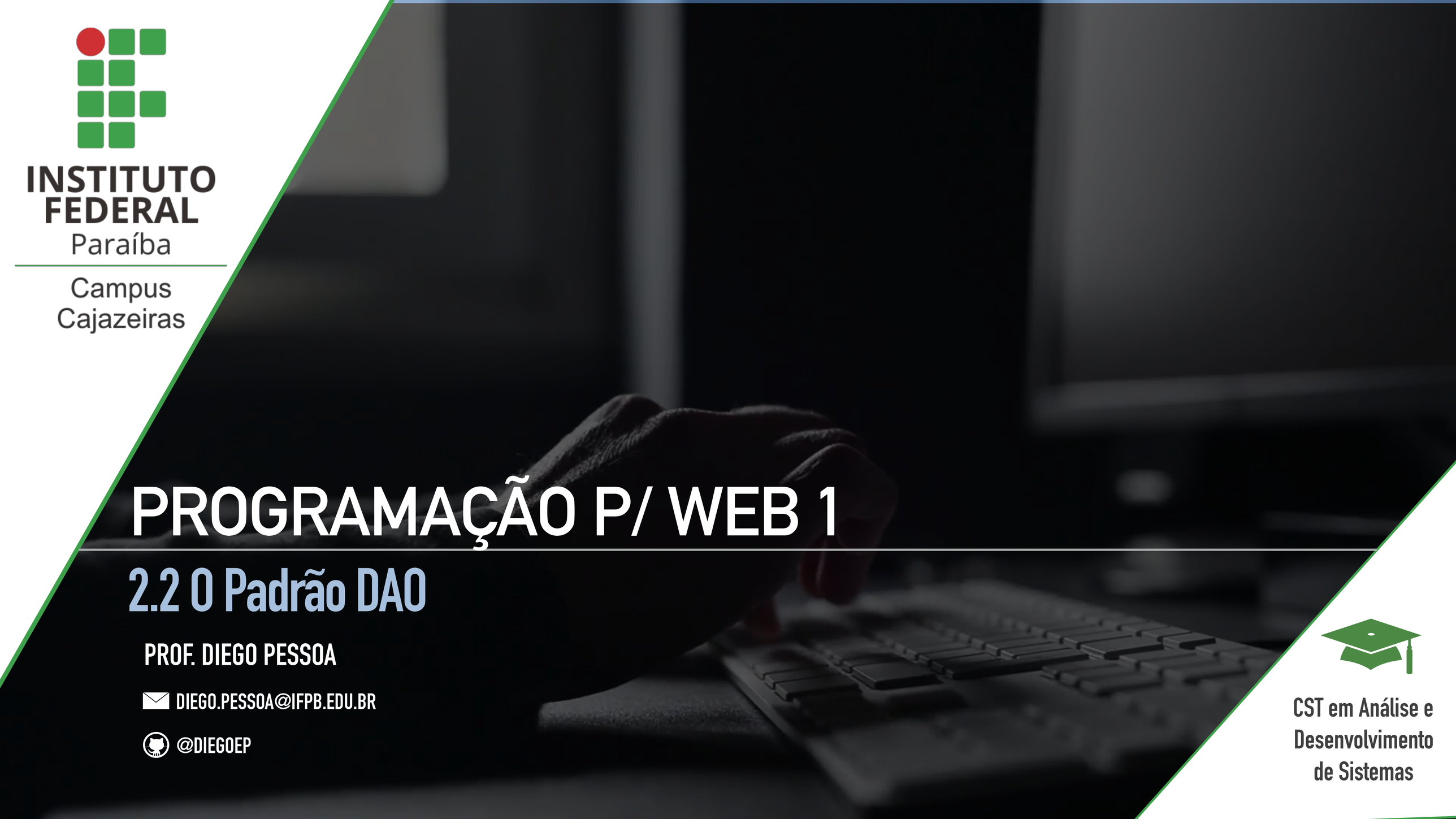


INSTITUTO FEDERAL
Paraíba

Campus Cajazeiras

PROGRAMAÇÃO P/ WEB 1

2.2 O Padrão DAO

PROF. DIEGO PESSOA

DIEGO.PESSOA@IFPB.EDU.BR

@DIEGOEP

CST em Análise e Desenvolvimento de Sistemas

Introdução

- Muitas aplicações precisam manipular dados persistentes em algum momento;
- Atualmente, eta persistência pode ser implementada de várias formas:
  - Bancos de dados relacionais, bancos de dados NoSQL, planilhas, arquivos, serviços, etc;

Introdução

- Cada mecanismo de armazenamento:
  - Possui diferentes restrições de acesso;
  - Requer o uso de APIs específicas;
  - Possui uma estratégia diferente armazenamento/recuperação dos dados;

Introdução

- A forma como o acesso à fonte de dados é implementado pode deixar este código muito acoplado ao código das regras de negócio;
- O padrão DAO é uma solução para evitar este tipo de situação.

