Unidade III

5 DESIGN PATTERNS (PADRÕES DE PROJETOS)

No desenvolvimento de um sistema, espera-se que alguns requisitos sejam garantidos, por exemplo: o desempenho do programa (a eficiência e a eficácia do sistema, ou seja, que ele responda às necessidades do cliente e que funcione corretamente), a sua compreensão (a facilidade na manutenção por parte dos desenvolvedores), na utilização por parte dos usuários e na reutilização (à medida que a equipe de desenvolvimento possa fazer sem problemas uma implementação no sistema).

Saiba mais

Para aprender mais sobre design patterns, que trata entre outros assuntos acerca de padrões de projeto, leia o capítulo 9 da obra a seguir:

HORSTMANN, C. S.; CORNELL, G. Core *Java*. Volume I: fundamentos. 8. ed. São Paulo: Pearson Prentice Hall, 2010.

Os design patterns surgiram como um meio de agilizar e melhorar os processos de criação e de manutenção de sistemas a fim de facilitar o trabalho dos desenvolvedores, tornando-os mais rápidos e permitindo preços mais competitivos.

Assim, a partir das melhores práticas, reuniram-se as mais adequadas e eficazes soluções, transformando-as em padrões de desenvolvimento (design patterns). Alguns deles até se tornaram frameworks (ou seja, programas ou pacotes com bibliotecas prontas, com códigos que fornecem alguma funcionalidade ou solução específica).

Observação

É importante entendermos que design patterns são procedimentos, ou, ainda, formas e padrões de conduzir o desenvolvimento de um sistema para que haja mais agilidade e garantias de um trabalho eficaz (que seja bem feito e funcional), melhorando o trabalho do desenvolvedor, além da satisfação do cliente.

A ideia de seguir esses padrões (patterns) de desenvolvimento (de design) é conseguir trabalhar com mais certeza para produzir sistemas funcionais e manuteníveis (que permitem fácil manutenção).

Devemos lembrar que um sistema que esteja sendo construído hoje "por mim", poderá ser alterado ou implementado por outro profissional (daqui a algum tempo). Desta forma, utilizando-se de padrões, permite-se que o profissional que trabalhará sobre o "meu sistema" tenha um entendimento mais ágil, já que ele ainda não o conhece.

5.1 DTO/VO

O pattern DTO (Data Transfer Object) – ou ainda Objeto de Transferência de Dados – indica que quando vamos criar um método que receberá informações diversas em seus parâmetros, a maneira mais eficaz de fazê-la é utilizando um objeto no qual esses dados estão nos atributos do objeto.

Assim, com esse pattern, um objeto é utilizado para transferir os dados (as informações) de um local a outro (ou seja, quando um método de uma classe chama o método de outra). Isso é muito comum quando precisamos transferir dados entre a camada de visualização (view layer) e a camada de persistência dos dados (model layer) (sobre elas, estudaremos posteriormente quando virmos o pattern MVC – item 5.2).

Com o pattern DTO, um objeto (geralmente encapsulado) é preenchido com dados em seus atributos, e ele é transportado por entre as camadas de um MVC. Tal objeto é considerado um JavaBean, ou seja, uma classe encapsulada simples que representa uma entidade do BD (que pode ser tabela, view etc.).

Essa transferência de informação é na verdade a chamada de um método de uma classe a partir do método de outra (método de uma classe acionando o método de outra classe). Desta forma, tem-se uma classe com um método que inicialmente monta as informações em um objeto, preenchendo seus atributos, e que em seguida chama um método de outra classe, enviando aquele objeto como parâmetro de invocação do método. Trata-se da forma como uma classe se comunica com outra enviando ou recebendo informações.

Lembrete

Um método pode ter como parâmetro (ou ainda, receber como parâmetro) uma ou mais classes, de forma que os objetos enviados podem conter informações nos seus atributos.

