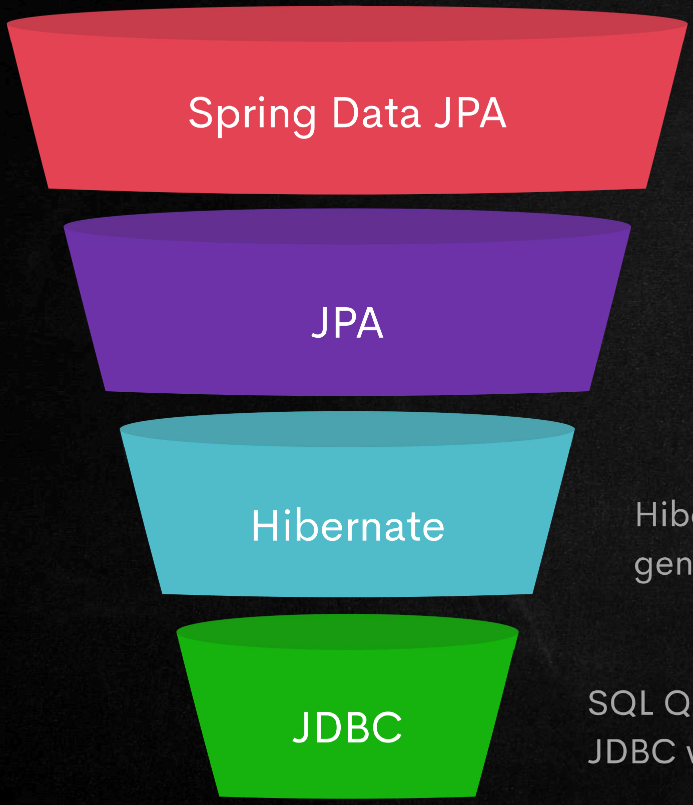
Spring Data JPA



Spring Data JPA is an abstraction layer on top of JPA to reduce the boilerplate code required to implment Data Access Object

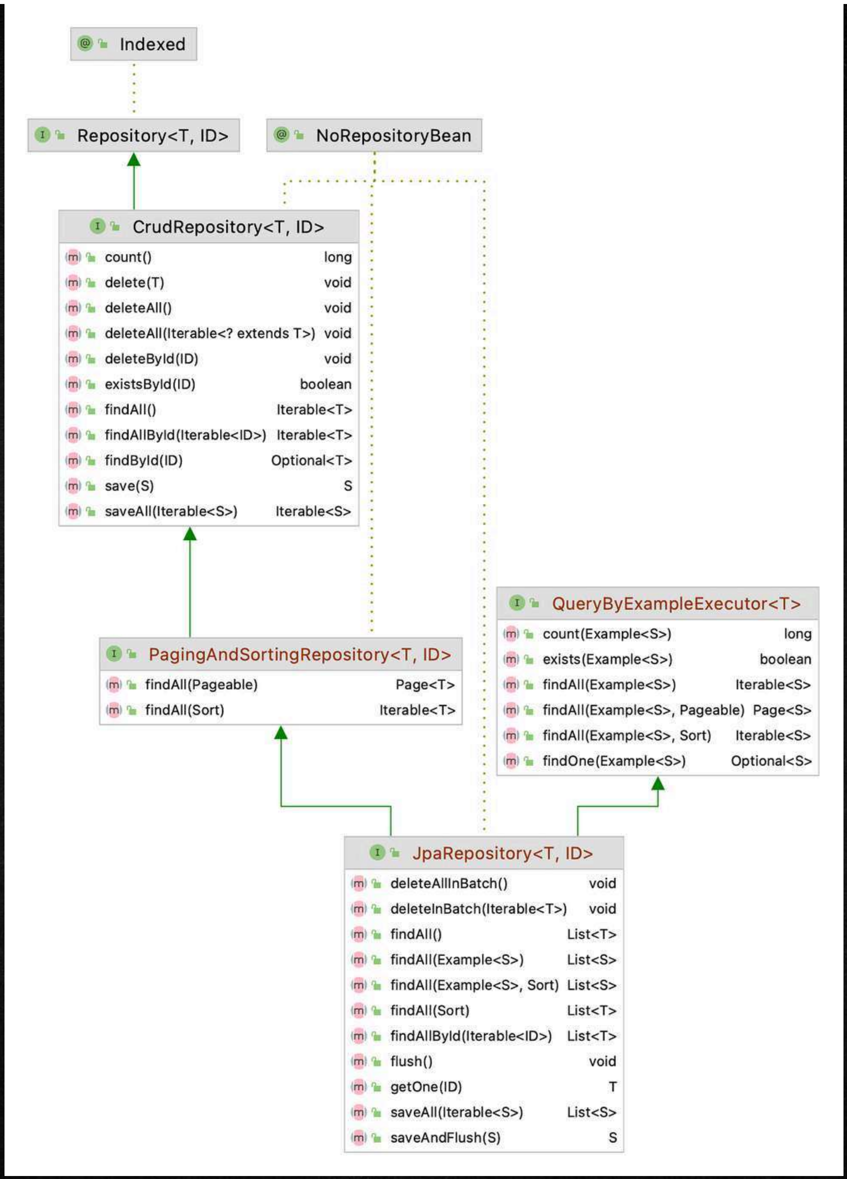
JPA (Jakarta Persistence API) is a specification that facilitates Object-Relationship mapping in JPA

