private EntityManagerFactory emf =
          Persistence.createEntityManagerFactory("myPU");
  private EntityManager em = emf.createEntityManager();
  private EntityTransaction tx = em.getTransaction();

  public Book createBook(Long id, String title, String desc) {
    Book book = new Book();
    book.setId(id);
    book.setTitle(title);
    book.setDescription(desc);
    tx.begin();
    em.persist(book);
    tx.commit();
    return book;
  }
```

Managed vs. Detached: Persisting

```
public class BookService {

  private EntityManagerFactory emf =
          Persistence.createEntityManagerFactory("myPU");
  private EntityManager em = emf.createEntityManager();
  private EntityTransaction tx = em.getTransaction();

  public Book createBook(Book book) {
    tx.begin();
    em.persist(book);
    tx.commit();
    return book;
  }



}
```

Managed vs. Detached: Finding

```
public class BookService {

  private EntityManagerFactory emf =
          Persistence.createEntityManagerFactory("myPU");
  private EntityManager em = emf.createEntityManager();
  private EntityTransaction tx = em.getTransaction();

  public Book findBook(Long id) {
    return em.find(Book.class, id);
  }




}
```

Managed vs. Detached: Updating

```
public class BookService {

  private EntityManagerFactory emf =
          Persistence.createEntityManagerFactory("myPU");
  private EntityManager em = emf.createEntityManager();
  private EntityTransaction tx = em.getTransaction();

  public Book raiseUnitCost(Long id, Float raise) {
    Book book = em.find(Book.class, id);
    if (book != null) {
      tx.begin();
      book.setUnitCost(book.getUnitCost() + raise);
      tx.commit();
    }
    return book;
  }
}
```

Managed vs. Detached: Updating

```
public class BookService {

  private EntityManagerFactory emf =
          Persistence.createEntityManagerFactory("myPU");
  private EntityManager em = emf.createEntityManager();
  private EntityTransaction tx = em.getTransaction();

  public Book raiseUnitCost(Book book, Float raise) {
    Book bookToBeUpdated = em.merge(book);
    tx.begin();
    bookToBeUpdated.setUnitCost(bookToBeUpdated.getUnitCost() + raise);
    tx.commit();
    return book;
  }


}
```

Managed vs. Detached: Removing

```
public class BookService {

  private EntityManagerFactory emf =
          Persistence.createEntityManagerFactory("myPU");
  private EntityManager em = emf.createEntityManager();
  private EntityTransaction tx = em.getTransaction();

  public void removeBook(Book book) {
    Book bookToBeDeleted = em.merge(book);
    tx.begin();
    em.remove(bookToBeDeleted);
    tx.commit();
  }



}
```

Managed vs. Detached: Removing

```
public class BookService {

  private EntityManagerFactory emf =
          Persistence.createEntityManagerFactory("myPU");
  private EntityManager em = emf.createEntityManager();
  private EntityTransaction tx = em.getTransaction();