Exemplo Motivacional

- Suponha que uma uma empresa está desenvolvendo uma aplicação com três módulos principais:
  - Controle de empregados, controle de departamentos e controle de projetos;

Exemplo Motivacional

- Estrutura da classe responsável pelo controle de empregados:

```
public class GerenciadorDeEmpregado{

        public GerenciadorDeEmpregado(){...}

        public void adicionaEmpregado(String matricula, String nome, float salario,
                Departamento departamento, Empregado supervisor) throws PersistenciaException{...}

        public void removeEmpregado(String matricula) throws PersistenciaException{...}

        public List<Empregado> lista() throws PersistenciaException{...}

        private Empregado leEmpregado(ResultSet rs) throws SQLException  {...}

}
```

Implementação do método que adiciona um novo empregado:

```
public void adicionaEmpregado(String matricula, String nome, float salario,
        Departamento departamento, Empregado supervisor) throws PersistenciaException{
        Empregado novoEmp = new Empregado();
        novoEmp.setMatricula(matricula);
        novoEmp.setNome(nome);
        novoEmp.setSalario(salario);
        novoEmp.setDepartamento(departamento);
        novoEmp.setSupervisor(supervisor);

        String sql = "INSERT INTO EMPREGADO (Matricula, Nome, Salario, Supervisor, CodDepartamento)" +
                " VALUES (?, ?, ?, ?, ?) ";

        try{
              Connection connection = GerenciadorBD.getInstance().getConnection();
              PreparedStatement statement = connection.prepareStatement(sql);
              statement.setString(1, novoEmp.getMatricula());
              statement.setString(2, novoEmp.getNome());
              statement.setFloat(3, novoEmp.getSalario());
              statement.setString(4, novoEmp.getSupervisor().getSupervisor().getMatricula());
              statement.setInt(5, novoEmp.getDepartamento().getCodigo());
              statement.execute();
              statement.close();
        }
        catch(Exception e){
            throw new PersistenciaException(e);
        }
}
```