Como exemplo, a classe a seguir representa uma pessoa qualquer, que contém atributos que são as informações sobre ela. Essa classe pode, por exemplo, ser utilizada em um programa de inscrição de indivíduos para um curso, de forma que cada objeto gerado em memória poderá conter os dados sobre alguém que está se inscrevendo no curso:

```
public class Pessoa {
        public String nome;
        public String cpf;
        public int idade;
        public boolean cursoPago;
```

```
        // métodos da classe Pessoa...
        //...
}

public class Curso {
        public String nome;
        public String codigo;
        public double valorMensal;

        public static void inscrever(Pessoa p1) {
        // Lógica de inscrição (enviando dados ao BD)...
        }
        // demais métodos da classe Curso...
        //...
}
```

Percebam que na classe Curso, o método inscrever tem como parâmetro um objeto da classe Pessoa (representado pelo objeto p1), de forma que ao chamá-lo, devemos já ter um objeto do tipo Pessoa com seus dados (atributos) preenchidos, os quais poderão ser utilizados para gerar o cadastro daquele aluno (que é a pessoa) no curso, para então gerar um registro no BD.

Essa forma de envio de informação é muito mais prática e fácil do que se tivéssemos que fazê-la com as informações da pessoa como se fosse um parâmetro do método de inscrição (que deveria ter muitos parâmetros).

5.2 MVC

O Pattern MVC ou, ainda, arquitetura MVC (Model-View-Controller), é também uma arquitetura, já que seu objetivo é organizar um projeto em pacotes, os quais denominaremos aqui como camadas.

Nessa arquitetura, o sistema é dividido em três camadas (ou três pacotes, em que cada um deles conterá classes relacionadas às suas características) a fim de separar as classes por responsabilidades, de forma que:

- As classes responsáveis pelas telas (montagem das interfaces com o usuário) ficam na camada de visualização (camada **view**).

- As classes responsáveis pelo processo de persistência (acesso a BD) devem ficar na camada de modelagem (camada **model**).

- Na camada de controle (camada **control**) ficam as classes responsáveis pelo controle das informações que tramitam (que são transportadas) entre as camadas de visualização e de modelagem (ou seja, entre a tela que o usuário está preenchendo e o BD). Nela, são realizadas as operações de validação, controle e encaminhamento das informações.

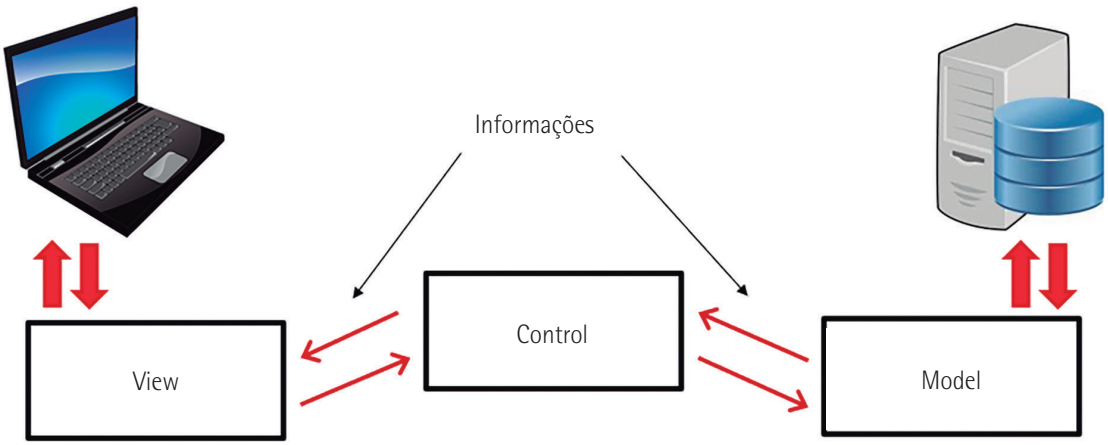

Figura 18 – Representação do fluxo de informação do design pattern MVC

Adaptada de: https://bit.ly/3bAnhrk e https://bit.ly/3P78iCH. Acesso em: 25 jul. 2022.

Cada um dos retângulos da figura anterior representa uma camada do pattern MVC. As setas indicam o fluxo das informações entre as camadas (quando um método aciona outro).

