Pós-Graduação Lato Sensu

Curso de Especialização em Tecnologia Java

APOSTILA – IMPLEMENTANDO REPOSITORY E SERVICE

Professor Hugo Baker Goveia

hbakergoveia@hotmail.com

UTFPR

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAMPUS CORNÉLIO PROCÓPIO

1. Implementando o Repository

A camada de repository em um projeto Spring Data JPA tem a responsabilidade de abstrair o acesso aos dados e fornecer métodos de alto nível para realizar operações de persistência e consulta no banco de dados.

A camada de repository atua como uma interface entre a camada de serviço (ou de negócios) e o mecanismo de persistência (banco de dados).

Conforme estudado no Livro "Repositórios (Parte 1)" da disciplina, o repository encapsula a lógica de acesso aos dados, fornecendo métodos que são utilizados pela camada de serviço para interagir com as entidades do domínio.

As principais funções da camada de repository são:

Abstração do acesso aos dados: A camada de repository oculta os detalhes específicos de como os dados são armazenados e consultados no banco de dados. Isso permite que a camada de serviço se concentre na lógica de negócios, sem precisar se preocupar com os detalhes de persistência.

Definição de operações de consulta: A camada de repository define métodos para realizar operações de consulta no banco de dados. Esses métodos podem ser personalizados para atender às necessidades específicas do projeto, como consultas simples, consultas personalizadas ou consultas complexas envolvendo junções, agregações, etc.

Herança de recursos do Spring Data JPA: O Spring Data JPA fornece um conjunto de interfaces base, como CrudRepository, JpaRepository, PagingAndSortingRepository, que já incluem métodos de acesso padrão para operações básicas de CRUD, além de operações de paginação e ordenação. A camada de repository pode estender essas interfaces para herdar esses recursos prontos para uso, economizando tempo e esforço no desenvolvimento de código repetitivo.

Implementação de consultas personalizadas: Além dos métodos herdados das interfaces base do Spring Data JPA, a camada de repository pode incluir métodos personalizados para consultas específicas do domínio. Esses métodos podem ser definidos usando convenções de nomenclatura, consultas com anotações **@Query** que será amplamente abordada na disciplina, ou também consultas dinâmicas com o uso de Criteria, Example ou Specifications.

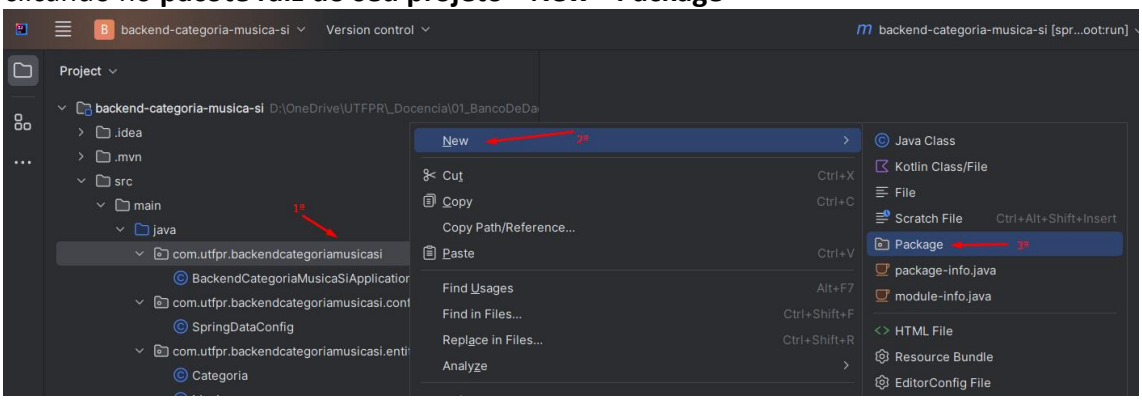
Suporte a transações: O Spring Data JPA gerencia automaticamente as transações em métodos do repositório. Isso significa que você pode anotar os métodos do repositório com @Transactional para garantir a consistência dos dados e a atomicidade das operações.

