Explorando Recursos Essenciais em Java para Acesso a Bancos de Dados

1. Annotations na Linguagem Java

Conceitos:

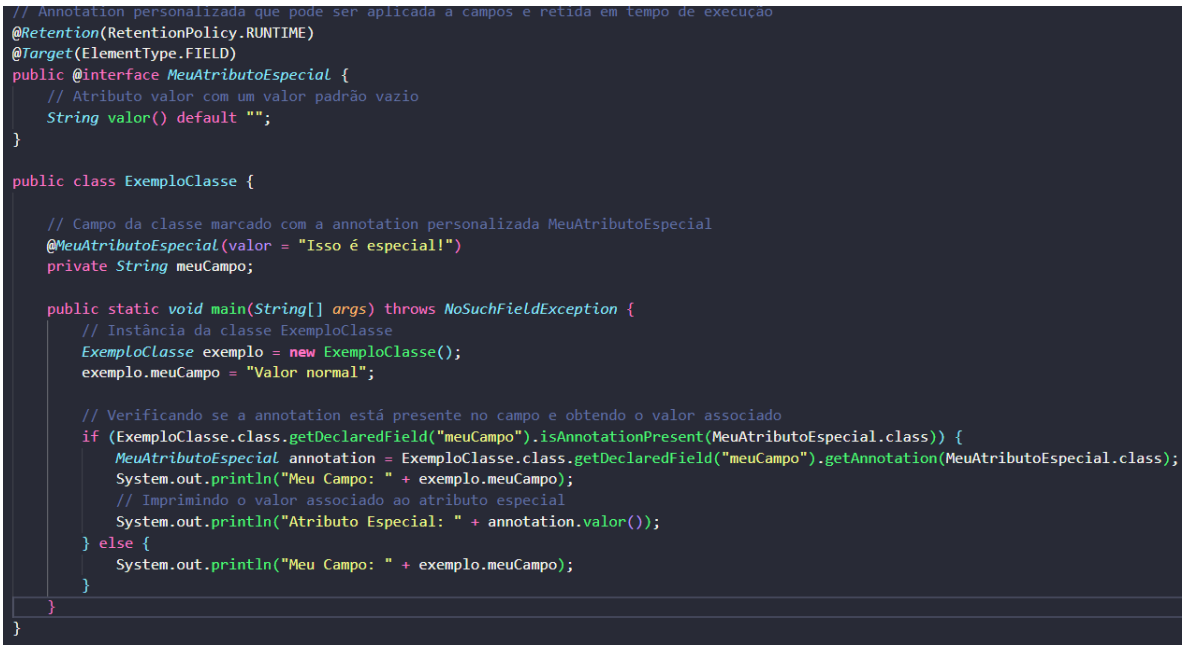
Annotations em Java são metadados incorporados ao código-fonte para fornecer informações adicionais sobre o código.Elas são marcadores especiais que podem ser aplicados a elementos como classes, métodos e campos. Esse tipo de marcação, ou pelo menos uma “versão” próxima, é oferecido por padrão pelos comentários(//) ou pelo *javaDoc*, e o annotations oferece uma maior variedade de usos e aplicações dos metadados como por exemplo o uso em RUNTIME(ver o primeiro exemplo de código).

Existem cinco tipos de annotations de metadado, sendo elas:

- @Documented: que indica que uma annotation deve ser incluída na documentação gerado pelo javadoc e semelhantes.
- @Target: indica quais elementos de código podem ser marcados com “tipo” pelo annotations.
- @Inherited: Indica que um “tipo” de annotations foi herdado.
- @Retention: Indica por quanto tempo annotations devem ser retidas, em outras palavras, é determinar em quais estágios do desenvolvimento ou execução do programa as annotations podem ser acessadas e utilizadas, lembrando que depende da políticas de retenção utilizada.
- @Repeatable: Indica que o tipo de annotations utilizado pode ser repetido.

Origem:

As annotations foram introduzidas no Java 5 para simplificar a configuração e fornecer informações suplementares a um programa e uma forma mais eficiente de incorporar metadados ao código.



```java
// Annotation personalizada que pode ser aplicada a campos e retida em tempo de execução
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface MeuAtributoEspecial {
    // Atributo valor com um valor padrão vazio
    String valor() default "";
}

public class ExemploClasse {

    // Campo da classe marcado com a annotation personalizada MeuAtributoEspecial
    @MeuAtributoEspecial(valor = "Isso é especial!")
    private String meuCampo;

    public static void main(String[] args) throws NoSuchFieldException {
        // Instância da classe ExemploClasse
        ExemploClasse exemplo = new ExemploClasse();
        exemplo.meuCampo = "Valor normal";

        // Verificando se a annotation está presente no campo e obtendo o valor associado
        if (ExemploClasse.class.getDeclaredField("meuCampo").isAnnotationPresent(MeuAtributoEspecial.class)) {
            MeuAtributoEspecial annotation = ExemploClasse.class.getDeclaredField("meuCampo").getAnnotation(MeuAtributoEspecial.class);
            System.out.println("Meu Campo: " + exemplo.meuCampo);
            // Imprimindo o valor associado ao atributo especial
            System.out.println("Atributo Especial: " + annotation.valor());
        } else {
            System.out.println("Meu Campo: " + exemplo.meuCampo);
        }
    }
}
```

