INSTITUTO FEDERAL
Rio Grande do Sul

Padrões para persistência de Dados

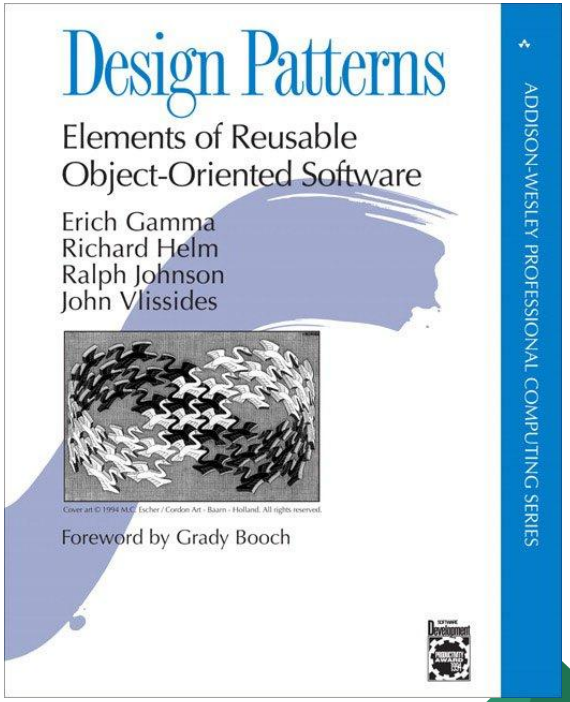
PROF. TIAGO MORAES

Roteiro

- Introdução a padrões de projeto
- Fábrica de conexões
- Acesso ao BD
  - *Active Record*
  - DAO – *Data Access Object*
  - *Repository*
- Considerações finais

Introdução a padrões de projeto

- O Paradigma Orientado a Objetos (POO)
  - Facilitou muito desenvolvimento de projetos, resolvendo problemas de programção procedural
- Mesmo assim alguns problemas seguiam (muitas vezes repetidos) ocorrendo
  - Como resposta surge os padrões de projeto (Desing Patterns)
  - Movimento ganha força em 1995 –
    - Livro Design Patterns: Elements of Reusable object-Oriented Software (GoF – gang of four – quatro autores)

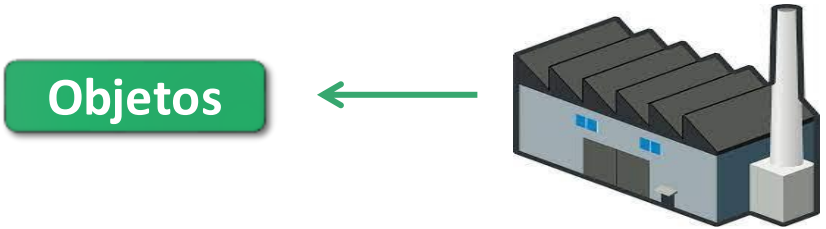


Introdução a padrões de projeto

- Desing Pattern:
  - Formalização de soluções baseadas nas melhores práticas da comunidade para problemas recorrentes
  - Utilização de POO (encapsulamento, associação entre classes, polimorfismo ...)
  - Cada Padrão é definido por um **nome**, **problema**, **solução** e **consequências**
- Utilização:
  - Descobrir partes do projeto de um software que recaem em um problema típico
  - Projetar uma solução para o problema (utilizando um *pattern*):
    - Muitas vezes mais de um padrão pode responder um mesmo problema
    - Experiência vai ajudar a definir a solução ideal para cada projeto
    - Um padrão irá impactar em parte do projeto ou até mesmo no projeto inteiro

Fábrica de conexões

- Factory Method:
  - **Problema**: construir objetos complicados e que são utilizados em muitas partes do código
  - **Solução**: encapsular a construção desse objeto em uma classe específica um Factory
  - **Consequências**:
    - Se alguma mudança for necessária, ocorrerá em um só lugar
    - Diminui o acoplamento da construção desse objeto com o resto do projeto
    - Sempre que se necessitar de um objeto deve-se usar o Factory que cria as instâncias