Para concluir, entendemos então que a camada de repository no Spring Data JPA fornece uma abstração de acesso aos dados e oferece métodos de alto nível para realizar

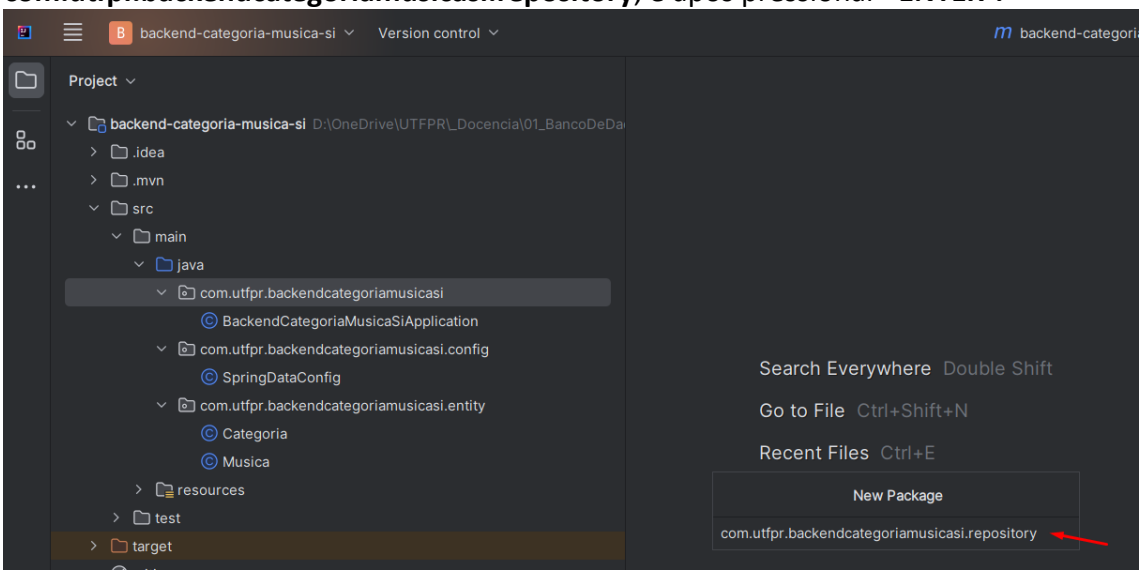
operações de persistência e consulta. Ela permite separar a lógica de acesso aos dados da lógica de negócios, facilita o desenvolvimento de consultas personalizadas e oferece recursos prontos para uso, tornando o acesso aos dados mais simples e eficiente.

Etapas para criar a interface repository:

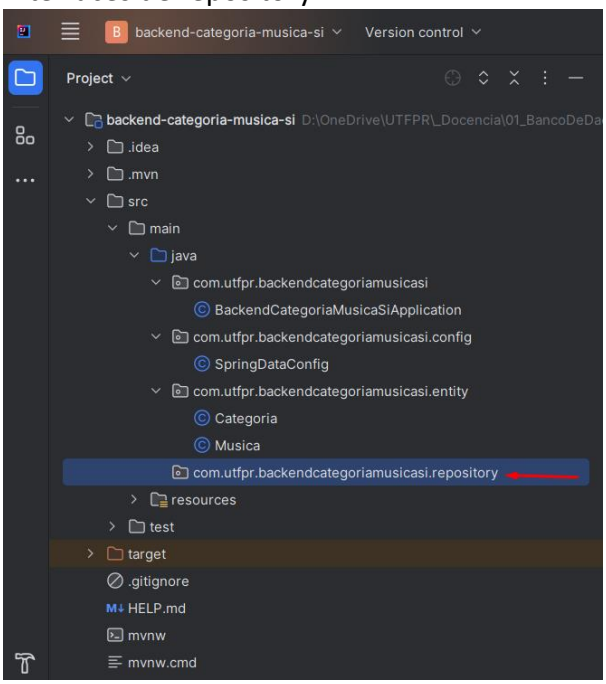
1. Primeiramente vamos criar o pacote onde colocaremos nossas interfaces, clicando no **pacote raiz do seu projeto->New->Package**



