As vezes necessitamos de **querys mais "flexiveis"**, por exemplo, eu posso ter uma **query** que **faz uma filtragem por nome, data e preço de um produto**. Mas eu posso querer um **comportamento de SELECT que seja de acordo com o informado**, ou seja, pode ser **que eu tenha só o nome, então vou usar só o nome**, ou pode ser que **eu tenha a data e o preço, então uso ambos e etc**...

Para esse tipo de situação existe a **Criteria API**, ela serve para criar **querys dinâmicas em troca de mais de complexidade e verbosidade** já que ela envolve bastante objetos.

A ideia do criteria é ter um **builder** que recebe **predicados** que serão aplicados na **query criteria** tendo como alvo a **Entidade root**, esses predicados são filtros como ***like, >=, where etc..*** Que vão **sendo adicionados se passaram na checagem e construindo dinamicamente a query** que depois é executada pelo EntityManager:

```java
public List<Produto> buscarPorParametros(String nome, BigDecimal preco, LocalDate dataCadastro){

    CriteriaBuilder builder = entityManager.getCriteriaBuilder();
    CriteriaQuery<Produto> query = builder.createQuery(Produto.class);
    Root<Produto> from = query.from(Produto.class);

    Predicate filtros = builder.and();

    if(nome != null && !nome.trim().isEmpty()){
        filtros = builder.and(filtros, builder.equal(from.get("nome"),nome));
    }
    if(dataCadastro != null){
        filtros = builder.and(filtros, builder.equal(from.get("dataCadastro"), dataCadastro));
    }
    if(preco != null){
        filtros = builder.and(filtros, builder.equal(from.get("preco"), preco));
    }

    query.where(filtros);
    return entityManager.createQuery(query).getResultList();
}
```