Fábrica de conexões

- Fábrica de Conexões:
  - A conexão com um banco de dados é um exemplo de um objeto complexo que é criado em várias partes de código (sempre que precisamos acessar e rodar *queries* no BD)

- Exemplo:

```
public class ConnectionFactory{
  public Connection getConnection(){
    String login = "postgres";
    String senha = "postgres";
    String urlcon = "jdbc:postgresql://localhost:5432/testebd";
    try{
      return DriverManager.getConnection(urlcon, login, senha);
    } catch (SqlException e){
      throw new RuntimeException(e);
    }
  }
}
```

Acesso ao BD

- Acessar e trocar dados com um BD é um problema recorrente
  - Dificuldade de mapear dados (relacionais) em Objetos (POO)
    - **Objetos, atributos**
    - **Agregações, composições**
    - **Herança**

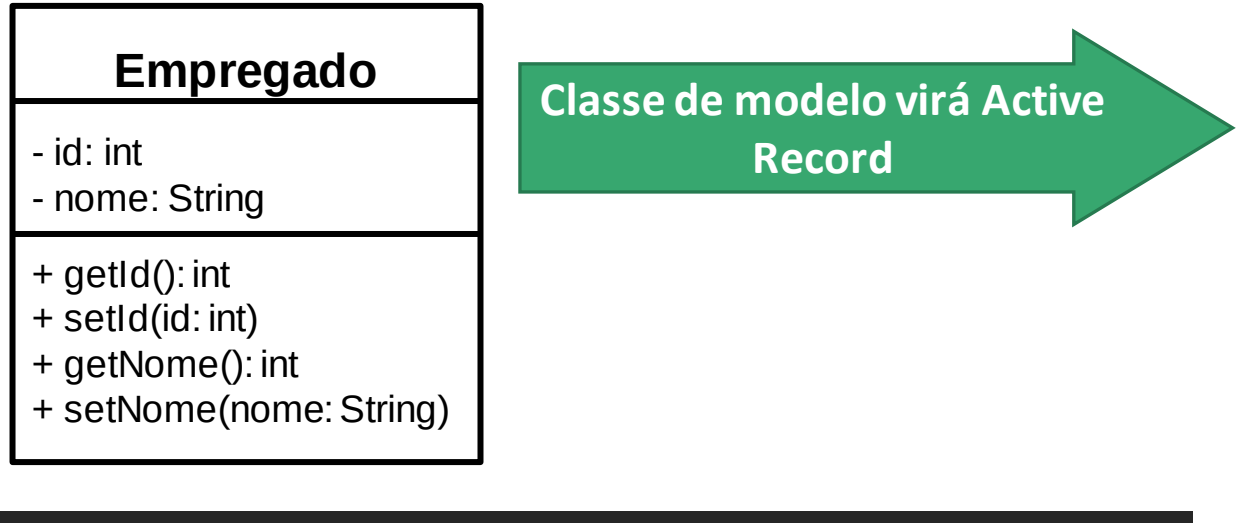
    - **Tabelas**
    - **PK's FK's**
    - **Tipos e restrições de integridade do SGBD**
  - Alguns padrões de projeto podem ser utilizados para responder esse problema
    - *Active Record*
    - *Data Access Object* - DAO
    - *Repository*
    - ...
  - Dependendo da arquitetura e tamanho do projeto um padrão ou outro pode ser mais recomendado
  - Os padrões também podem ser utilizados em conjunto
    - Por exemplo, muitas vezes *repositories* usam DAO's

Active Record

- A responsabilidade de acesso ao BD fica junto com a classe de modelo.
- Cada Objeto da classe de modelo passa a representar um registro ativo do BD
  - se deleta (do BD)
  - se salva (no BD)
  - se altera (no BD)
- Consequências:
  - Formato mais enxuto
  - Mais vantajoso para problemas mais simples
    - Poucas classes de modelo e com poucas regras de negócio
  - Para problemas maiores pode trazer problemas de acoplamento e classes muito grandes e com muitas responsabilidades

Active Record

- Exemplo
  - Considerando a tabela Empregado(id, nome)