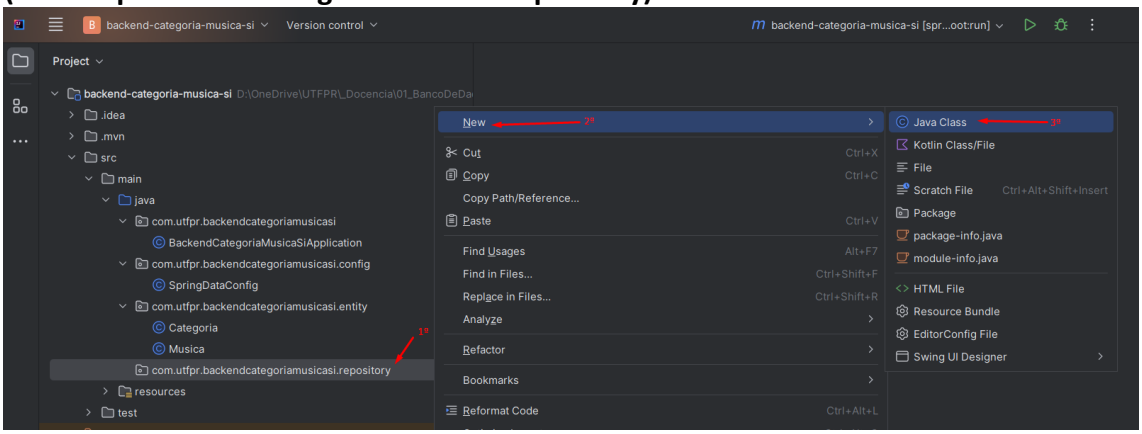
2. Na modal que abrir, informar o nome do pacote que será **com.utfpr.backendcategoriamusicasi.repository**, e após pressionar **"ENTER".**



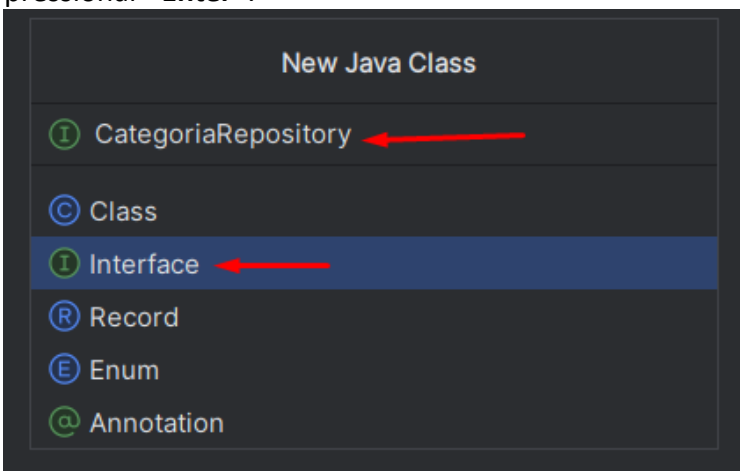
3. Após essa ação o novo pacote é criado e será nele que iremos criar nossas interfaces de repository.



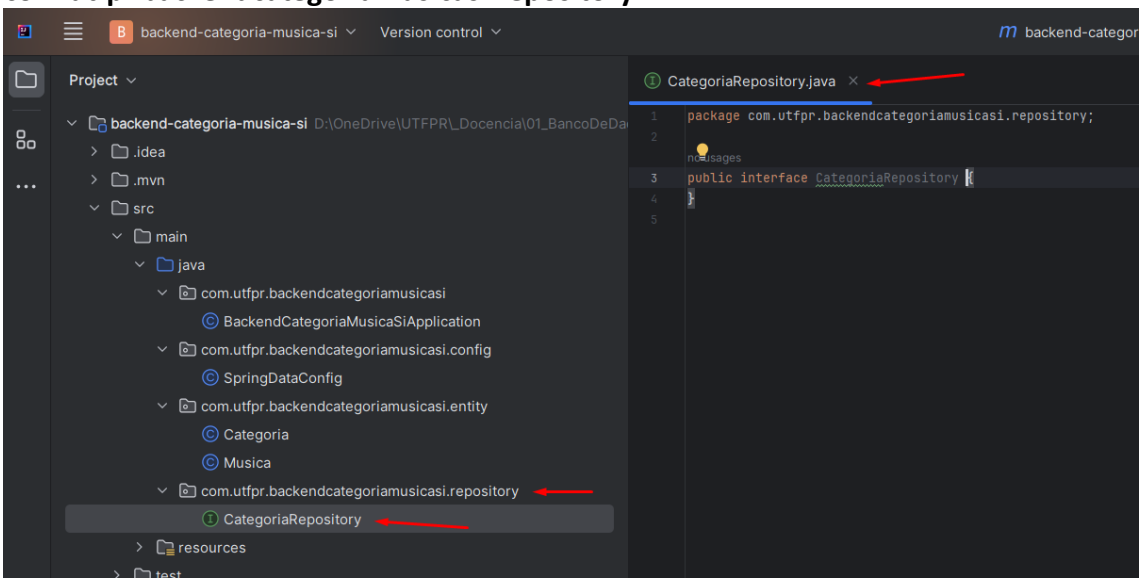
4. Para criar a interface repository, clicar com o botão direito no pacote **(com.utfpr.backendcategoriamusicasi.repository) -> New -> Java Class**