Observação

Quando dizemos que as informações são transmitidas entre os métodos, estamos apenas tratando acerca da forma como eles recebem e devolvem informações de/para quem (ou qual método) chama aquele método.

Quando um método chama outro, o primeiro deles envia informações na forma de parâmetros, os quais são recebidos e processados pelo método que está sendo chamado.

Um método devolve alguma informação a partir da definição do seu retorno, ou seja, é definido pelo tipo de retorno que o método chamado tem. No entanto, sabemos que qualquer método pode ser construído para receber e retornar classes, ou melhor, objetos (instanciados) que contêm dados em seus parâmetros.

Lembrete

É importante lembrarmos da sintaxe da declaração de um método, que é:

```
modificadores tipoRetorno nomeMetodo (parametros) {
  //...lógica de ação do método...
}
```

Assim, um método recebe dados ao ser chamado, de forma que ficam descritos nos parâmetros, devolvendo uma informação na variável que será retornada, de acordo com o tipo de retorno definido na declaração do método.

5.3 DAO (Data Access Object)

Em geral, todo e qualquer sistema necessita da persistência de alguma informação. Os DAO são os objetos que acessam dados em comunicação com o BD. Eles, por convenção geral, possuem a sigla DAO no nome da classe (por exemplo: classe AlunoDAO, ou ProdutoDAO etc.).

Segundo o pattern MVC, a camada de modelagem será a responsável pela persistência dos dados, direto do, ou para o, BD.

Assim, as classes da camada model, que ficarão responsáveis por preparar a comunicação direta com o BD, são as classes DAO. Elas ficam encarregadas por criar corretamente as queries (SQL) que o BD receberá para manipular as informações.

6 O HIBERNATE

O Hibernate é uma ferramenta de consulta ou ainda de persistência objeto/relacional de alta performance. Trata-se de uma solução ORM (de mapeamento objeto/relacional) muito flexível e poderosa, que faz o mapeamento de classes Java em relação às tabelas de um BD, e de tipos de dados Java para tipos de dados SQL. Ele fornece consultas e facilidades para o retorno dos dados que reduzem significativamente o tempo de desenvolvimento.

O Hibernate abstrai o seu código SQL de forma que toda a camada JDBC e SQL será gerada em tempo de execução. E vai além, ele gera o SQL que servirá para um determinado BD, já que cada BD utiliza um dialeto diferente dessa linguagem (possui diferenças, às vezes, significativas). Assim, há também a possibilidade de troca de BD sem precisar alterar o código Java, uma vez que isso fica como responsabilidade da própria ferramenta.

Saiba mais

Compreenda mais sobre a utilização do Hibernate acessando o texto a seguir:

APACHE NETBEANS. *Usando o Hibernate em uma Aplicação Java Swing:* tutorial do NetBeans IDE. [s.d.]. Disponível em: https://bit.ly/3BWotzI. Acesso em: 25 jul. 2022.

Usando a classe a seguir (classe Tarefa) como exemplo, vamos configurá-la (mapeá-la) para que o Hibernate consiga identificar onde e quais serão seus campos e colunas.

Inicialmente veremos a classe Tarefa sem o mapeamento (mas que representa um item de uma tabela, cujo nome é Tarefa, no BD).

```
public class Tarefa {

        private Long id;
        private String descricao;
        private boolean finalizado;
        private Calendar dataFinalizacao;
        // Métodos Setters e Getters
        // ...
}
```

A classe anterior é encapsulada como qualquer outra que já aprendemos a escrever em Java. Precisamos então configurá-la (mapeá-la) para utilizar o Hibernate, para que ele saiba da existência dessa classe e, desta forma, entenda que deve inserir uma linha (um dado) na tabela Tarefa, toda vez que for requisitado, fazendo com que um objeto desse tipo seja salvo.

Para mapear a classe Tarefa, basta adicionar algumas poucas anotações em nosso código. Anotação é um recurso do Java que permite inserir metadados em relação a nossa classe, atributos e métodos. Essas anotações depois poderão ser lidas por frameworks e bibliotecas (como o Hibernate, por exemplo), para que eles tomem decisões baseadas nessas pequenas configurações.

