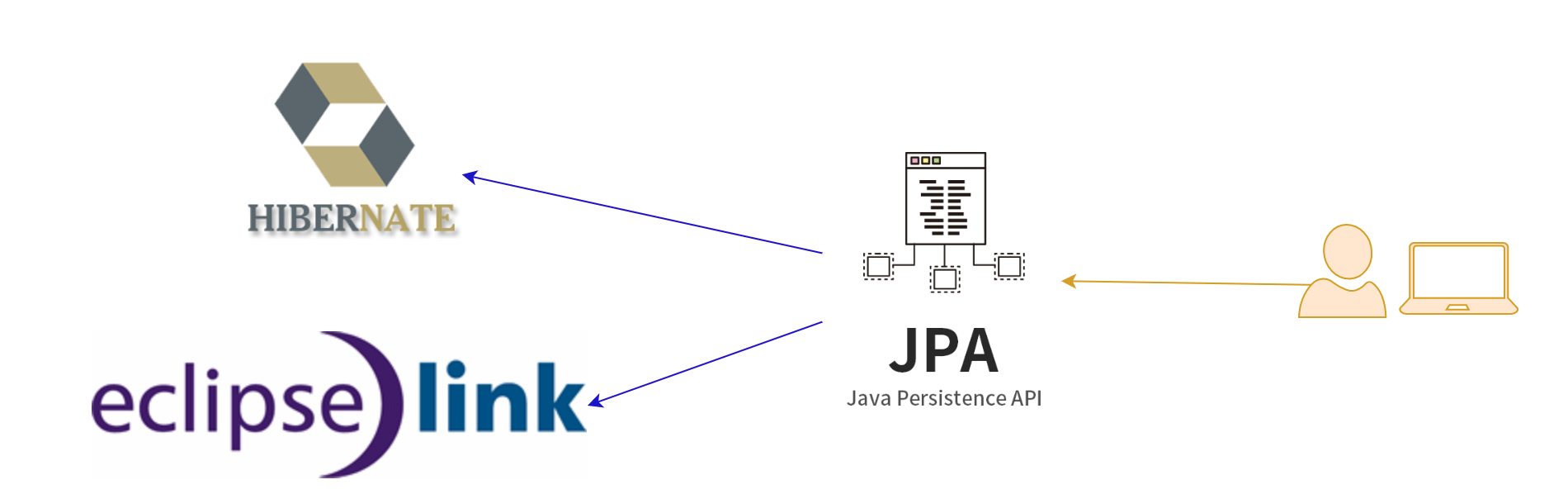
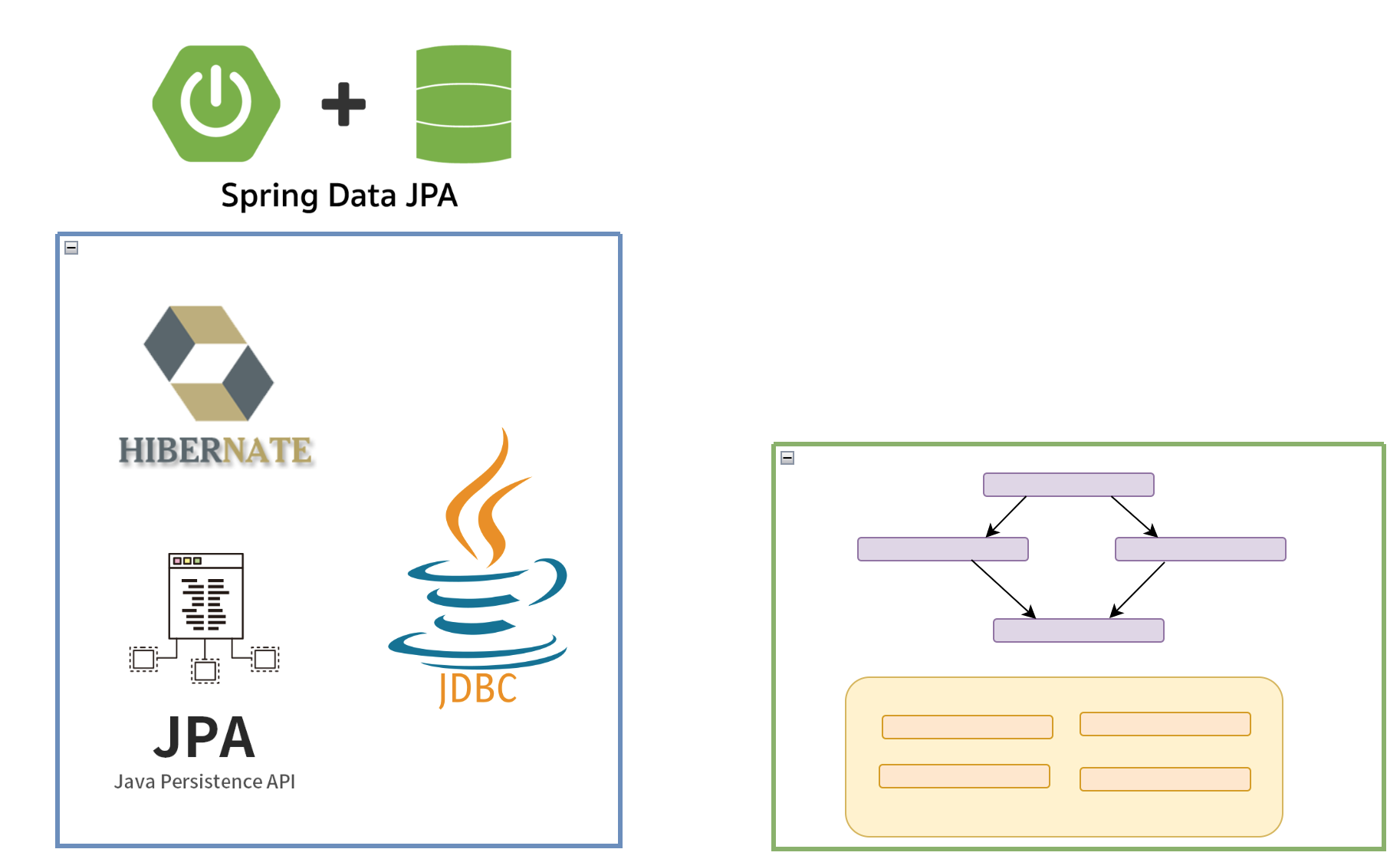