5. Escolher o **“Interface”**, informar o nome da interface **“CategoriaRepository”** e pressionar **“Enter”**.

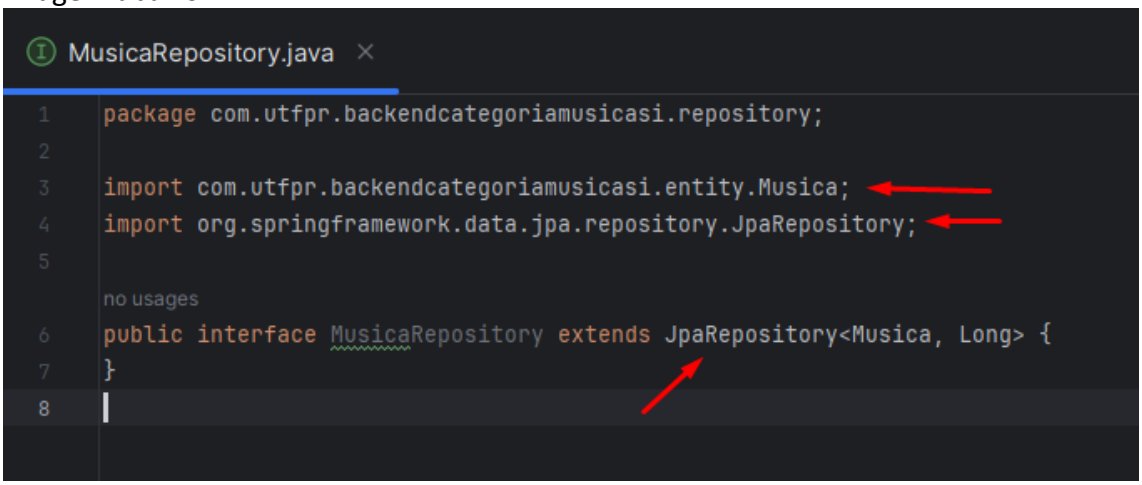


6. Observe pela imagem abaixo que a interface **CategoriaRepository** foi criada dentro do pacote que definimos **com.utfpr.backendcategoriamusicasi.repository**



7. Agora devemos realizar a implementação para a interface repository estender o JpaRepository e fazer as importações necessárias.

8. Repita o processo para implementar a interface repository de **Música** conforme imagem abaixo.



2. Implementando o Service

A camada de serviço (@Service) em um projeto Spring Data JPA tem a responsabilidade de encapsular a lógica de negócios e coordenar as operações entre a camada de controle (Controller) e a camada de acesso aos dados (Repository). É importante lembrar que nessa disciplina não iremos abordar a camada de controle (Controller), pois focaremos na persistência dos dados e assuntos pertinentes ao banco de dados.

A camada de serviço desempenha um papel crucial na separação de preocupações dentro da arquitetura de um projeto. Ela permite isolar e centralizar a lógica de negócios, tornando o código mais modular, reutilizável e de fácil manutenção.

Algumas das funções da camada de serviço são:

Coordenação das operações: A camada de serviço é responsável por orquestrar a execução das operações de negócio, geralmente envolvendo múltiplas operações de acesso a dados. Ela coordena a interação entre os diferentes objetos de domínio e a camada de acesso aos dados para realizar ações mais complexas, como processamento de regras de negócio, validações, cálculos, etc.

Validação de dados: A camada de serviço realiza validações dos dados recebidos da camada de controle, garantindo a integridade dos dados e a conformidade com as regras de negócio. Isso pode incluir validações de campos obrigatórios, formatos, restrições de negócio, entre outros.