Mapeando a classe Tarefa:

```
@Entity
public class Tarefa {

        @Id
        @GeneratedValue
        private Long id;

        private String descricao;
        private boolean finalizado;

        @Temporal(TemporalType.DATE)
```

```
        private Calendar dataFinalizacao;

        // Métodos Setters e Getters
        // ...
}
```

- A anotação @Entity indica que os objetos dessa classe se tornarão persistíveis no BD.
- A anotação @Id indica que o atributo id será nossa chave primária (é preciso ter uma chave primária em toda entidade).
- A anotação @GeneratedValue indica que queremos que essa chave seja populada e criada pelo banco (isto é, ela será utilizada como um auto increment, ou sequence, dependendo do BD).
- A anotação @Temporal configura um mapeamento do tipo Calendar para o BD, de forma que aqui usamos apenas como data (sem a hora), mas poderíamos fazê-lo também como apenas hora (TemporalType.TIME) ou ainda como timestamp (TemporalType.TIMESTAMP), dependendo da necessidade.

Observação

A anotação @Id não recebe esse nome porque o nome do atributo é id, o que significa que, apesar de esse nome representar a chave primária, muitas vezes ele pode ser: id, cod, num, cp, pk etc. Independentenmente disso, o nome da anotação será sempre @Id.

Essas anotações precisam dos devidos imports, que pertencem ao pacote javax.persistence. Mas em que tabela essa classe será gravada? Em quais colunas? Que tipo de coluna?

Na ausência de configurações mais específicas, o Hibernate vai usar convenções: a classe Tarefa será gravada na tabela de nome também Tarefa, e o atributo descrição em uma coluna de nome descricao.

Se quisermos configurações diferentes das convenções, basta usarmos outras anotações, que são completamente opcionais.

Por exemplo, para mapear o atributo dataFinalizacao em uma coluna chamada data_finalizado:

```
"...
@Column(name="data_finalizado", nullable=true)
private Calendar dataFinalizacao;
..."
```

Para usar uma tabela com o nome tarefas:

```
"...
@Entity
@Table(name="tarefas")
public class Tarefa {
..."
```

Repare que nas entidades há todas as informações sobre as tabelas. Baseado nelas, podemos até pedir para gerar as tabelas no banco, mas para isso é preciso configurar o JPA.

Configurando as propriedades do banco

Em qual BD vamos gravar nossas tarefas? Qual é o login? Qual é a senha?

O JPA necessita dessas configurações, de forma que para isso criaremos o arquivo persistence.xml.

Com o objetivo de configurar a conexão com o BD, o Hibernate precisa saber como se conectar a ele, e isso é feito através do arquivo persistence.xml, que deve ficar dentro de uma pasta nomeada como META-INF, localizada na raiz do projeto (criada enquanto se está programando o sistema).

Para este nosso sistema, precisamos de quatro linhas com configurações que já conhecemos do JDBC: a string de conexão com o banco, o driver, o usuário e a senha que permitem acessar o BD. Além disso, temos de dizer qual dialeto SQL deverá ser utilizado no momento em que as queries forem geradas automaticamente pelo Hibernate.

O exemplo a seguir mostra a configuração caso utilizemos o MySQL como SGBD. Ele está no formato xml (arquivo .xml).

```
“...
<?xml version=”1.0” encoding=”UTF-8”?>
<persistence version=”2.0”
        xmlns=”http://java.sun.com/xml/ns/persistence”
        xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
        xsi:schemaLocation=”http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd” >

                <persistence-unit name=”hibAlunos”>
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
                        <class>Aluno</class>
                        <properties>
                                <property name=”javax.persistence.jdbc.driver”
value=”com.mysql.cj.jdbc.Driver” />
                                <property name=”javax.persistence.jdbc.url”
value=”jdbc:mysql://localhost:3306/testehiber?useTimezone=true&amp;serverTimezon
e=UTC” />
                                <property name=”javax.persistence.jdbc.user”
value=”root” />
                                <property name=”javax.persistence.jdbc.password”
value=”1234” />

                                <property name=”hibernate.dialect” value=”org.
hibernate.dialect.MySQL8Dialect” />
                                <property name=”hibernate.max_fetch_depth”
value=”3” />

                                <property name=”hibernate.hbm2ddl.auto”
value=”update” />
```

```
                                <property name=”hibernate.show_sql” value=”true”
/>
                        </properties>
                </persistence-unit>
        </persistence>
...”
```

Os itens de maior atenção desse arquivo são:

- **<persistence-unit name="nome">**: configura o nome do contexto que será utilizado pela aplicação.