Hibernate is an implementation of JPA, and it generates SQL queries

SQL Queries are executed by JDBC which connects to the Database

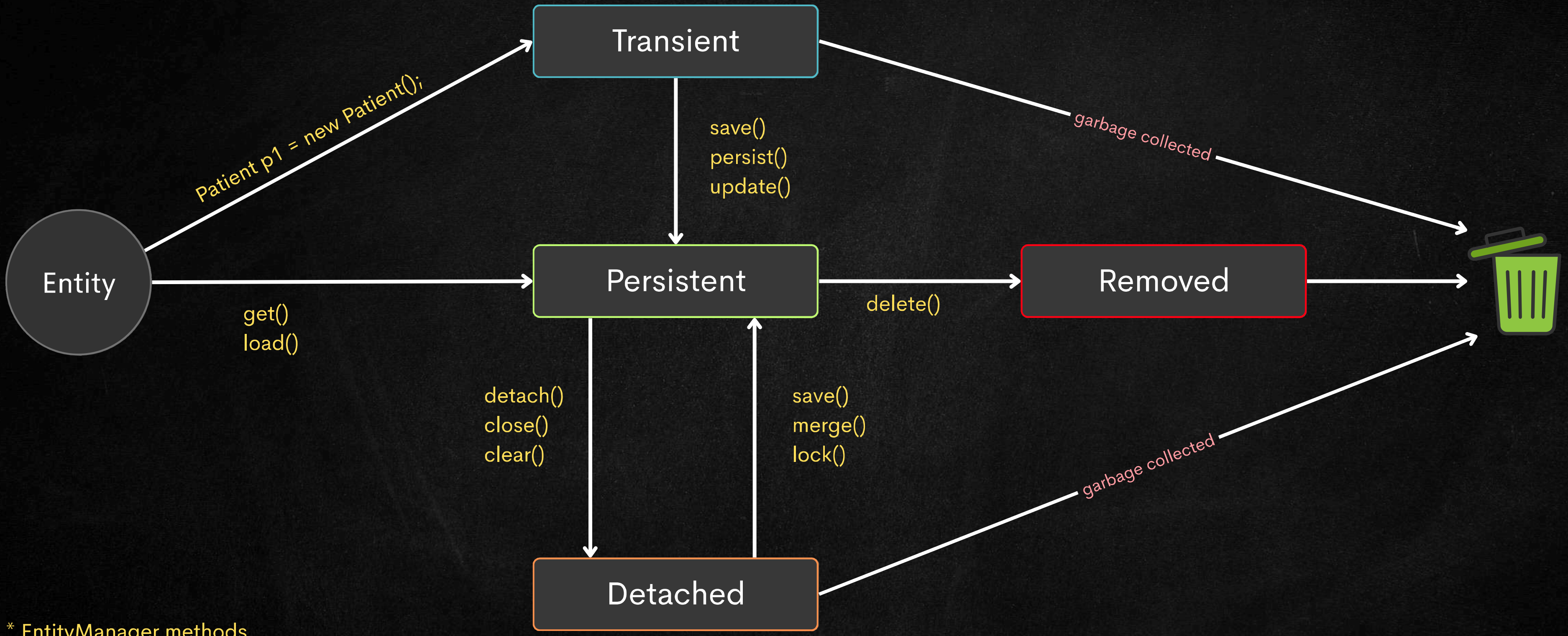
JPA Repository

CODING SHUTTLE



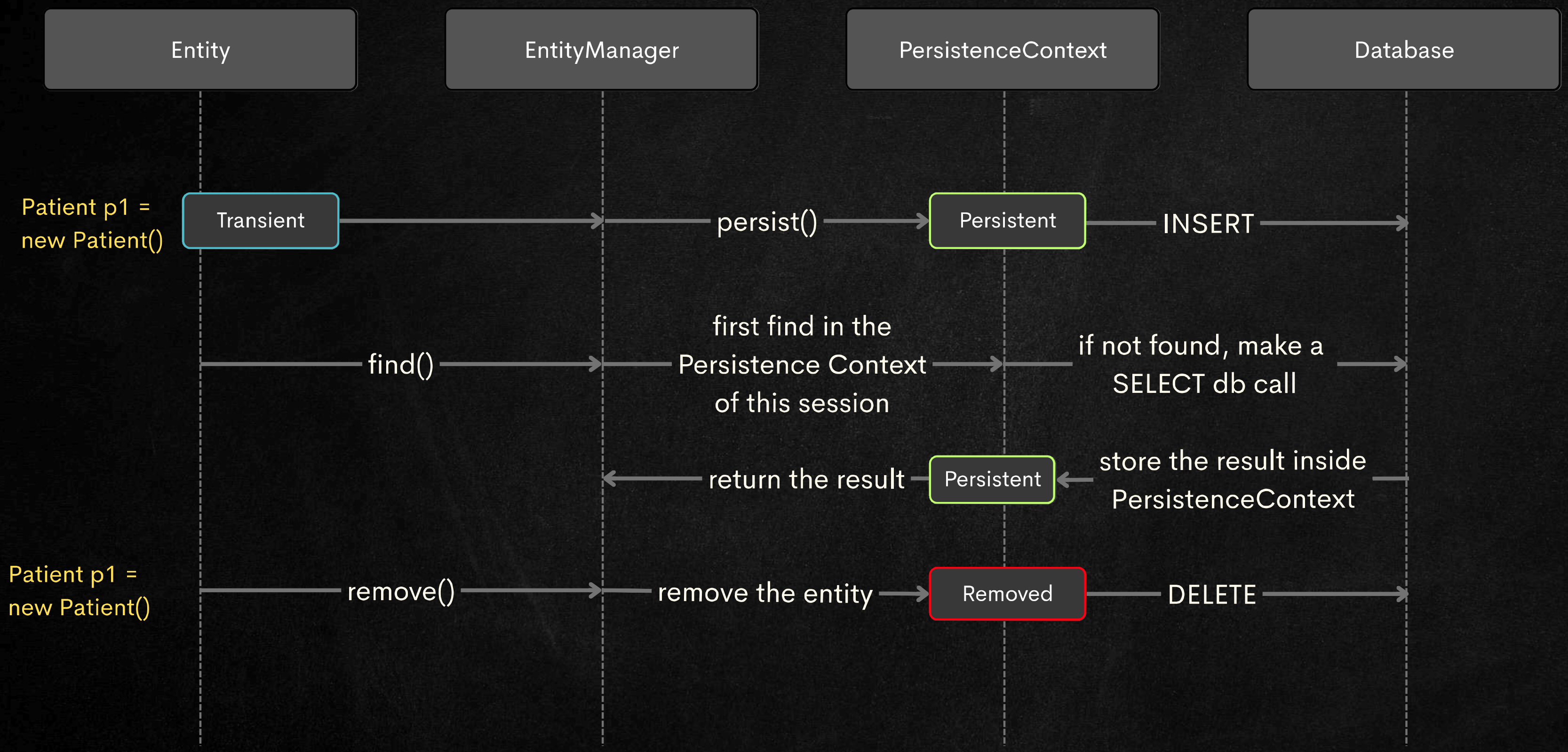
Indexed
Repository<T, ID>
NoRepositoryBean
CrudRepository<T, ID>
count() long
delete(T) void
deleteAll() void
deleteAll(Iterable<? extends T>) void
deleteById(ID) void
existsById(ID) boolean
findAll() Iterable<T>
findAllById(Iterable<ID>) Iterable<T>
findById(ID) Optional<T>
save(S) S
saveAll(Iterable<S>) Iterable<S>
PagingAndSortingRepository<T, ID>
findAll(Pageable) Page<T>
findAll(Sort) Iterable<T>
QueryByExampleExecutor<T>
count(Example<S>) long
exists(Example<S>) boolean
findAll(Example<S>) Iterable<S>
findAll(Example<S>, Pageable) Page<S>
findAll(Example<S>, Sort) Iterable<S>
findOne(Example<S>) Optional<S>
JpaRepository<T, ID>
deleteAllInBatch() void
deleteInBatch(Iterable<T>) void
findAll() List<T>
findAll(Example<S>) List<S>
findAll(Example<S>, Sort) List<S>
findAll(Sort) List<T>
findAllById(Iterable<ID>) List<T>
flush() void
getOne(ID) T
saveAll(Iterable<S>) List<S>
saveAndFlush(S) S

