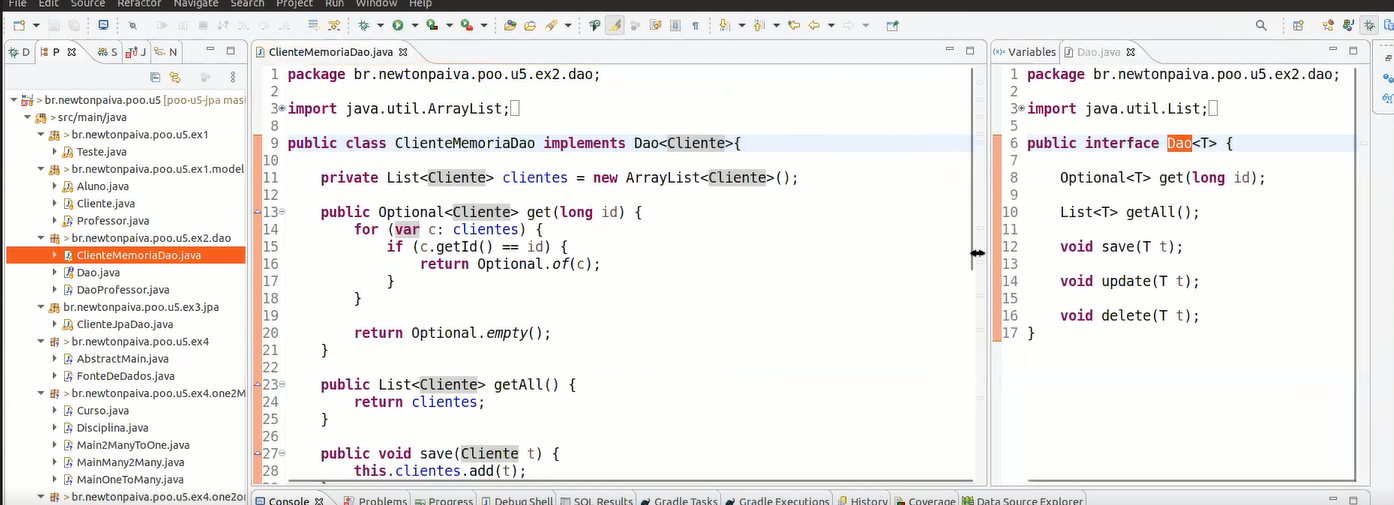
Explicação do WALDIR sobre a classe Optional

Exemplo do uso da classe Optional:



Eu tenho uma classe ClienteMemoriaDao que implementa a interface DAO<T> o T é substituído pelo Cliente DAO<Cliente> então cliente agora representa o t. O DAO do lado que é a interface java que vai generalizar as operações que todo DAO tem que ter que o meu sistema vai usar e do lado esrquedo eu tenho a implementação dessa interface.

**Como Optional funciona:**

Primeiro método get(), eu quero recuperar uma entidade no banco, a partir de um Id(Identificador), resultados possíveis desse método get quando eu tento recuperar uma entidade no banco, na real aqui estamos fazendo como se fosse o **SELECT \* FROM Cliente c Where c.id = ?(Valor que eu estou passando)**, essa é a operação que eu estou executando no banco a partir do DAO, quais o resultado desse SELECT 🡪 Achar o cliente que tem esse id e não achar o cliente(Null) retornando o resultado vazio, duas possibilidades, ou eu tenho um objeto cliente retornado no banco, ou eu tenho nada retornado NULL, então essa classe Optional ela trata erros NullPointerException. A implementação clássica sem a classe Optional seria:

Public Cliente get(long id){

For(var c : clientes){

If(c.getId() == id){

Return c;

}

}

Return null;

}

Com esse return null a gente tem um risco, desse nulo retornado causar problema no nosso sistema, ao invés da a gente retornar nullo, a gente utiliza uma classe do Java que é a Optional que um container que vai armazenar ou um objeto ou um resultado vazio, é uma alternativa pra a gente evitar ter que manipular Nulos, objetos nulos, ele trata erros NullPointerException, o Optional é para isso, encapsular o resultado de uma operação que eu posso ter um resultado bem sucedido ou eu posso não ter nada eventualmente vazio, porque o objeto não existe