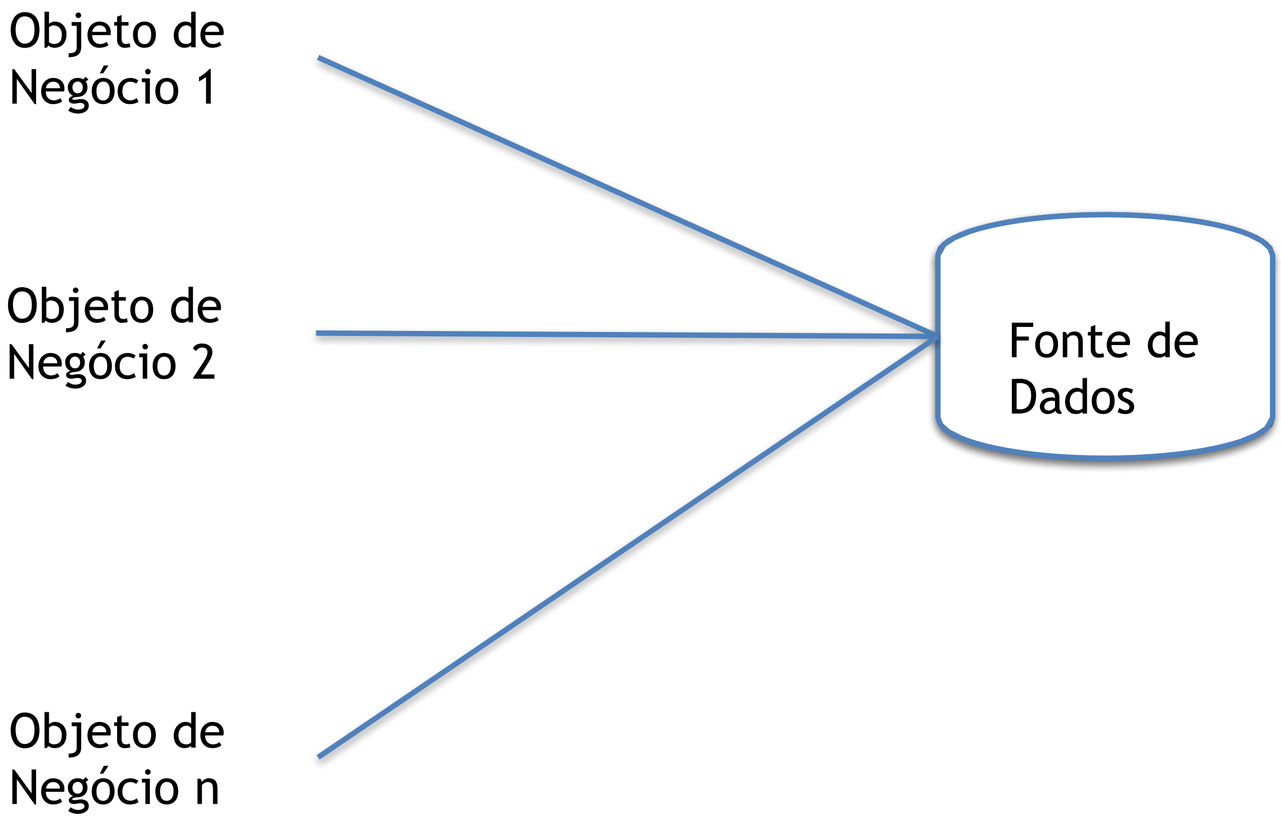
- Implementação do método que remove um empregado:

```
public void removeEmpregado(String matricula) throws PersistenciaException{
        String sql = "DELETE FROM Empregado WHERE Matricula = ? ";
        try{
              Connection connection = GerenciadorBD.getInstance().getConnection();
              PreparedStatement statement = connection.prepareStatement(sql);
              statement.setString(1, matricula);
              statement.execute();
              statement.close();
        }
        catch(Exception e){
            throw new PersistenciaException(e);}
}
```

Exemplo Motivacional

- Problema:
- A classe, que é um objeto de negócio, sabe muitos detalhes sobre a persistência dos dados;
  - Sabe que os dados estão sendo armazenados em um banco de dados relacional;
  - Conhece o esquema lógico usado para o armazenamento dos dados no banco de dados;
- Este fato gera uma grande dependência (forte acoplamento) entre as classes;
- Este acoplamento dificulta a expansão e a manutenção do sistema;

- Na prática, temos um cenário parecido com este:



Exemplo Motivacional

- Imagine o que vai acontecer:
  - Se o esquema do banco de dados for alterado;
  - Se a empresa decidir migrar para outra estratégia de armazenamento dos dados;

Exemplo Motivacional

- Resultado:
  - Código pouco flexível;
  - O programador pode ter que alterar todas as classes que interagem com o banco de dados;
  - Nestas classes, o programador pode ter que alterar todos os métodos que realizam acesso ao banco de dados;