  public void removeBook(Book book) {

    tx.begin();
    em.remove(em.merge(book));
    tx.commit();
  }



}
```

Callback Methods

Four operations

- Persisting
- Updating
- Removing
- Loading

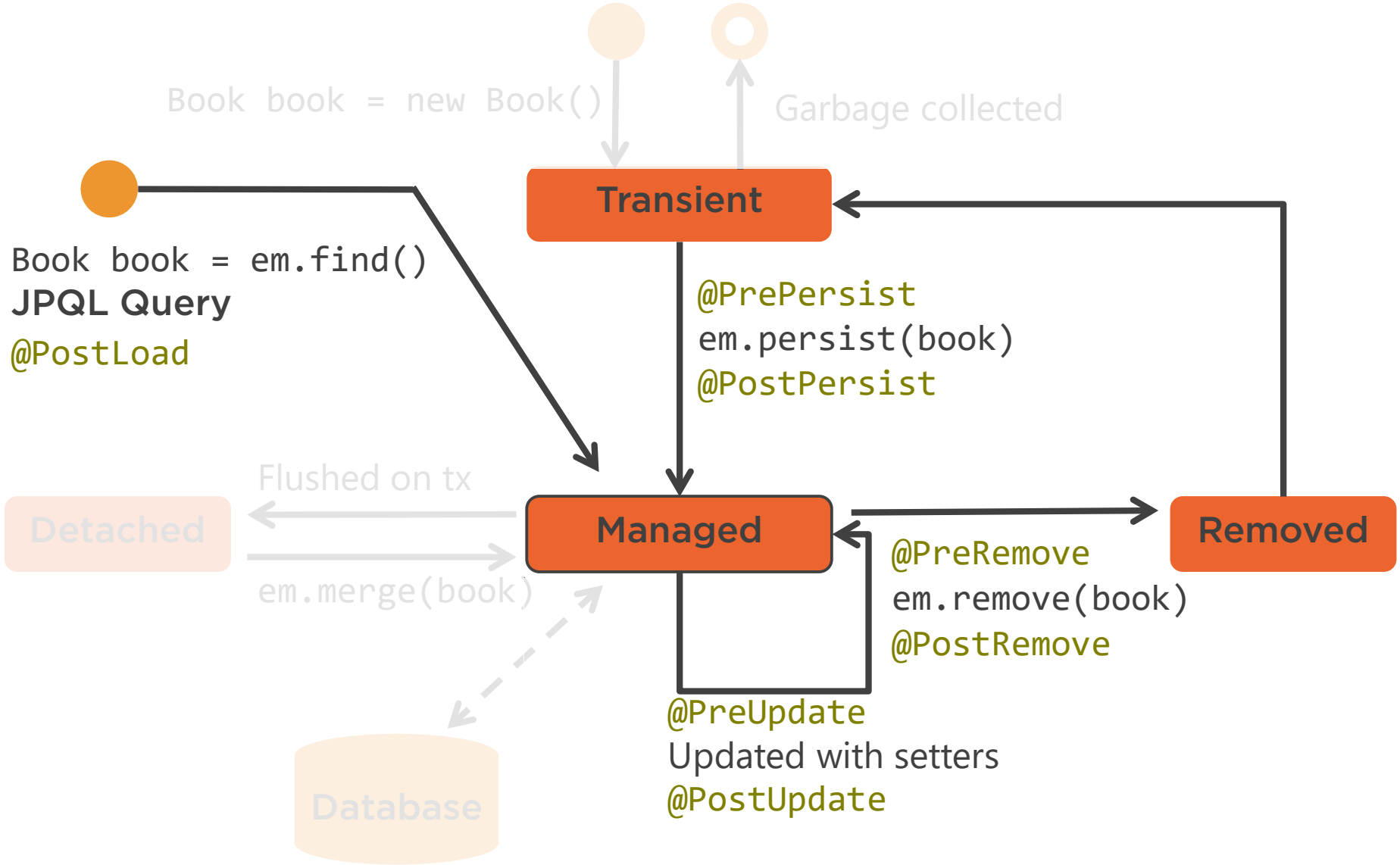
Pre event

Post event

Business method

Annotations

Pre/Post Events



Callback Annotations

`@PrePersist` **and** `@PostPersist`

`@PreUpdate` **and** `@PostUpdate`

`@PreRemove` **and** `@PostRemove`

`@PostLoad`

Public, private, protected, or package

Cannot be static of final

Throw unchecked exceptions

Entity with Callback Annotations

```
@Entity
public class Author {

  @Id @GeneratedValue
  private Long id;
  private String firstName;
  private String lastName;
  private LocalDate dateOfBirth;
  @Transient
  private Integer age;

  // Constructors, Getters & Setters
  ...
```

Entity with Callback Annotations

```
...

@PrePersist
@PreUpdate
private void validate() {
if (firstName == null || "".equals(firstName))
  throw new IllegalArgumentException("Invalid first name");
if (lastName == null || "".equals(lastName))
  throw new IllegalArgumentException("Invalid last name");
}

...
```

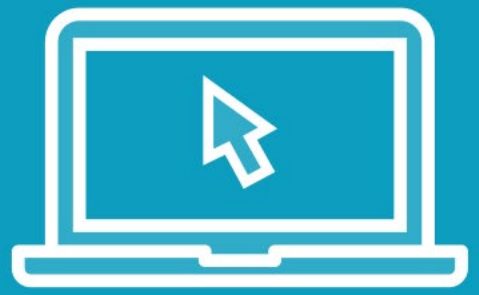
Entity with Callback Annotations

```
    ...