Neste exemplo reduzido, a annotation MeuAtributoEspecial é aplicada ao campo meuCampo da classe ExemploClasse. O programa verifica se a annotation está presente e, se estiver, imprime o valor associado ao atributo especial(em um cenário ideal já que meu computador não os executou corretamente.

2. ORM (Object Relational Mapper) / Hibernate no Java

Conceitos:

O Hibernate é um framework de mapeamento objeto-relacional(ORM) em Java. Ele atua como uma camada de meio entre objetos Java e bancos de dados relacionais, permitindo a persistência de objetos de maneira simplificada. Permitindo assim que o Banco de dados seja escrito pelo código java, utilizando o HQL (Hibernato Query Language).

Origem:

Desenvolvido por Gavin King em 2001,provavelmente, o Hibernate tornou-se uma ferramenta amplamente adotada para simplificar o acesso a bancos de dados relacionais em aplicações Java. A empresa que fornece suporte a ferramenta foi comprada pela Red Hat.

```java
// Configuração da SessionFactory
SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();

// Abrindo uma sessão
Session session = sessionFactory.openSession();
session.beginTransaction();

// Operações de persistência
Usuario usuario = new Usuario();
usuario.setNome("John Doe");
usuario.setEmail("john.doe@example.com");
session.save(usuario);

// Commit da transação
session.getTransaction().commit();

// Fechando a sessão
session.close();
```

Neste exemplo, o Hibernate simplifica as operações de persistência, gerenciando automaticamente as transações e abstraindo(o possível) as complexidades do acesso ao banco de dados.

3. JDBC (Java Database Connectivity)

Conceitos:

O JDBC (Java EE Database Connectivity) é uma API de nível de chamada, ou seja, as instruções SQL são levadas como sequência para API executá-la,que facilita a interação com bancos de dados relacionais em Java. Ele fornece métodos para conectar-se a um banco de dados, executar consultas SQL e processar os resultados. Famosa pela “write once, run anywhere”, permitindo uma levar para diferentes sistemas sem muitos gastos com adaptação.

Origem:

O JDBC faz parte do Java desde suas versões iniciais e é uma API padrão para acesso a bancos de dados relacionais.

```java
// Estabelecendo a conexão
Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/meubanco", "usuario", "senha");

// Executando uma consulta
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery("SELECT * FROM usuarios");

// Processando os resultados
while (resultSet.next()) {
    String nome = resultSet.getString("nome");
    String email = resultSet.getString("email");
    System.out.println("Nome: " + nome + ", Email: " + email);
}

// Fechando recursos
resultSet.close();
statement.close();
connection.close();
```

```java
public class ConsultaSimplesJDBC {

    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/seu_banco_de_dados";
        String usuario = "seu_usuario";
        String senha = "sua_senha";

        try (Connection conexao = DriverManager.getConnection(url, usuario, senha)) {
            String consultaSQL = "SELECT nome, email FROM usuarios";

            try (Statement stmt = conexao.createStatement();
                 ResultSet resultado = stmt.executeQuery(consultaSQL)) {

                while (resultado.next()) {
                    String nome = resultado.getString("nome");
                    String email = resultado.getString("email");

                    System.out.println("Nome: " + nome + ", Email: " + email);
                }
            }
        }
    }
}
```

Neste exemplos, o JDBC é utilizado para estabelecer uma conexão, executar uma consulta SQL e processar os resultados obtidos do banco de dados. Lembrando também que enquanto o JDBC não é apenas para SQL, ele é projetado para trabalhar em conjunto com um bd SQL facilitando a comunicação.

Conclusão

Compreender os conceitos, origens e exemplos de código relacionados a annotations em Java, Hibernate (ORM), e JDBC oferece uma visão abrangente das opções disponíveis para interação com bancos de dados em aplicações Java. Cada tecnologia tem seus pontos fortes e a escolha entre elas dependerá das necessidades específicas do projeto e das preferências de desenvolvimento.

Bibliografia

https://www.ibm.com/br-pt

https://medium.com/@leonardogiuliani/o-que-%C3%A9-e-porque-devo-utilizar-o-hibernate-66fae865a22f

https://www.digitalocean.com/community/tutorials/java-annotations

https://medium.com/@leonardogiuliani/o-que-%C3%A9-e-porque-devo-utilizar-o-hibernate-66fae865a22f