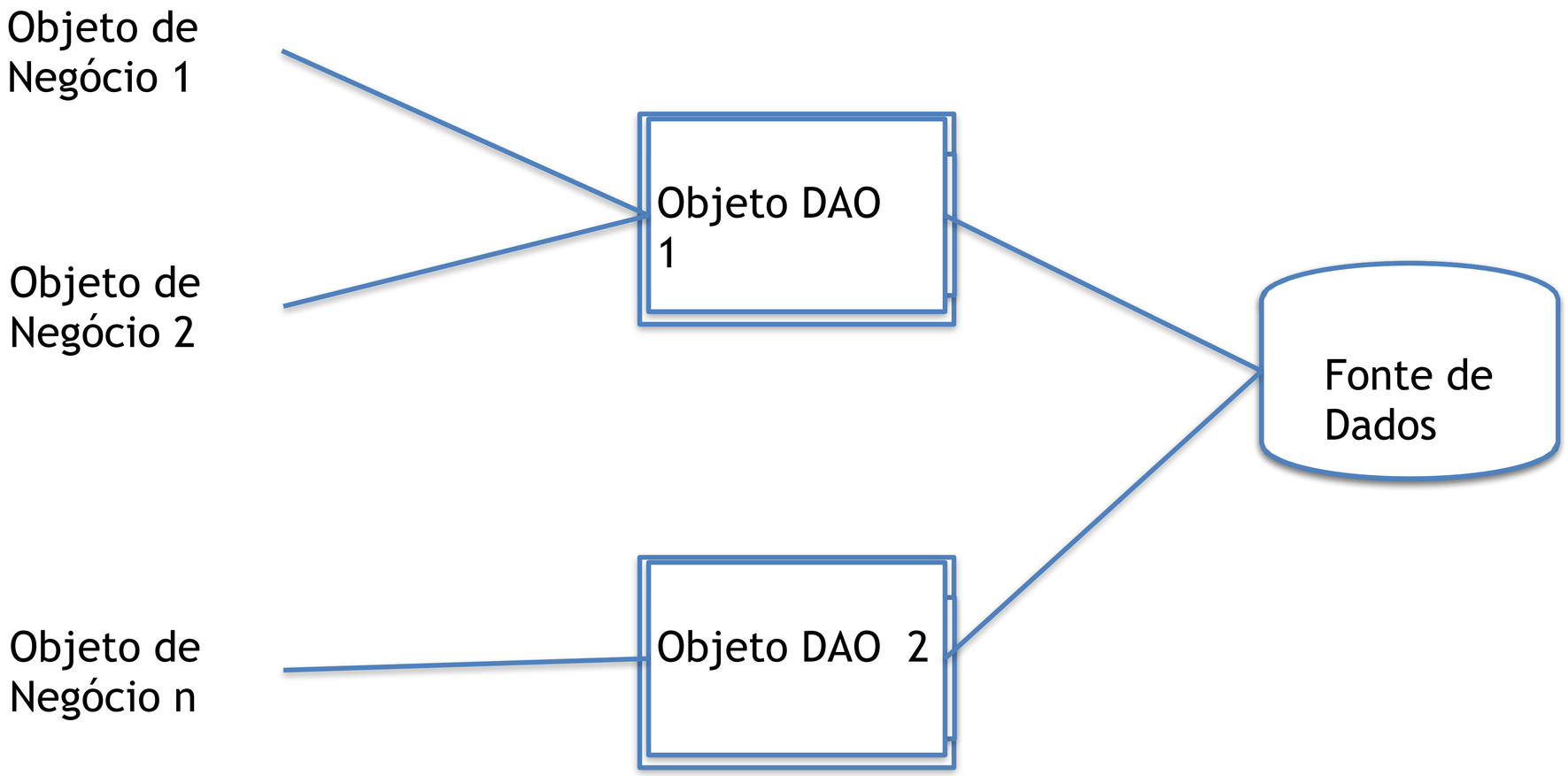
Exemplo Motivacional

- Para resolver esta limitação, devemos esconder dos objetos de negócio os detalhes sobre o armazenamento dos dados;
- Podemos fazer isso usando o padrão de projeto DAO;