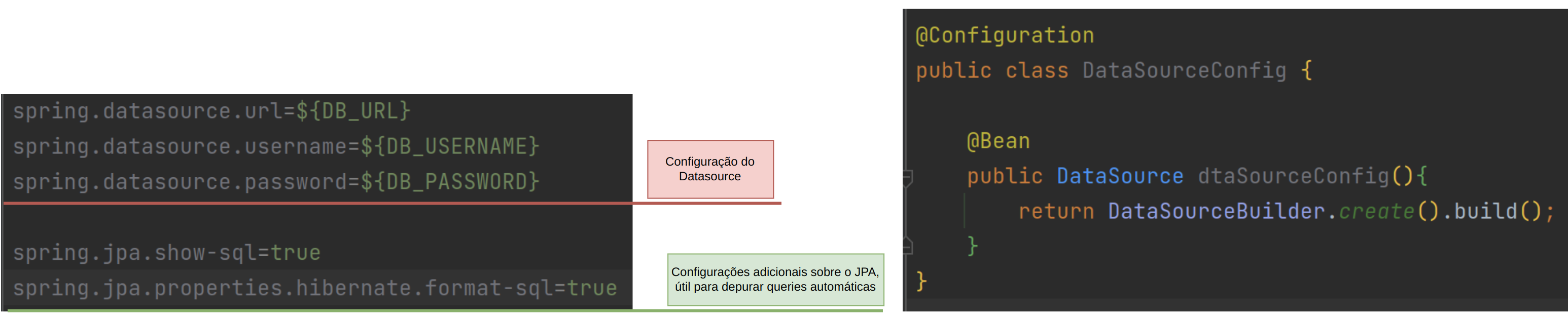
**JPA e ORM**

