@NamedQueries or @NamedQuery

They are useful for organising, maintaining, optimising query definitions in spring.

Use cases:

1. Encapsulation of query logic (separate query definitions from the rest of the application logic)
2. Reusability
3. Optimize the performance (parsed, validated and optimised during application startup)
4. Centralised query definition

They might not be as flexible as dynamic query for example when we use query dsl or criteria api or build queries on varying conditions.

We can also create @NamedNativeQuery or @NamedNativeQueries

e.g annotation used over the entity

Name should be like <name of Entity>.<name of query>

```
@NamedQuery(
        name = “Author.findByNamedQuery”,
        query = “select a from author a where a.age >= :age”
)
```

We can directly mention this in the repository.

```
public interface AuthorRepository extends JPA…. {
        List<Author> findByNamedQuery(@Param(“age”) int age);
}
```

There is also other way around:

```
public class AuthorRepository {

    @PersistenceUnit
    private EntityManagerFactory entityManagerFactory;

    public List<Author> findByName(String name) {
        EntityManager entityManager =
entityManagerFactory.createEntityManager();
        return entityManager.createNamedQuery("Author.findByNamedQuery”,
Author.class)
                        .setParameter("name", name)
```

```
                    .getResultList();
    }
}
```

For query containing update sql we have to mention @Modifying and @Transactional over the method in repository.