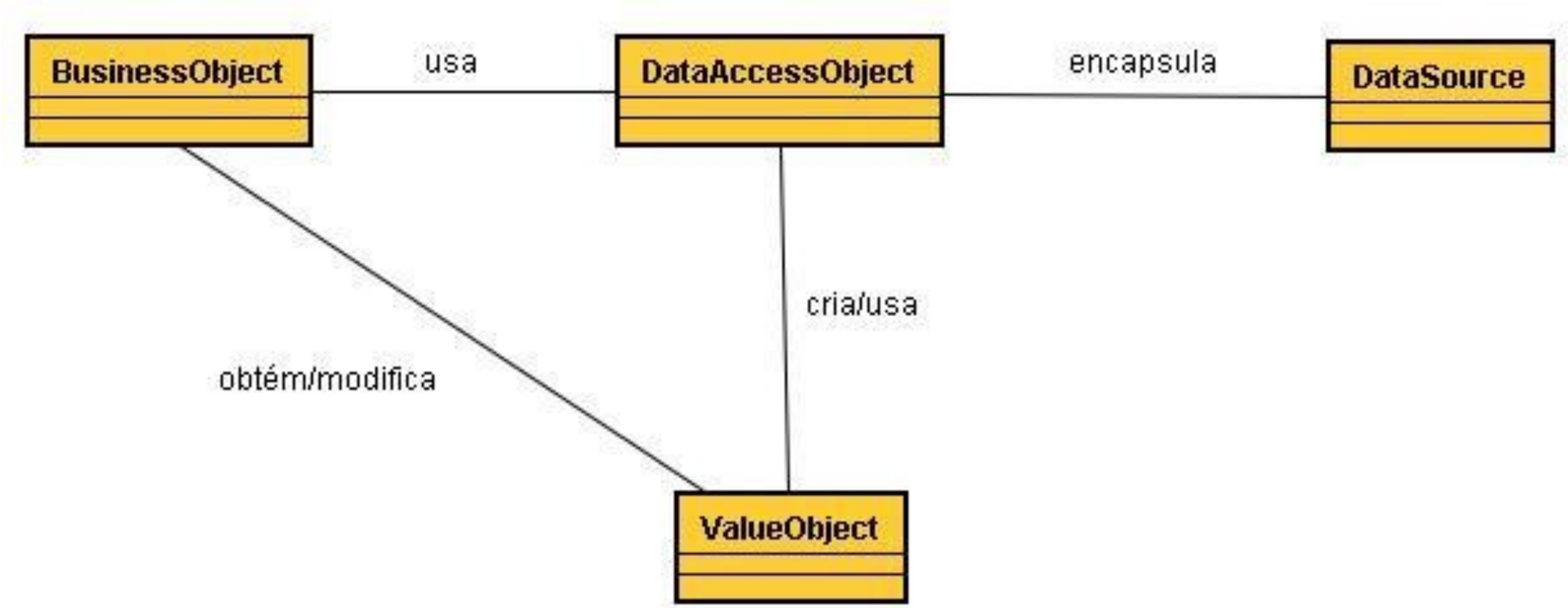
O Padrão DAO

- Data Access Object;
- O objeto DAO atua como uma camada entre o objeto de negócio e a fonte de dados;
  - Abstraindo e encapsulando todo o processo de acesso aos dados;
- O DAO é o responsável por gerenciar a conexão com a fonte de dados

- Com o padrão DAO, temos agora o seguinte cenário:



O Padrão DAO – Estrutura



Os Participantes

- **Business Object:**
  - Representa o objeto do negócio que precisa interagir com a fonte de dados;
    - Para armazenar, atualizar ou recuperar dados;
  - Pode ser qualquer objeto java bean;

Os Participantes

- **DataAccessObject (DAO):**
  - Corresponde ao principal objeto deste padrão;
  - Este objeto conhece os detalhes sobre como armazenar/atualizar informações na fonte de dados;
  - O BusinessObject delega ao DAO as tarefas que requerem qualquer interação com a fonte de dados;
  - O DAO expõe apenas a sua interface para os objetos de negócio;
  - Vantagem: a implementação da fonte de dados pode ser alterada sem a necessidade de alterações nos clientes;

Os Participantes

- **DataSource:**
  - Representa a implementação da fonte de dados;
  - Pode ser um banco de dados relacional, um banco de dados noSQL, uma planilha, um arquivo, um serviço ou outro tipo de recurso;

Os Participantes

- **ValueObject:**
  - Representa o objeto responsável por carregar os valores dos dados;
  - O DAO usa este objeto para repassar as informações obtidas na fonte de dados;
  - O BusinessObject usa este objeto para passar para o DAO as informações que devem ser atualizadas na fonte de dados;

Implementação

- O padrão DAO é geralmente implementado com os padrões de projeto Abstract Factory e Factory Method;
- Para implementar o padrão DAO, precisamos:
  - Criar os value objects;
  - Criar os DAOs;
  - Criar a fábrica de DAOs;
  - Criar os objetos de negócio;

Implementação – Value Objects

- Geralmente é criado um Value Object para cada entidade que compõe a fonte de dados. Exemplo:

```
public class Empregado {

        private String matricula;
        private String nome;
        private float salario;
        private Empregado supervisor;
        private Departamento departamento;

        public String getMatricula() {...}

        public void setMatricula(String matricula) {...}

        public String getNome() {...}

        public void setNome(String nome) {...}

        public float getSalario() {...}

        public void setSalario(float salario) {...}

        public Empregado getSupervisor() {...}

        public void setSupervisor(Empregado supervisor) {...}

        public Departamento getDepartamento() {...}

        public void setDepartamento(Departamento departamento) {...}

}
```

Implementação – DAO

- Normalmente, também cria-se um DAO para cada entidade definida na fonte dados. (Exemplo: EmpregadoDAO, DepartamentoDAO etc.).
- Cada DAO é implementado em duas etapas:
  - 1. Definição da interface do objeto
  - 2. Implementacão da interface definida através de uma ou mais classes concretas