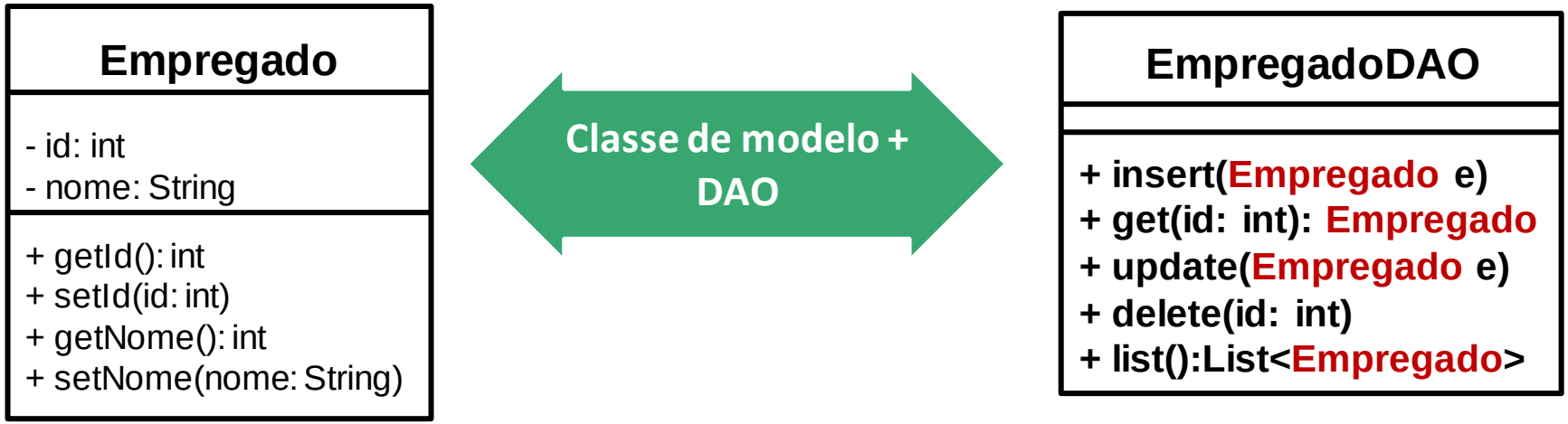

```
Empregado emp3, emp1 = new Empregado(1,"João");
emp1.save();           //salva no BD o empregado criado
emp3 = emp1.get(3);    //busca o empregado de código 3
emp3.delete();         //apaga o empregado de código 3
```

Data Access Object - DAO

- A responsabilidade de acesso ao BD fica separado da classe de modelo.
- Toda complexidade de conectar no BD, rodar SQL's e transformar linhas em objetos (e vice versa) é encapsulada
  - Em uma classe de acesso aos dados
  - Na maior parte das vezes para cada classe de modelo que se deseja persistir dados se cria um DAO
  - Formato mais tradicional e utilizado para tarefa de acesso a BD's
- Consequências:
  - Formato com mais classes – classes menores
  - Melhor separação das responsabilidades (regras de negócio X acesso a dados)

Data Access Object - DAO

- Exemplo
  - Considerando a tabela Empregado(id, nome)




```
EmpregadoDAO empDao = new EmpregadoDAO();
Empregado emp3, emp1 = new Empregado(1,"João");
empDao.insert(emp1);       //salva no BD o empregado criado
emp3 = empDao.get(3);      //busca o empregado de código 3
empDao.delete(3);          //apaga o empregado de código 3
```