Transformação de dados: A camada de serviço é responsável por converter os objetos de transferência de dados (DTO) recebidos pela camada de controle em objetos de domínio apropriados para serem processados pelas operações de negócio. Ela também realiza a conversão inversa, convertendo objetos de domínio em DTOs para serem enviados de volta à camada de controle.

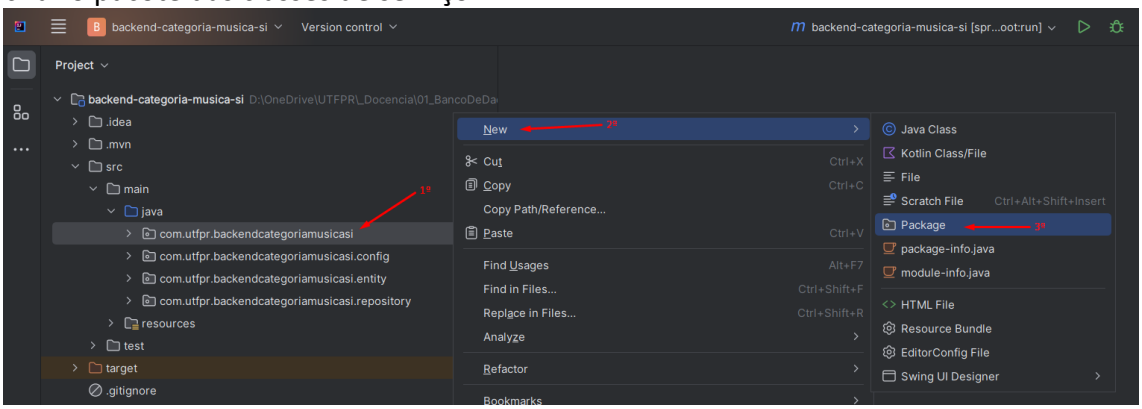
Transações: A camada de serviço é o local onde as transações podem ser iniciadas e controladas. Ela pode definir métodos transacionais usando a anotação @Transactional, garantindo que um conjunto de operações de negócio seja executado de forma atômica e consistente.

Exposição de serviços: A camada de serviço expõe uma interface para a camada de controle, definindo métodos que encapsulam as operações de negócio disponíveis para serem consumidas pelos clientes da aplicação. Esses métodos podem ser invocados pela camada de controle em resposta a eventos do usuário ou solicitações externas.

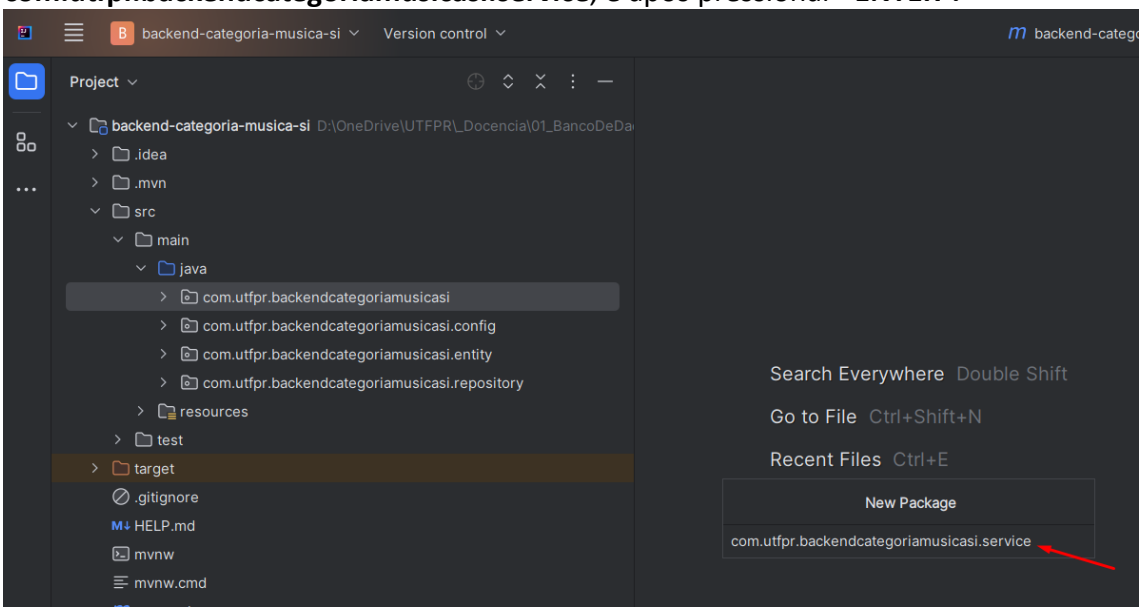
Sendo assim, a camada de serviço (@Service) **é responsável por encapsular a lógica de negócios**, coordenar as operações entre a camada de controle e a camada de acesso aos dados, realizar validações, transformações de dados e garantir a consistência e atomicidade das operações por meio do controle de transações. Ela desempenha um papel fundamental na organização e modularização do código, promovendo a separação de preocupações e facilitando a manutenção e evolução do projeto.