Implementação – DAO

- Esta implementação oferece algumas vantagens:
  - Os clientes não ficam "amarrados" a uma determinada implementação;
  - Podemos substituir a classe que implementa o DAO q qualquer momento, de forma transparente para o cliente;
  - Podemos criar várias implementações da interface;

Implementação – DAO

- Criando a Interface do DAO:

```
public interface EmpregadoDaoIF {

        public void persiste(Empregado empregado) throws PersistenciaException;

        public Empregado localiza(String matricula) throws PersistenciaException;

        public void exlcui(String matricula) throws PersistenciaException;

}
```

Implementação – DAO

- Criando a Interface do DAO EmpregadoDAO:

```
public interface EmpregadoDaoIF {

        public void persiste(Empregado empregado) throws PersistenciaException;

        public Empregado localiza(String matricula) throws PersistenciaException;

        public void exlcui(String matricula) throws PersistenciaException;

}
```

Implementação – DAO

- Criando a classe concreta que implementa a interface:

```
public class EmpregadoDao implements EmpregadoDaoIF{

        private Connection connection;

        public EmpregadoDao() throws PersistenciaException{...}

        public void persiste(Empregado empregado) throws PersistenciaException{...}

        public Empregado localiza(String matricula) throws PersistenciaException{...}

        public List<Empregado> lista() throws PersistenciaException{...}

        public void exlcui(String matricula) throws PersistenciaException{...}

        private Empregado leEmpregado(ResultSet rs) throws SQLException  {...}


}
```

Implementação – DAO

- Criando a classe concreta que implementa a interface:

```
public void persiste(Empregado empregado) throws PersistenciaException{
      String sql = "INSERT INTO EMPREGADO (Matricula, Nome, Salario, " +
              "Supervisor, CodDepartamento) VALUES (?, ?, ?, ?, ?) ";
      try{
            PreparedStatement statement = connection.prepareStatement(sql);
            statement.setString(1, empregado.getMatricula());
            statement.setString(2, empregado.getNome());
            statement.setFloat(3, empregado.getSalario());
            statement.setString(4, empregado.getSupervisor().getSupervisor().getMatricula());
            statement.setInt(5, empregado.getDepartamento().getCodigo());
            statement.execute();
            statement.close();
      }
      catch(Exception e){
          throw new PersistenciaException(e);}
}
```

Implementação – DAO

- Note que o DAO conhece os detalhes de implementação da fonte de dados;
  - Neste exemplo, o DAO sabe que os dados são persistidos em um banco de dados relacional;
  - O DAO também conhece o esquema usado para armazenar os dados;

Implementação – Fábrica de DAOs

- Definidos os DAOs, precisamos de uma estratégia que nos permita instanciá-los de forma conveniente;
- Para isso, podemos usar dois padrões de projeto:
  - Abstract Factory e Factory Method;

Implementação – Fábrica de DAOs

- Estes padrões nos permitem esconder dos clientes os detalhes sobre como os objetos são instanciados;
- O cliente informa para a fábrica o tipo de objeto que ele deseja instanciar;
- A fábrica instancia o objeto desejado e o retorna para o cliente;