**JPA é uma especificação do Java EE**, ORM são frameworks para mapeamento de objeto relacional que implementam as especificações do JPA:

**Spring Data JPA**

O **projeto SDJ do Spring** é uma facilitação de interação com operações de banco de dados, então ele **reúne vários frameworks desse contexto *(JPA, Hibernate)*** e ainda fornece **Interfaces com implementações em tempo de execução** que evitam código boiler plate:

Quando a gente usa o **SDJ** é importante configurar um **DataSource** *(é um objeto que contém as informações de conexão com o database)* que pode ser configurado como classe ou direto no properties:



**Geração de PK**

Você pode passar a responsabilidade da **geração da PK** pro banco de dados para que ele faça isso de maneira automatizada, existem alguns valores constantes para geração de chaves primárias no JPA, ***você define o valor de acordo com o banco de dados que está usando***:



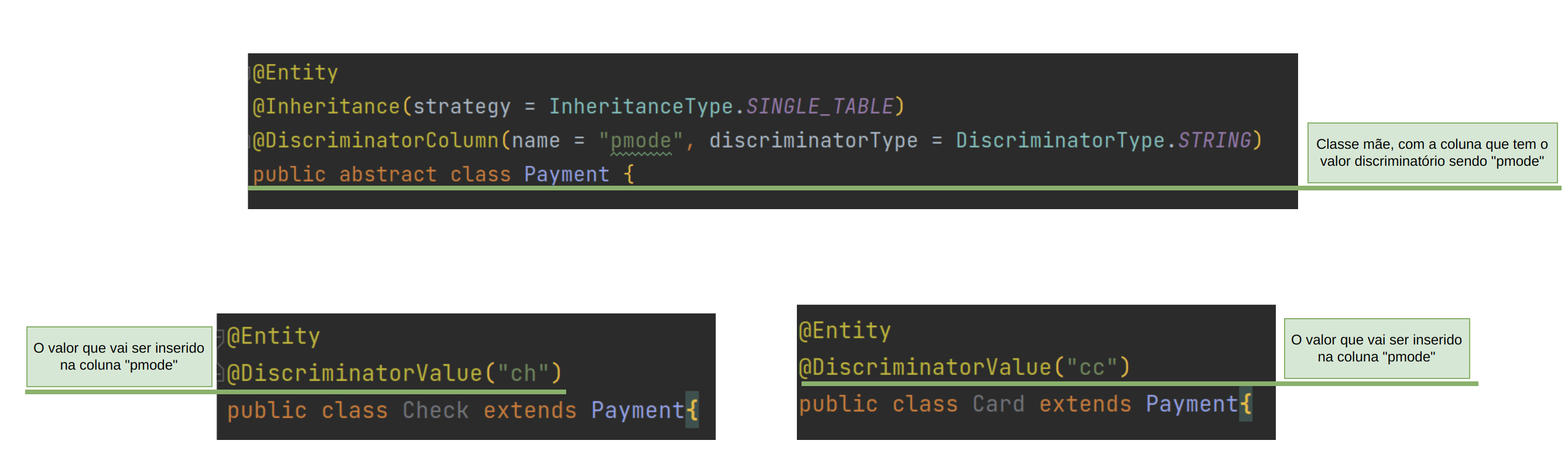
**AUTO =** o DB se virá  
**IDENTITY =** A PK vai ser um campo auto\_increment  
**SEQUENCE =** você usa a estratégia de SEQUENCE do database, é preciso dizer ao Hibernate a estratégia.  
**TABLE** **=** usa uma tabela especial para gerenciar as PK, ele salva o valor numa tabela e usa como referência para criar o próximo valor da PK.

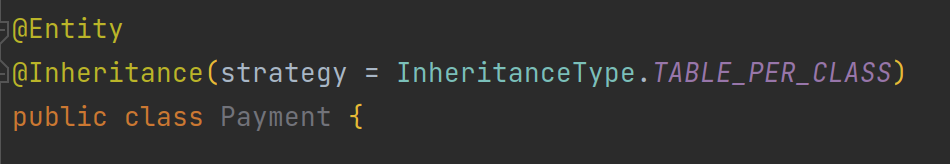
Uma chave composta pode ser representada de algumas maneiras.