Etapas para criar a classe Service (@Service):

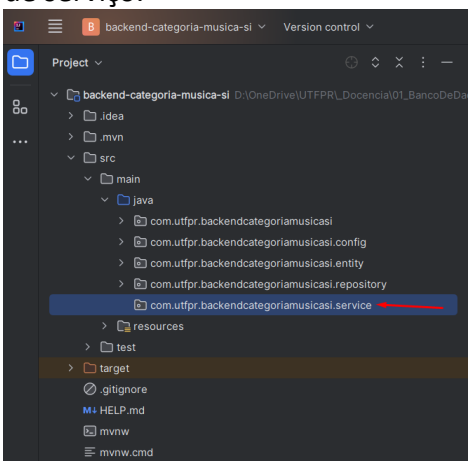
1. Clicar com o botão direito no **pacote raiz do seu projeto -> New -> Package** para criar o pacote das classes de serviço.



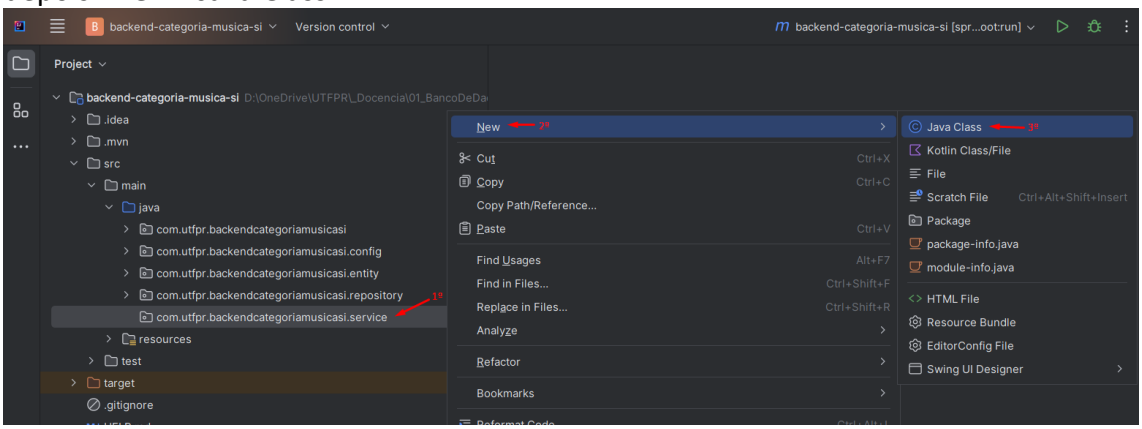
2. Na modal que abrir, informar o nome do pacote que será **com.utfpr.backendcategoriamusicasi.service**, e após pressionar **"ENTER".**



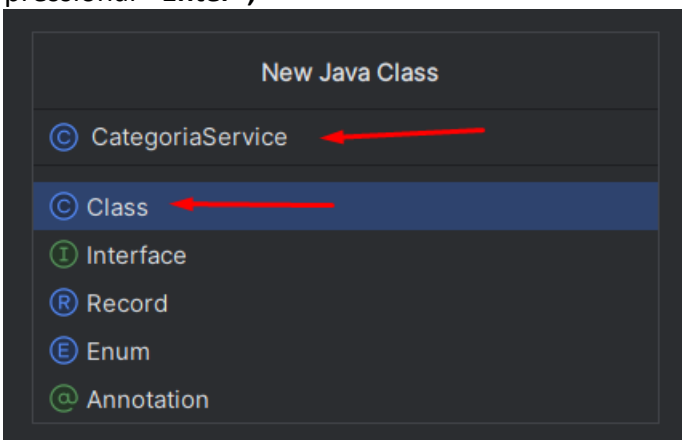
3. Após essa ação o novo pacote é criado e será nele que iremos criar nossas classes de serviço.