Implementação – Fábrica de DAOs

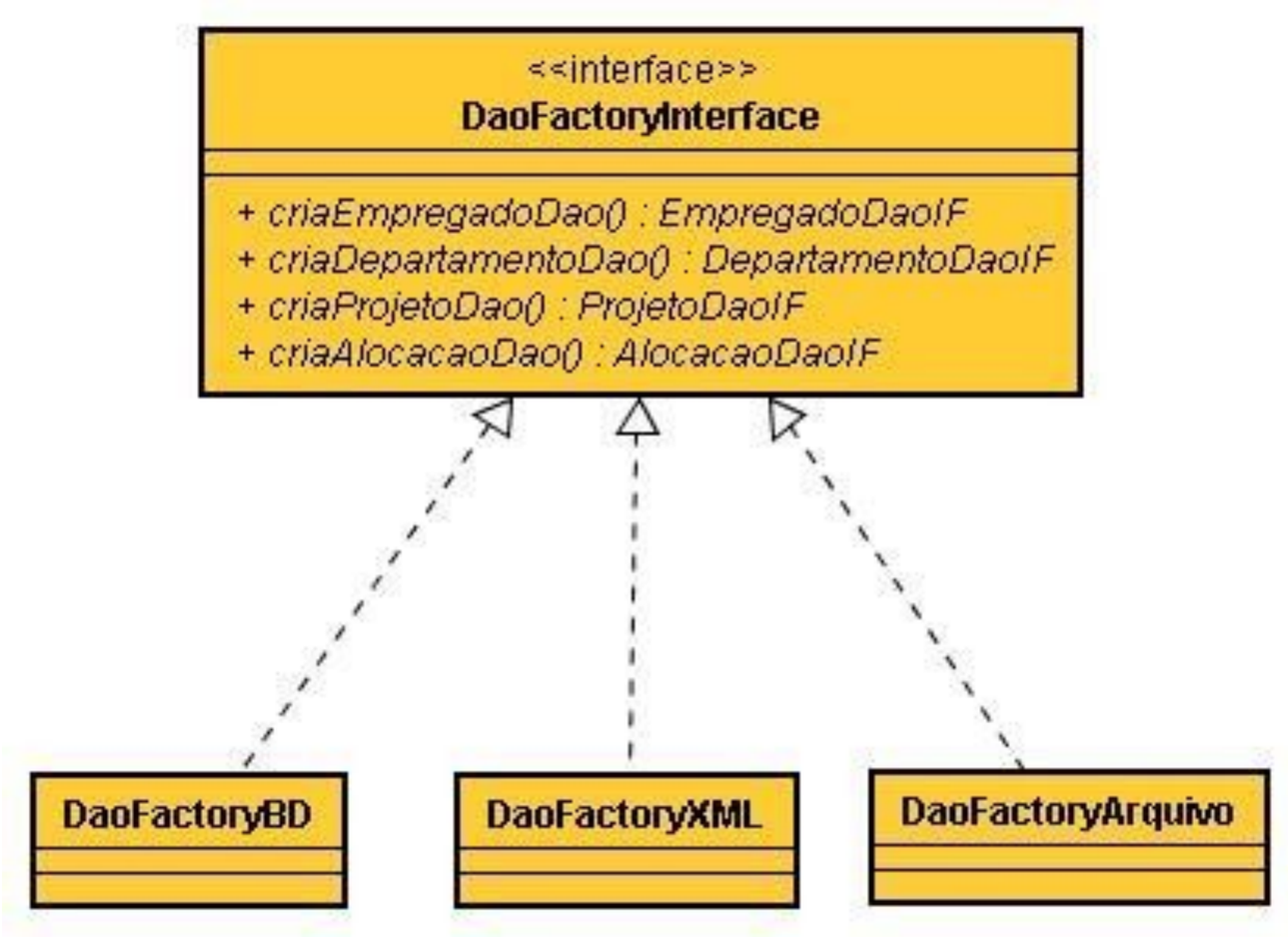
- Em uma aplicação, podemos ter várias fábricas de DAO;
  - Normalmente uma para cada tipo de armazenamento persistente;
- Cada fábrica precisa saber como instanciar os DAOs da aplicação para o seu respectivo tipo de armazenamento;

Implementação – Fábrica de DAOs

- Também criamos a fábrica de DAOs em duas etapas:
  - Definimos a interface da fábrica;
  - Criamos a classe concreta que implementa a fábrica;
- Esta estratégia nos permite criar várias fábricas de DAO na mesma aplicação;
- É importante mesmo quando a aplicação só possui uma forma de armazenamento
  - A aplicação fica mais flexível e mais fácil de se adaptar a possíveis mudanças.

Implementação – Fábrica de DAOs

- Estrutura de implementação do DAO:



Implementação – Fábrica de DAOs

- Veja que a estrutura usada para a implementação:
  - Permite a criação de várias fábricas de DAOs;
  - Define que cada fábrica deve implementar os métodos para criar cada tipo de DAO da aplicação;

Implementação – Fábrica de DAOs

- Definindo a interface de uma fábrica de DAOs:

```
public interface DaoFactoryIF {

	public EmpregadoDaoIF criaEmpregadoDaO() throws PersistenciaException;

	public DepartamentoDaoIF criaDepartamentoDaO() throws PersistenciaException;

	public ProjetoDaoIF criaProjetoDao() throws PersistenciaException;

	public AlocacaoDaoIF criaAlocacaoDao() throws PersistenciaException;

}
```

Implementando a fábrica de DAOs:

```
public class DaoFactoryBD implements DaoFactoryIF{

	private DaoFactoryBD instance;

	public DaoFactoryBD(){...}

	public EmpregadoDaoIF criaEmpregadoDaO() throws PersistenciaException{
		return new EmpregadoDao();
	}

	public DepartamentoDaoIF criaDepartamentoDaO() throws PersistenciaException{
		return new DepartamentoDao();
	}

	public ProjetoDaoIF criaProjetoDao() throws PersistenciaException{
		return new ProjetoDao();
	}

	public AlocacaoDaoIF criaAlocacaoDao() throws PersistenciaException{
		return new AlocacaoDao();
	}

}
```

