Embeddable and EmbeddedId Annotations

🚗 Understanding `@Embeddable` and `@EmbeddedId`

🔷 1️⃣ The Normal Case — Single Primary Key

Usually, we create an entity like this:

```
@Entity
public class Car {
   @Id
   private int id;
   private String companyName;
   private double price;
}
```

✅ Here, `id` is the **primary key** — one single column in the database uniquely identifies each row.

🔷 2️⃣ The Problem — When One Column is Not Enough

But sometimes, one column is **not enough to uniquely identify** an entity.

👉 Example:

Two cars might have the same `id`, but **different engine numbers** — or vice versa.

So, to **uniquely identify a car**, you might need a **combination** of:

- `id`
- `engineNumber`

📘 This combination becomes your **composite key**.

🔷 3️⃣ The Solution — `@Embeddable` + `@EmbeddedId`

To create a **composite key**, we use two annotations together:

Annotation	Used On	Purpose
`@Embeddable`	A **separate class**	Marks it as a class that can be **embedded inside another entity**
`@EmbeddedId`	Inside the **entity class**	Marks the field as the **composite primary key** of that entity

🔷 4️⃣ Your Code — Explained Perfectly 👇

📦 Class 1: `CarId` → the *composite key class*

```
@Embeddable
public class CarId {
   private int id;
   private String engineNumber;

   // Constructors, getters, setters, toString()
}
```

💡 **Explanation:**

- `@Embeddable` tells JPA:

  > "This class is not a full entity by itself.
  >
  > It will be embedded inside another entity to form a composite key."

- The fields `id` and `engineNumber` are the **columns** that together form the **primary key**.

🚘 Class 2: `Car` → the *main entity*

```
@Entity
public class Car {

    @EmbeddedId
    private CarId carId;

    private String companyName;
    private double price;
}
```

💡 **Explanation:**

- `@EmbeddedId` means:

  > "The primary key of this entity is an embedded object (CarId)."

- JPA will **combine the columns of CarId** (`id`, `engineNumber`) into the Car table as its **composite primary key**.

🔷 5️⃣ Database Table Representation

After running this code with JPA/Hibernate, your **Car table** will look like this 👇

id	engine_number	company_name	price
1	ENG123	BMW	70000
2	ENG987	Tesla	85000

✅ The **primary key** will be a **combination** of `(id, engine_number)`.

🔷 6️⃣ Important: Equals and HashCode

When you use an `@Embeddable` class as a key (like `CarId`),

you must **override** `equals()` **and** `hashCode()` — because JPA uses them internally for identity management.

Example 👇

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof CarId)) return false;
    CarId carId = (CarId) o;
    return id == carId.id &&
           Objects.equals(engineNumber, carId.engineNumber);
}

@Override
public int hashCode() {
    return Objects.hash(id, engineNumber);
}
```

Without these, JPA may not correctly identify duplicates or manage caching properly.

🔷 7️⃣ Advantages

✅ Cleaner design — composite key logic separated from entity

✅ More readable and reusable

✅ Matches real-world identifiers (like ProductID + StoreID, CarID + EngineNumber)

✅ Supported by Hibernate and all JPA providers

🔷 8️⃣ Real-Life Analogy

Imagine a **car registration system** at an RTO (Regional Transport Office):

RTO ID	Engine Number	Owner Name
TN10	ENG500	Sai
TN10	ENG501	Rahul

RTO ID	Engine Number	Owner Name
TN11	ENG500	Deepak

If you just used `engineNumber`, two cars from different RTOs could conflict.

If you just used `RTO ID`, you couldn't identify unique cars within the same RTO.

✅ So you use **both together** — `(RTO ID, Engine Number)` — that's your **@Embeddable composite key**.

🔷 9️⃣ Summary Table

Concept	Annotation	Example	Description
Composite Key Class	`@Embeddable`	`CarId`	Contains multiple fields for the key
Used in Entity	`@EmbeddedId`	`Car carId;`	Embeds composite key into main entity
Equals/HashCode	Must override	In `CarId`	Needed for JPA identity checks
Table PK	Combined columns	(id, engineNumber)	Forms composite primary key

---