Aula 4

Desenvolvimento Web – Back-End

Prof.ª Luciane Yanase Hirabara Kanashiro

1

Conversa Inicial

2

- **Nesta aula, resgataremos alguns conceitos fundamentais e estudaremos sobre persistência e manipulação de dados**

3

- **Classe Java Revisitada**
- **Persistência de Dados – JPA e ORM**
- **Persistência de Dados – ciclo de vida e DAO**
- **Camada de modelo – classe de serviço**
- **Gerenciamento de transações**

4

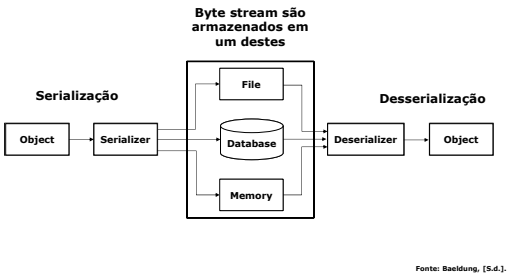
Classe Java revisitada: serializable

5

Serialização

- **Converte objeto em sequência de bytes**
- **Armazenamento persistente**
- **Transmissão de dados pela rede**
- **Reconstrução do objeto em ambiente diferente**

6



Fonte: Baeldung, [S.d.].

7

Serialização

- **Serializable**
  - **Interface vazia**
  - **Implementação: suficiente para sinalizar a capacidade de serialização**

```
import java.io.Serializable;

public class MinhaClasse implements Serializable {
    // Código da classe
}
```

8

Parametrização

- **Parâmetro genérico: uso de tipos genéricos**
- **Classes, interfaces e métodos: diferentes tipos de dados**
- **Segurança de tipo**
- **Vantagem: código flexível e reutilizável**

9

Exemplo

```
public class Caixa<T> {
    private T conteudo;

    public void adicionarConteudo(T novoConteudo) {
        this.conteudo = novoConteudo;
    }

    public T obterConteudo() {
        return this.conteudo;
    }
```

10

```
public static void main(String[] args) {
        // Uso da classe Caixa com um tipo específico (Integer)
        Caixa<Integer> caixaDeInteiro = new Caixa<>();
        caixaDeInteiro.adicionarConteudo(42);
        Integer valorInteiro = caixaDeInteiro.obterConteudo();
        System.out.println("Conteúdo da Caixa de Inteiro: " +
                valorInteiro);

        // Uso da classe Caixa com um tipo diferente (String)
        Caixa<String> caixaDeString = new Caixa<>();
        caixaDeString.adicionarConteudo("Olá, mundo!");
        String valorString = caixaDeString.obterConteudo();
        System.out.println("Conteúdo da Caixa de String: " +
                valorString);
    }
}
```

11

Persistência de dados – JPA e ORM

12

JPA

- **Java Persistence API**
- **Facilita o gerenciamento de persistência e ORM**
- **Construção: conceito de POJO**

13

ORM (Object Relational Mapping)

⇧ **SOLUÇÃO**

"Incompatibilidade de impedância objeto-relacional' ou 'incompatibilidade de paradigma'"

14

Entidades e anotações

- **Entidades no JPA: POJOs**
- **POJOs: dados que podem ser persistidos no BD**
- **Entidade: tabela armazenada em BD**
- **Instância: linha na tabela**

```
public class Funcionario{

    private Long id;
    private String name;
    private Integer idade;

//getters and setters
}
```

15

Entidades e anotações

Anotação	Significado
@Entity	Define entidade
@Id	Define a chave primária
@GeneratedValue	Geração dos identificadores
@Table	Nomear uma tabela
@Column	Nomear e detalhar uma coluna da tabela
@Transient	Quando não se quer que um campo seja persistido na tabela

16

Exemplo

```
@Entity
@Table(name="FUNCIONARIO")
public class Funcionario {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    @Column(name=" NOME_FUNCIONARIO ", length=50, nullable=false)
    private String nome;

    @Transient
    private Integer idade;

    // other fields, getters and setters
}
```

17

Relacionamentos e anotações

- **Classe pode ser mapeada como tabela do BD**
- **Realizado de acordo com a direção do relacionamento e cardinalidade**

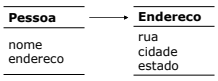
Relacionamento	Anotação
Um para um	@OneToOne
Muitos para um	@ManyToOne
Um para muitos	@OneToMany
Muitos para muitos	@ManyToMany

18

Unidirecional

```
@Entity
public class Pessoa {
private String nome;
private Endereco endereco;
    @OneToOne(cascade={cascadeType.ALL})

    @JoinColumn(name="ENDERECO_ID_fk")
    public Endereco getEndereco(){
        return endereco;
    }
}
```




```
@Entity
public class Endereco{
 private String rua;
 private String cidade;
 private String estado;
}
```

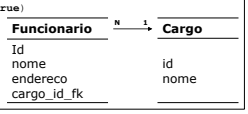
19

Bidirecional

```
@Entity
@Table(name = "FUNCIONARIO")
public class Funcionario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String nome;
    @Column(nullable = false, unique = true)
    private String endereco;

@ManyToOne
@JoinColumn(name = "cargo_id_fk")
    private Cargo cargo;
}
```



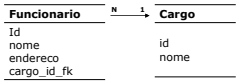
20

Bidirecional

```
@Entity
public class Cargo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nome", nullable = false, unique = true, length = 60)
    private String nome;

    @OneToMany(mappedBy = "cargo")
    //@JsonIgnore
    private List<Funcionario> funcionarios;
}
```



21

Persistência de dados – ciclo de vida de classes persistentes e padrão DAO

22

Classes persistentes

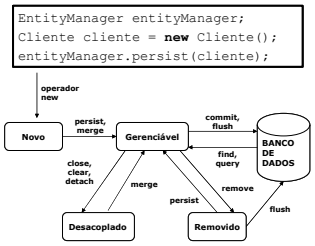
- **Classes Java**
- **Associadas à camada de persistência de dados**
- **Entidades de negócios**
- **Armazenadas e recuperadas de BD**

23

Ciclo de vida

- **Instância classe: linha na tabela**

```
EntityManager entityManager;
Cliente cliente = new Cliente();
entityManager.persist(cliente);
```



Fonte: Elaborado com base em Cordeiro, 2012.