Implementação – Fábrica de DAOs

- Instanciando a fábrica de DAOs:

```
public class DaoFactory {

        public static DaoFactoryIF createFactory(){
              return new DaoFactoryBD();
        }

}
```

Implementação – Fábrica de DAOs

- Quando há mais de uma forma de persistência, informamos instanciamos a fábrica desejada usando o padrão Factory Method;
- Neste padrão, o tipo de fábrica desejada é passado no construtor do objeto, Instanciando a fábrica de DAOs para diferentes implementações:

```
public class DaoFactory {

        public static final int DAO_BD = 0;
        public static final int DAO_XML = 1;
        public static final int DAO_FILE = 2;

        public static DaoFactoryIF createFactory(int factoryType){
            if(factoryType==DAO_BD)
                 return new DaoFactoryBD();
            if(factoryType==DAO_XML)
                 return new DaoFactoryXML();
            if(factoryType==DAO_FILE)
                 return new DaoFactoryFile();
            return null;
        }

}
```

Implementação – Objetos de Negócio

- Na última etapa da implementação do padrão está o desenvolvimento dos objetos de negócio da aplicação;
- Agora, o objeto de negócio interage apenas com um DAO e não acessa a fonte de dados diretamente.

Implementando o objeto de negócio

```
public class GerenciadorDeEmpregado{

    private DaoFactoryIF fabrica = null;
    private EmpregadoDaoIF empDao = null;

    public GerenciadorDeEmpregado(){
        fabrica = DaoFactory.createFactory(DaoFactory.DAO_BD);
        try {
            empDao = fabrica.criaEmpregadoDao();
        }
        catch (PersistenciaException ex) {}
    }

    public void adicionaEmpregado(String matricula, String nome, float salario,
        Departamento departamento, Empregado supervisor) throws PersistenciaException{
        Empregado novoEmp = new Empregado();
        novoEmp.setMatricula(matricula);
        novoEmp.setNome(nome);
        novoEmp.setSalario(salario);
        novoEmp.setDepartamento(departamento);
        novoEmp.setSupervisor(supervisor);
        empDao.persiste(novoEmp);
    }

    public void removeEmpregado(String matricula) throws PersistenciaException{
        empDao.exlcui(matricula);
    }

}
```

Implementação – Objetos de Negócio

- Note que o objeto de negócio não sabe nada da implementação da fonte de dados;
- Ele também não conhece detalhes sobre a implementação da fábrica;
  - Ele conhece apenas a interface que ele precisa utilizar;
- Assim, podemos alterar a implementação das fábricas sem ter de alterar o objeto de negócio;
- O mesmo acontece para os objetos que implementam os DAOs.

Implementando o objeto de negócio para múltiplas fábricas

```
public class GerenciadorDeEmpregado{

     public void adicionaEmpregado(String matricula, String nome, float salario,
               Departamento departamento, Empregado supervisor) throws PersistenciaException{
               Empregado novoEmp = new Empregado();
               novoEmp.setMatricula(matricula);
               novoEmp.setNome(nome);
               novoEmp.setSalario(salario);
               novoEmp.setDepartamento(departamento);
               novoEmp.setSupervisor(supervisor);

               DaoFactoryIF fabrica = DaoFactory.createFactory(DaoFactory.DAO_BD);
               EmpregadoDaoIF empDao = fabrica.criaEmpregadoDao();
               empDao.persiste(novoEmp);
       }

       public void removeEmpregado(String matricula) throws PersistenciaException{
               DaoFactoryIF fabrica = DaoFactory.createFactory();
               EmpregadoDaoIF empDao = fabrica.criaEmpregadoDao();
               empDao.exlcui(matricula);
       }

}
```

Conclusão

- O padrão DAO:
  - Provê transparência para o processo de acesso aos dados;
  - Permite alterar facilmente a implementação da fonte de dados do sistema;
  - Reduz a complexidade da implementação dos objetos de negócio;
  - Centraliza todo o acesso aos dados numa camada separada;
  - Adiciona uma camada extra de objetos entre os objetos de negócio e a fonte de dados;
  - Aumenta a complexidade do projeto;