    @PostLoad
    @PostPersist
    @PostUpdate
    public void calculateAge() {

      age = Period.between(dateOfBirth, LocalDate.now())
                  .getYears();
    }
}
```

Demo



- Business logic
- Author entity
- Method to validate the author
- Calculate the age
- Callback annotations
- Entity life cycle

Entity Listeners

- Callback method
- Business logic related to that entity
- Entity listeners
- Business logic to a separate class
- Share it between other entities
- An entity listener is just a POJO
- `@EntityListeners` annotation

Entity Listener

```
public class ValidationListener {

  @PrePersist
  @PreUpdate
  private void validate(Author author) {
    if (author.getFirstName() == null ||
            "".equals(author.getFirstName()))
      throw new IllegalArgumentException("Invalid first name");
    if (author.getLastName() == null ||
            "".equals(author.getLastName()))
      throw new IllegalArgumentException("Invalid last name");
  }
}
```

Entity Listener

```
public class AgeCalculationListener {

  @PostLoad
  @PostPersist
  @PostUpdate
  public void calculateAge(Author author) {
    author.setAge(Period.between(author.getDateOfBirth(),
                  LocalDate.now()).getYears());
  }
}
```

Entity Listener

```
public class AgeCalculationListener {

  @PostLoad
  @PostPersist
  @PostUpdate
  public void calculateAge(Author author) {
    author.setAge(Period.between(author.getDateOfBirth(),
                  LocalDate.now()).getYears());
  }
}
```

Entity Listener

```
public class AgeCalculationListener {

  @PostLoad
  @PostPersist
  @PostUpdate
  public void calculateAge(Artist artist) {
    artist.setAge(Period.between(artist.getDateOfBirth(),
                  LocalDate.now()).getYears());
  }
}
```

Entity with Listeners

```
@Entity
@EntityListeners({
  AgeCalculationListener.class,
  ValidationListener.class
})
public class Musician extends Artist {

  @Id @GeneratedValue
  private Long id;
  private String firstName;
  private String lastName;
  private LocalDate dateOfBirth;
  @Transient
  private Integer age;
```

Entity with Listeners

```
@Entity
@EntityListeners({
  AgeCalculationListener.class,
  ValidationListener.class
})
public class Author extends Artist {

  @Id @GeneratedValue
  private Long id;
  private String firstName;
  private String lastName;
  private LocalDate dateOfBirth;
  @Transient
  private Integer age;
```

Default Listeners

Applied by default to all the entities

No annotation

Specified in XML

`@ExcludeDefaultListeners`

Default Listener

```
public class LifecycleListener {

  @PrePersist
  void prePersist(Object object) {
    System.out.println("PrePersist");
  }

  @PostPersist
  void postPersist(Object object) {
    System.out.println("PostPersist");
  }
}
```

Defining Default Listeners

```
<entity-mappings (...) version="2.2">

  <persistence-unit-metadata>
    <persistence-unit-defaults>
      <entity-listeners>
        <entity-listener class="com.pluralsight.
                               LifecycleListener"/>
      </entity-listeners>
    </persistence-unit-defaults>
  </persistence-unit-metadata>

</entity-mappings>
```

One-to-many Bidirectional

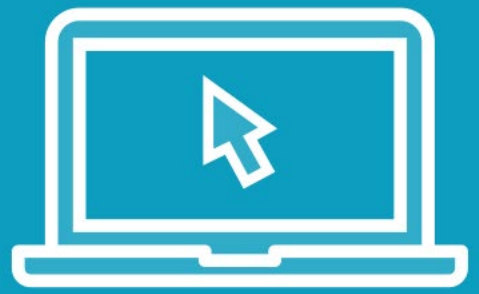
```
@Entity
public class Author extends Artist {

  // Attributes and Constructors
}
```

```
@Entity
@ExcludeDefaultListeners
public class Musician extends Artist {

  // Attributes and Constructors
}
```

Demo



Musician entity and Artist mapped superclass

Validate data and calculate age

Separate listeners

Default listener

XML file

Summary

- Entity lifecycle
- Managed vs. detached
- Entities can have business logic
- Callback methods
- Listeners
- Default listeners

Next Module

JPA integrates with Java EE

Java EE 8

Context and Dependency Injection (CDI)

Transactions (JTA)

Bean Validation

JAX-B