Hibernate - Entity Lifecycle

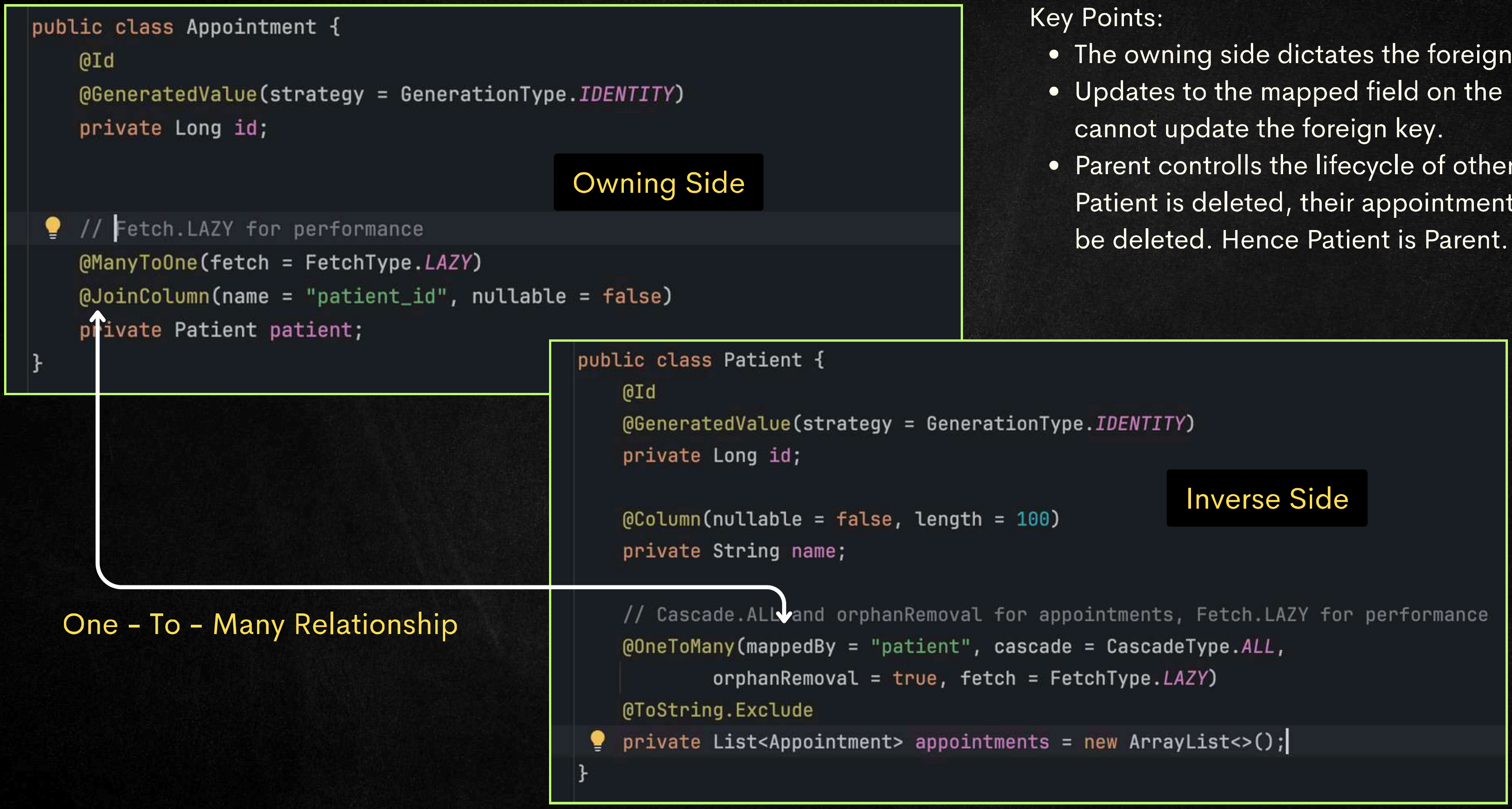


* EntityManager methods

EntityManager and PersistenceContext


Entity
EntityManager
PersistenceContext
Database
Patient p1 = new Patient()
Transient
persist()
Persistent
INSERT
find()
first find in the Persistence Context of this session
if not found, make a SELECT db call
return the result
Persistent
store the result inside PersistenceContext
Patient p1 = new Patient()
remove()
remove the entity
Removed
DELETE

Relationship Owning side and Inverse Side



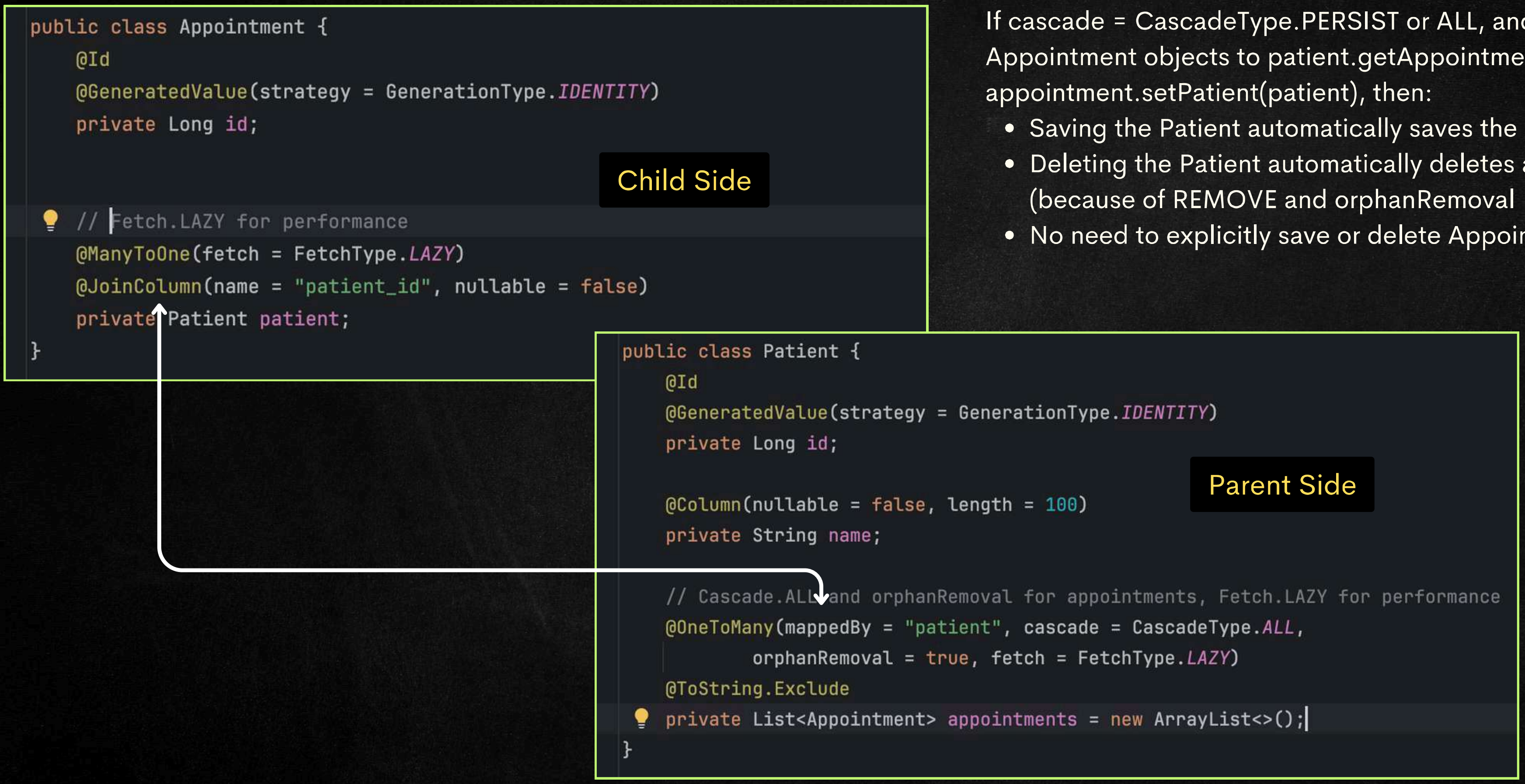
```java
public class Appointment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;


    // Fetch.LAZY for performance
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "patient_id", nullable = false)
    private Patient patient;
}
```

```java
public class Patient {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 100)
    private String name;

    // Cascade.ALL and orphanRemoval for appointments, Fetch.LAZY for performance
    @OneToMany(mappedBy = "patient", cascade = CascadeType.ALL,
            orphanRemoval = true, fetch = FetchType.LAZY)
    @ToString.Exclude
    private List<Appointment> appointments = new ArrayList<>();
}
```