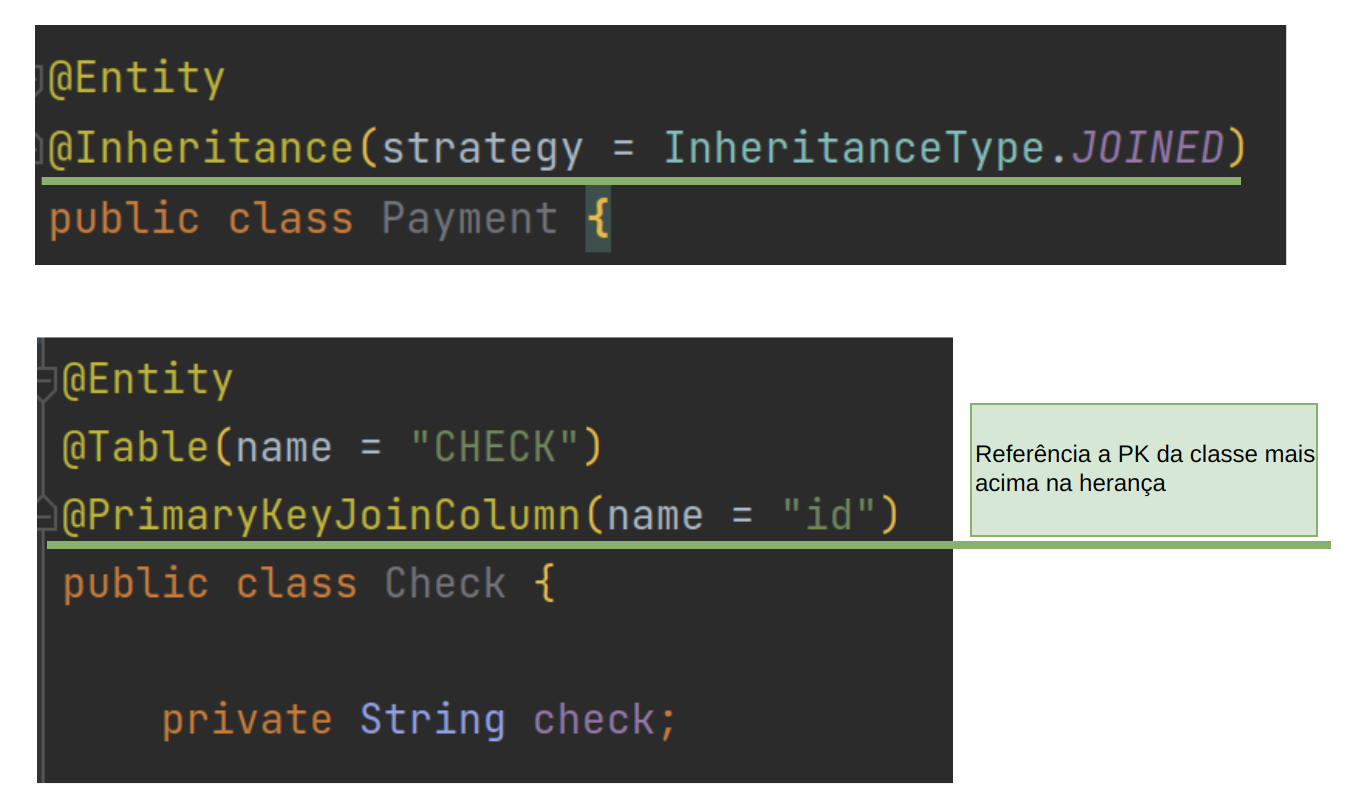
**@IdClass**  
**@Embeddable**  
**@EmbeddedID**

**Mapeando Herança em ORM**

Existe algumas abordagens para mapear entidades que tem herança. Uma delas é a **SINGLE TABLE**, onde todos os atributos da herança vão para uma tabela e cada linha vai ser descriminada por uma coluna que representa a classe específica de toda a herança:

  
**TABLE PER CLASS:** Aqui a estratégia é fazer uma tabela para cada classe concreta da herança, basta definir na classe mãe a estratégia e mapear normalmente as classes filhas como tabelas próprias:



**JOINED:** Aqui cada classe da hierarquia tem sua própria tabela, cada uma carregando seus respectivos atributos. A vantagem aqui é ter tabelas que guardam valores mínimos e a desvantagem é que o **ORM vai precisar fazer um join em todas que ele precisar pra um determinado retorno:**

**Mapeando Composição**

As composições podem ser mapeadas através de Embeddable e Embedadded classes, essas classes se tornam uma única tabela no banco de dados:

**Cuidados com non readable queries**

Queries que não são só readable precisam ser mapeadas com **@Modifying**. Queries que não são apenas consultas também precisam de **@Transactional**, isso garante que a operação deve acontecer por completo ou não acontecer, mantendo a transação “viva” durante os processos:

**Paginação**

A paginação vem de uma interface específica de PaginAndSorting, se você estender de JpaRepository já vai ter acesso a ela, buscas paginadas precisam de um objeto de **Paginação:** 