As propriedades a seguir definem as configurações de acesso ao BD

- **javax.persistence.jdbc.driver**: nome completo da classe do driver JDBC do BD
- **javax.persistence.jdbc.user**: nome do usuário que será utilizado para estabelecer a conexão com o BD.
- **javax.persistence.jdbc.password**: senha do usuário.
- **javax.persistence.jdbc.url**: string de conexão com o BD.
- **hibernate.dialect**: configura o dialeto que o Hibernate utilizará para a montagem dos comandos SQL.

Comunicando-se com o banco

O primeiro passo é fazer com que o JPA leia a nossa configuração, tanto o arquivo persistence.xml quanto as anotações que colocamos na entidade Tarefa. Para tal, usaremos a classe com o mesmo nome do arquivo XML: Persistence. Ela é a responsável por carregar o XML e inicializar as configurações. O resultado dessa configuração é uma EntityManagerFactory:

```
EntityManagerFactory factory = Persistence.createEntityManagerFactory(“taref
as”);
```

Antes de gravar uma tarefa, precisamos que exista uma tabela correspondente no nosso BD. Em vez de criarmos o script que define o schema (ou DDL – data definition language – de um banco) do nosso banco (os famosos CREATE TABLE ....), podemos deixar isso a cargo do próprio Hibernate. Ao inicializarmos a EntityManagerFactory também será gerada uma tabela Tarefas, pois configuramos que o banco deve ser atualizado pela propriedade do Hibernate: hbm2ddl.auto.

6.1 Persistindo dados com o Hibernate

Persistindo objetos

Para facilitar a criação de objetos EntityManager, foi construída a classe EntityManagerProvider, que encapsula todo o trabalho necessário à criação de tais objetos, classe que está demonstrada na sequência:

```
"...
package hiber;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class EntityManagerProvider {
        private static EntityManagerFactory emf = null;
        private EntityManagerProvider()         {

        }

        public static EntityManager getEntityManagerInstance() {
                if (emf == null) {
                        emf = Persistence.createEntityManagerFactory("hibAlun
os");
                }
                return emf.createEntityManager();
        }

}
..."
```

O parâmetro utilizado na criação do objeto tem o valor artigo (ou tarefa). Esse é exatamente o nome definido na propriedade persistence-unit do arquivo persistence.xml, ou seja, é através deste parâmetro que o Hibernate entende em que BD deve estabelecer a conexão, e com quais parâmetros.

A classe EntityManager disponibiliza métodos pelos quais os objetos podem ser manipulados.

- **Método remove**: exclui do BD o registro na tabela referente ao objeto passado como parâmetro.
- **Método createQuery**: permite a consulta de dados persistidos no BD, recuperando-os na forma de objetos.
- **Método persist**: envia efetivamente os dados do objeto criado ou alterado para o BD.
- **Método find**: realiza a busca por um objeto através de seu identificador.
- **Método getTransaction**: obtém a referência de um objeto do tipo EntityTransaction, o qual permite o controle das transações.

- **Método begin**: indica o início de um bloco que deve ser controlado em um contexto transacional.

- **Método commit**: possibilita efetivar todas as ações executadas no BD, desde a execução do método begin.

- **Método rollback**: propicia desfazer todas as ações executadas no BD, desde a execução do método begin.

Apesar de a classe EntityManager possuir métodos bem definidos para a manipulação dos objetos, ainda existe a necessidade de iniciar a transação, executar a ação desejada sobre o objeto e encerrar a transação, confirmando ou cancelando as ações.