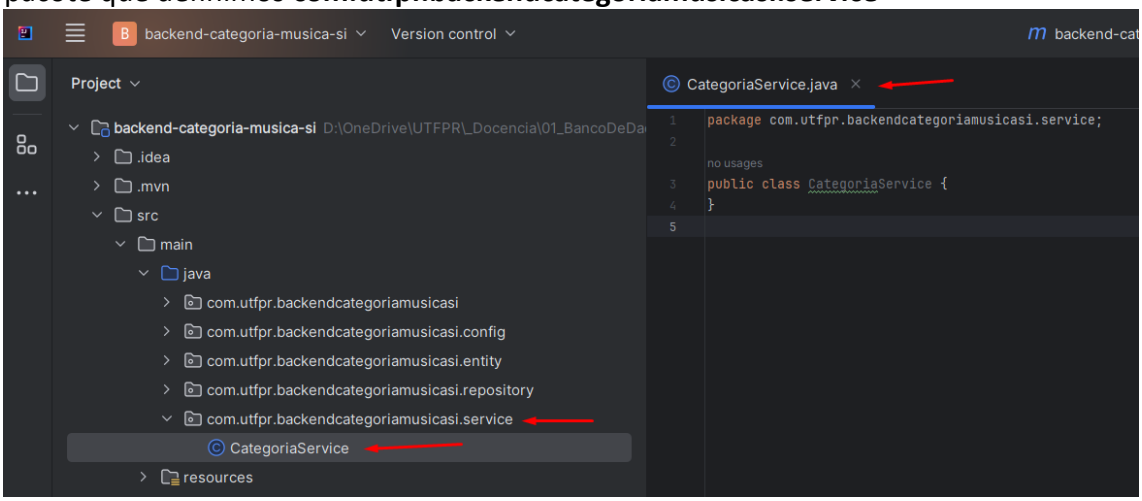
4. Para criar a classe CategoriaService, você deve clicar com o botão direito mouse no pacote **(com.utfpr.backendcategoriamusicasi.service)** que acabamos de criar, depois **“New->Java Class”.**



5. Manter o **“Class”** selecionado, informar o nome da classe **“CategoriaService”** e pressionar **“Enter”;**



6. Observe pela imagem abaixo que a classe **CategoriaService** foi criada dentro do pacote que definimos **com.utfpr.backendcategoriamusicasi.service**



7. Agora devemos realizar a implementação para a classe service (@Service) de acordo **com o exemplo abaixo**. Conforme foi apresentado no Livro do bloco de

repository, podemos observar abaixo o uso da anotação @Service inserida acima da assinatura da classe para transformar essa classe em um bean gerenciável pelo spring.
Temos também a implementação da injeção de dependência do repository através da anotação @Autowired, dessa forma podemos usar aqui os métodos de CRUD presentes no nosso repository.

CategoriaService.java

```java
package com.utfpr.backendcategoriamusicasi.service;

import com.utfpr.backendcategoriamusicasi.repository.CategoriaRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

no usages
@Service
public class CategoriaService {

    no usages
    @Autowired
    private CategoriaRepository repository;
}
```

8. Vamos implementar um método para testar o acesso aos métodos do nosso repository, observe abaixo que ao usar o repository podemos ter acesso ao método findAll() dentre outros, que facilitam nossa implementação.

CategoriaService.java

```java
package com.utfpr.backendcategoriamusicasi.service;

import com.utfpr.backendcategoriamusicasi.entity.Categoria;
import com.utfpr.backendcategoriamusicasi.repository.CategoriaRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

no usages
@Service
public class CategoriaService {

    1 usage
    @Autowired
    private CategoriaRepository repository;

    no usages
    public List<Categoria> listarTodasCategorias() {
        return repository.findAll();
    }
}
```

9. Repita o processo para implementar a classe Service de **Música** conforme imagem abaixo.