24

Padrão DAO

- **Problema: código intimamente ligado aos detalhes específicos dos recursos de dados que utiliza**

25

Problema

- **Manutenção**
- **Atualização**
- **Substituição**

} **complicada e propensa a erros**

- **Boa prática: separação da lógica de negócios da lógica de acesso**

26

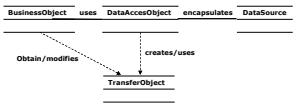
Padrão DAO

- **Padrão de projeto**
- **DAO (Data Access Object)**
- **Separação de responsabilidades**
- **Abstração de acesso a dados**

27

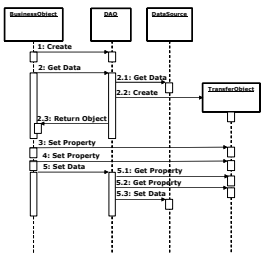
Padrão DAO

- **BusinessObject: cliente de dados**
- **DAO: implementação de acesso a dados**
- **DataSource: fonte de dados**
- **TransferObject: Objeto de transferência**



28

Padrão DAO



29

- **Encapsula o acesso e a manipulação de dados em uma camada separada**

```
public interface FuncionarioDao {

   void save(Funcionario funcionario);
   void update(Funcionario funcionario);
   void delete(Long id);
   Funcionario findById(Long id);
   List<Funcionario> findAll();

}
```

30

Camada de modelo – classe de serviço

31

Classe de serviço

- **Separa**
  - **Regras de negócio**
  - **Regras da aplicação**
  - **Regras de apresentação**

Podem ser testadas e reutilizadas

32

Classe de serviço

- **Anotação @Service**
- **Classe facilitadora: persistência > Models**
- **Controller: enxerga a Service e o repository fica encapsulado nela**

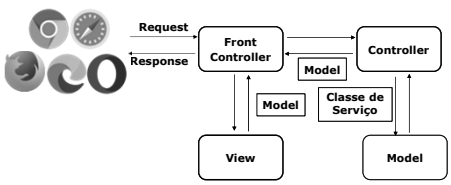
33

Camadas e tarefas

Modelo	• Modelo ou domínio da aplicação • Regras de negócios • Persistência de dados
Visão	• Interface gráfica • Interação usuário • Entrada/saída
Controlador	• Recebe requisições • Interage com o modelo e retorna a requisição ao usuário

34

Spring MVC



35

Implementação de classe de serviço

```
@Service
public class ProductService {
  @Autowired
  private ProductRepository productRepository;
  public List<Product> getAllProducts() {
    return productRepository.findAll();
  }
  public Product getProductById(Long productId){
    return productRepository.findById(productId).orElse(null);
}
 public void saveProduct(Product product) {
    productRepository.save(product);
}
 public void deleteProduct(Long productId) {
    productRepository.deleteById(productId);
} }
```

36

Gerenciamento de transações

37

Gerenciamento de transações

- Garante a consistência e a integridade dos dados ao realizar operações que podem afetar várias partes do sistema

38

Princípios

- Garantia do sucesso das operações
- Isolamento entre transações e efeitos no banco de dados

39

Transação de transferência bancária

Se, por algum motivo, uma parte da transação falhar, é imperativo que a transação seja revertida para garantir que o sistema permaneça em um estado consistente

Robodeski/shutterstock

40

Transação

- Unidade lógica de trabalho que é executada como uma operação indivisível e atômica
- Forma declarativa: @Transactional

41

Implementação

- Classe DAO
- Classe de serviço
  - Mais segurança nas transações, pois evita problemas relacionados a rollback

42

Transação na classe DAO

```
@Transactional
public class PalavraDaoImpl implements
PalavraDao {
void insert(Palavrs palavra);}
```

```
@Transactional
public class ExpresssaoDaoImpl implements
ExpressaoDao {
void insert(Expressao expressao);}
```

```
@Service
public class ExpressaoServiceImpl implements ExpressaoService {
    @Autowired
    private PalavraDao palavraDao;
    @Autowired
    private ExpressaoDao expressaoDao;
    void insert(PalavraDao palavraDao,Expressao expressao){
        palavraDao.insert(palavra);
        expressao.setPalavra(palavra);
        expressaoDao.insert(expressao);
}
```

43

Transação na classe de serviço

```
public class PalavraDaoImpl implements
PalavraDao {
void insert(Palavra palavra);}
```

```
public class ExpressaoDaoImpl implements
ExpressaoDao {
void insert(Expressao expressao); }
```

```
@Service
@Transactional(readOnly=false)
public class ExpressaoServiceImpl implements ExpressaoService {
@Autowired
private PalavraDao palavraDao;
@Autowired
private ExpressaoDao expressaoDao;
void insert(PalavraDao palavraDao ,Expressao expressao){
palavraDao.insert(palavra);
expressao.setPalavra(palavra);
expressaoDao.insert(expressao);
}
```

44