Resumo

Nesta unidade, vimos que os padrões de desenvolvimento, mais conhecidos como design patterns, surgiram das melhores práticas realizadas ao longo do desenvolvimento (criação ou implementação) em sistemas de software à medida que se identificava um processo que aumentava muito a eficiência da codificação. Além da eficácia do produto gerado, esse processo e todo o conhecimento adquirido em torno dele tornaram-se um padrão de desenvolvimento a ser seguido pelos desenvolvedores da linguagem.

Entendemos que alguns padrões estão ligados à forma básica de se codificar (como o DTO, que é um padrão que define como as informações devem ser enviadas através das classes ao acionarmos seus métodos). Outros padrões, como o MVC e o DAO, definem qual a melhor maneira de organizar um sistema quando o projeto prevê uma interação entre o usuário e um BD, de forma que o sistema que permitirá tal interação deverá ser criado em camadas, e cada uma delas terá uma finalidade no sistema.

Por fim, demonstramos a existência de padrões na codificação básica e na construção (na arquitetura) do projeto, sendo que em alguns casos um determinado padrão é tão específico que se transforma em um sistema à parte (framework).

Nesse sentido, criou-se um framework conhecido como Hibernate, que é um sistema instalável, gerado, e muito utilizado, para facilitar e padronizar a persistência de informações em BD. Para utilizarmos o framework, precisamos instalá-lo, configurá-lo no projeto, e mapear as classes que representarão as entidades dos bancos de dados e que conterão as informações trazidas pelo ou a serem levadas ao BD.

Exercícios

Questão 1. (IADES/2018, adaptada) A sigla do padrão de projeto MVC vem do inglês Model-View-Controller. A respeito dele, avalie as afirmativas a seguir:

I – A camada model é responsável pela exibição de dados na tela.

II – A camada view é responsável pelas regras de negócio.

III – A comunicação entre as interfaces e as regras de negócio é responsabilidade da camada controller.

É correto o que se afirma em:

A) I, apenas.

B) III, apenas.

C) I e II, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: alternativa B.

Análise das afirmativas

I – Afirmativa incorreta.

Justificativa: o MVC é um padrão de arquitetura de software focado na separação de conceitos em três camadas interconectadas. Enfim, nela, o sistema é dividido em três camadas, sendo que cada uma delas conterá classes relacionadas às suas características, a fim de separá-las por responsabilidades. A camada model, responsável pela modelagem da aplicação, gerencia o comportamento dos dados por meio de regras de negócios, de regras lógicas e de funções. Dessa forma, ela modela o problema que se pretende resolver com a aplicação. As classes responsáveis pelo processo de persistência (acesso ao BD) devem ficar na camada model.

II – Afirmativa incorreta.

Justificativa: a camada view, de visualização, é responsável pela exibição dos dados oriundos do model. Portanto, ela é utilizada para apresentar as informações ao usuário, utilizando combinações

de gráficos e textos. As classes responsáveis pelas telas (montagem das interfaces com o usuário) ficam na camada view.

III – Afirmativa correta.

Justificativa: a camada controller tem foco na interação do usuário, sendo responsável pela mediação entre as interfaces com o usuário e as regras de negócio. Ela interpreta as ações recebidas pelos periféricos de entrada (como teclado ou mouse), vindas do usuário, e mapeia essas ações, que são enviadas para o model ou para o view. Aqui, ficam as classes responsáveis pelo controle das informações transportadas entre as camadas view e model.

Questão 2. (CCV-UFC/2019) Leia o texto a seguir, a respeito da ferramenta Hibernate.

Com a popularização do Java em ambientes corporativos, logo se percebeu que grande parte do tempo do desenvolvedor era gasto na codificação de queries SQL e no respectivo código JDBC responsável por trabalhar com elas.