Key Points:

- The owning side dictates the foreign key updates.
- Updates to the mapped field on the Inverse side cannot update the foreign key.
- Parent controlls the lifecycle of other, here if a Patient is deleted, their appointments should also be deleted. Hence Patient is Parent.

Cascading in JPA Mappings



Child Side

```java
public class Appointment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;


    // Fetch.LAZY for performance
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "patient_id", nullable = false)
    private Patient patient;
}
```

Parent Side

```java
public class Patient {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 100)
    private String name;

    // Cascade.ALL and orphanRemoval for appointments, Fetch.LAZY for performance
    @OneToMany(mappedBy = "patient", cascade = CascadeType.ALL,
            orphanRemoval = true, fetch = FetchType.LAZY)
    @ToString.Exclude
    private List<Appointment> appointments = new ArrayList<>();
}
```

If cascade = CascadeType.PERSIST or ALL, and you've added Appointment objects to patient.getAppointments() and set appointment.setPatient(patient), then:

- Saving the Patient automatically saves the Appointments.
- Deleting the Patient automatically deletes all Appointments (because of REMOVE and orphanRemoval = true).
- No need to explicitly save or delete Appointment.

Cascading in JPA Mappings

In JPA, cascading tells the persistence provider (like Hibernate) what operations to propagate from a parent entity to its related child entities automatically.

- CascadeType.PERSIST: Propagate persist (save) operation.
- CascadeType.MERGE: Propagate merge (update) operation.
- CascadeType.REMOVE: Propagate remove (delete) operation.
- CascadeType.REFRESH: Propagate refresh operation.
- CascadeType.DETACH: Propagate detach operation.
- CascadeType.ALL: Propagate all operations (PERSIST, MERGE, REMOVE, REFRESH, DETACH).

Key Points About orphanRemoval

- When It Triggers:
  - For @OneToMany: When an entity is removed from the collection (e.g., List.remove(), clear(), or reassigning a new collection).
  - For @OneToOne: When the reference is set to null or replaced with a new entity.
- Automatic Deletion:
  - Orphaned entities are deleted automatically during the JPA flush or commit operation, without needing explicit calls to entity.remove()
- Difference from CascadeType.REMOVE:
  - CascadeType.REMOVE deletes child entities only when the parent is deleted.
  - orphanRemoval = true deletes child entities when they are no longer referenced by the parent, even if the parent remains in the database.
- Use Case:
  - Ideal for relationships where the child entity has no meaning without the parent (e.g., an Appointment without a Doctor or Patient, or an Insurance without a Patient).