Repository

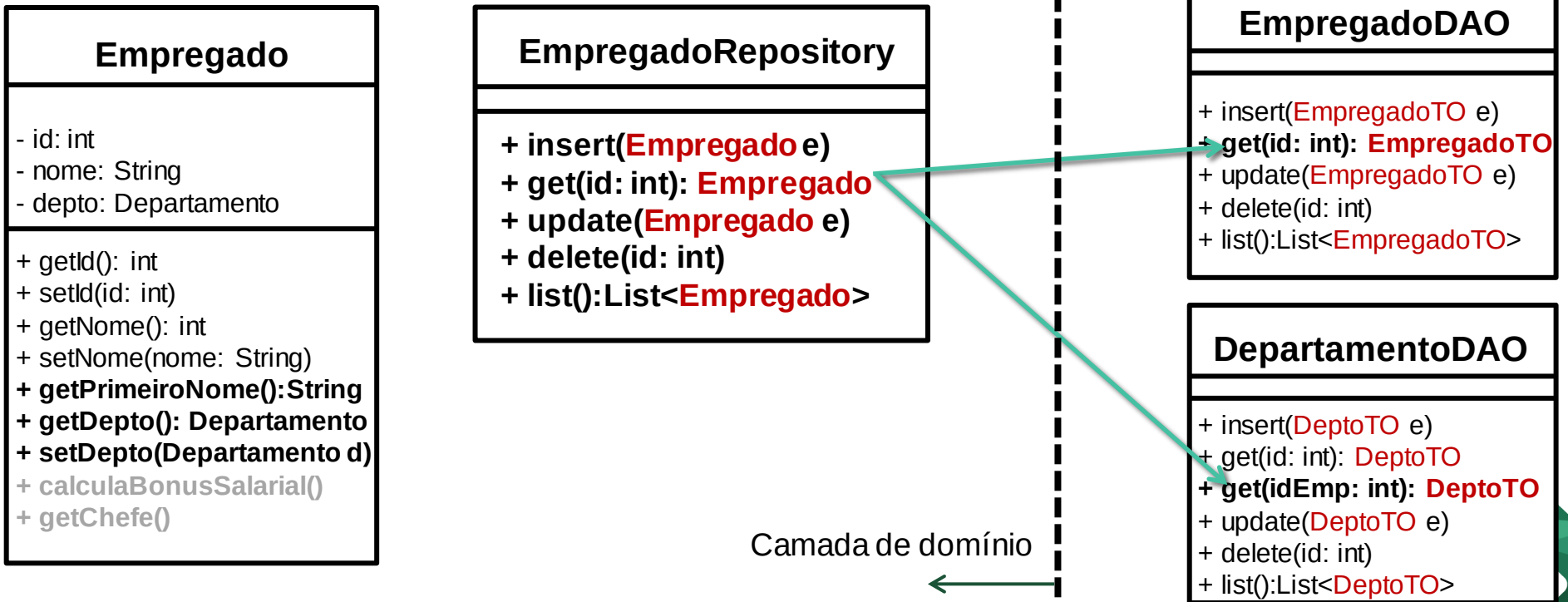
- Cria uma classe de repositório de objetos (de modelo)
  - Mais uma camada para separar responsabilidades
  - Ligado a ideia de DDD (*Domain Drive Development*)
  - A classe de repositório pode ser utilizada em conjunto com DAO's
- Mas então qual é a diferença entre DAO e Repository?
  - DAO – surge do problema de encapsular coisas de Infraestrutura (ex: acesso BD, SQL)
    - Camada de infraestrutura
  - Repository – surge da necessidade de se obter e guardar objetos de domínios
    - Camada de domínio (foco do DDD) - se comunica com os DAO's

Repository

- Retira do DAO a complexidade montar Models complexas (com associações por exemplo)
  - DAO's passam a operar TO (Transfer Objects) ou JavaBeans
    - Classes com atributos, construtor vazio e getters e setters apenas
  - TO's são transferidos para os repositórios que operam as classes de modelo
    - Repository faz o mapeamento objeto-relacional
- Consequências:
  - Formato com mais classes – classes menores
  - Facilita montagem de objetos com relacionamentos por exemplo, diminuindo a complexidade das DAO's
  - Útil para sistemas de complexidade média e alta e/ou utilizem DDD

Repository

- Exemplo
  - tabela Empregado(id, nome, **idDepto [FK]**)



Considerações Finais

- Não existe uma “bala de prata”
  - Cada projeto apresenta realidade diferentes
  - Cada equipe possui conhecimento e experiências diferentes
  - A metodologia de desenvolvimento pode influenciar
- Os Padrões podem ser utilizados separadamente ou em conjunto.
  - Por exemplo: todos podem usar o Factory para criar a conexão
  - DAO e Repository podem atuar juntos
  - Active record e Repository também ...
- DAO → formato mais utilizado – será o que utilizaremos de agora em diante