Além de um problema de produtividade, algumas outras preocupações aparecem: SQL, apesar de ter um padrão ANSI, apresenta diferenças significativas dependendo do fabricante. Não é simples trocar um banco de dados por outro. Há ainda a mudança do paradigma. A programação orientada a objetos difere muito do esquema entidade relacional, e precisamos pensar das duas maneiras para fazer um único sistema. Para representarmos as informações no banco, utilizamos tabelas e colunas. As tabelas geralmente possuem chave primária e podem ser relacionadas por meio da criação de chaves estrangeiras em outras tabelas. Quando trabalhamos com uma aplicação Java, seguimos o paradigma orientado a objetos, onde representamos nossas informações por meio de classes e atributos. Ademais, podemos utilizar herança, composição para relacionar atributos, polimorfismo, enumerações, entre outros. Essa diferença entre esses dois paradigmas gera bastante trabalho: a todo momento, devemos transformar objetos em registros e registros em objetos.

Ferramentas para auxiliar nesta tarefa tornaram-se populares entre os desenvolvedores Java e são conhecidas como ferramentas de mapeamento objeto-relacional (ORM). O Hibernate é uma ferramenta ORM open source e é a líder de mercado, sendo a inspiração para a especificação Java Persistence API (JPA). Ele nasceu sem JPA, mas, hoje em dia, é comum acessar o Hibernate pela especificação JPA. Como toda especificação, ela deve possuir implementações. Entre as implementações mais comuns, podemos citar: Hibernate da Red Hat, EclipseLink da Eclipse Foundation e o OpenJPA da Apache.

Apesar de o Hibernate ter originado a JPA, o EclipseLink é a implementação referencial. O Hibernate abstrai o seu código SQL. Assim, toda a camada JDBC e o SQL serão gerados em tempo de execução. Mais que isso, ele vai gerar o SQL que serve para um determinado banco de dados, já que cada banco fala um dialeto diferente dessa linguagem. Assim, há também a possibilidade de trocar de banco de dados sem ter de alterar o código Java, já que isso fica como responsabilidade da ferramenta.

Adaptada de: https://bit.ly/3zxAqta. Acesso em: 1º jul. 2022.

Sobre o framework Hibernate, avalie as afirmativas a seguir.

I – O Hibernate é uma solução tecnológica para o mapeamento objeto-relacional (ORM) que aceita o uso da Java Persistence API (JPA).

II – Devido à utilização do Hibernate, o Java Database Connectivity (JDBC) não é mais utilizado nas aplicações, tendo sido descontinuado.

III – O objetivo do Hibernate é fornecer uma forma de mapeamento dos objetos Java para o banco de dados apenas para fins de consulta das informações.

IV – A Java Persistence API (JPA) substitui todas as funções disponibilizadas pelo Hibernate, tornando-o desnecessário e obsoleto.

É correto apenas o que se afirma em:

A) I.

B) III.

C) I e II.

D) I e IV.

E) II, III e IV.

Resposta correta: alternativa A.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: de acordo com o texto do enunciado, o Hibernate é uma ferramenta ORM open source, que inspirou a especificação JPA. Hoje, é comum o utilizarmos por meio da JPA. Logo, é correto afirmar que o Hibernate é uma solução para o mapeamento objeto-relacional que aceita o uso da Java Persistence API (JPA).

II – Afirmativa incorreta.

Justificativa: o JDBC é uma API de conexão com um BD, sendo a API o padrão que aplicações Java utilizam para interagir com um BD. O Hibernate, por sua vez, é um framework para o mapeamento objeto-relacional, que utiliza o JDBC para interagir com um BD. Desse modo, não é correto dizer que o JDBC foi descontinuado devido à utilização do Hibernate.

III – Afirmativa incorreta.

Justificativa: o Hibernate fornece uma forma de mapeamento dos objetos Java para o BD. No entanto, sua utilização não se restringe a fins de consulta de informações em um BD. Assim, ele oferece também métodos de inclusão, exclusão ou atualização de registros, por exemplo.

IV – Afirmativa incorreta.

Justificativa: a Java Persistence API (JPA) é uma API padrão da linguagem Java que descreve uma interface comum para frameworks de persistência de dados. Vários frameworks de mapeamento objeto-relacional, como o Hibernate, implementam a JPA. Logo, ela não substitui as funções disponibilizadas pelo Hibernate, mas complementa sua funcionalidade.