Arquitetura de Software

<https://bit.ly/ArquiteturaSoftware>

[Downloads](https://drive.google.com/drive/folders/19WYu9H9x2pBHxkjdDyJZGcxbQOij4hph)

Referência Design Pattern

Site

<https://refactoring.guru/pt-br>

Livros

* Padrões de Projetos: Soluções Reutilizáveis de Software Orientados a Objetos

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

* Engenharia de Software Moderna

Princípios e Práticas para Desenvolvimento de Software com Produtividade

Marco Tulio Valente

Professor Gerson Risso

[gerson.risso@sp.senac.br](mailto:gerson.risso@sp.senac.br)

Duas avaliações e uma substitutiva.

Média das avaliações - 70%

Média ADO - 30%

**Datas das avaliações**

**P1 - 05/10/22**

**P2 - 16/11/22**

**Substitutiva - 23/11/22**

Aula 1 - Apresentação e revisão de conteúdo/conceitos

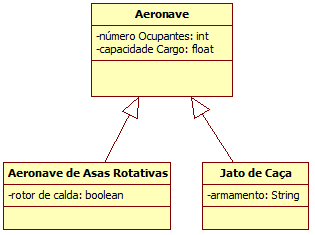
[Apresentação - Aula 1](https://docs.google.com/presentation/d/11peS38bPoYX-frK3hLQPNS9SornvgDKvvUiTWNPd_YU/edit#slide=id.g10e149974c6_3_16)

Exemplo 1

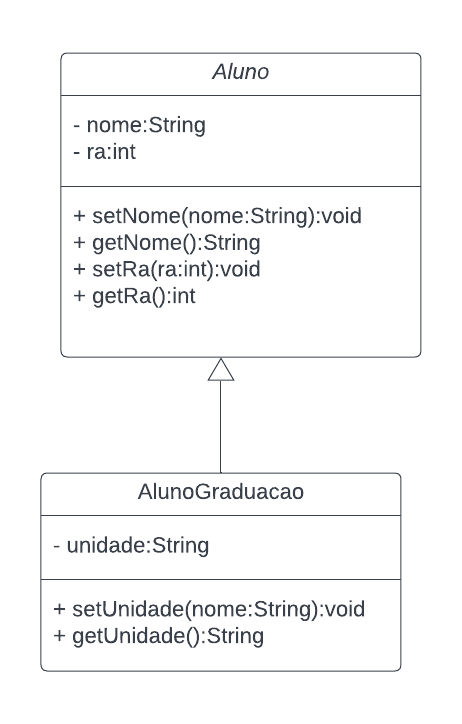
| public class Pessoa {  private String nome;    public Pessoa(){}    public Pessoa(String nome){  this.nome=nome;  }    public void setNome(String nome){  this.nome=nome;  }    public String getNome(){  return this.nome;  }  @Override  public String toString(){  return "Nome="+this.nome;  }  }  public class PessoaFisica extends Pessoa{  private long cpf;    public PessoaFisica(){}    public PessoaFisica(String nome,long cpf){  super(nome);  this.cpf=cpf;  }  public long getCpf() {  return cpf;  }  public void setCpf(long cpf) {  this.cpf = cpf;  }  @Override  public String toString() {  return "PessoaFisica{ Nome="+this.getNome()+  " - cpf=" + cpf + '}';  }    }  public class Funcionario extends PessoaFisica{  private int matricula;  public Funcionario() {  }  public Funcionario(int matricula, String nome, long cpf) {  super(nome, cpf);  this.matricula = matricula;  }  public int getMatricula() {  return matricula;  }  public void setMatricula(int matricula) {  this.matricula = matricula;  }  @Override  public String toString() {  return "Funcionario{" + "matricula=" + matricula + '}';  }    }  public class PessoaJuridica extends Pessoa{  private long cnpj;  public PessoaJuridica() {  }  public PessoaJuridica(long cnpj, String nome) {  super(nome);  this.cnpj = cnpj;  }  public long getCnpj() {  return cnpj;  }  public void setCnpj(long cnpj) {  this.cnpj = cnpj;  }  @Override  public String toString() {  return "PessoaJuridica{" + "cnpj=" + cnpj + '}';  }    }  public class Exemplo1 {  public static void main(String[] args) {  Pessoa p=new Pessoa("Gerson");  System.out.println(p);  }    } |
| --- |

**Exercícios**

1. Implemente, em java, o diagrama de classes da UML. Escreva os métodos e os construtores.

****

1. Observe o diagrama abaixo, analise-o e implemente (Java) às duas classes e adicione as classes AlunoEad (Polo: String) e AlunoPos (Tema:String).



Aula 2 - Continuando com a revisão e posteriormente abordaremos o conceito de design patterns.

Exemplo 2 - Dada a classe abstrata, implemente as classes Carro e Moto. Escreva também uma classe de teste para implementar o polimorfismo.

| public abstract class Veiculo {    public abstract void acelerar();    public void painel(){  System.out.println("Exibindo informações");  }  } |
| --- |

Exemplo 3 - Um projeto de Batalha de Tanques tem previamente definido três contratos (interfaces).

| public interface Canhao {  public void atirar();  }  public interface Cores {  public void setCorTanque(CoresRGB cor);  public CoresRGB getCorTanque();  public CoresRGB getCorCanhao();  public void setCorCanhao(CoresRGB cor);  }  public interface Posicao {  public void setX(float x);  public void setY(float y);  public float getX();  public float getY();  } |
| --- |

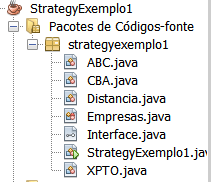
[Revisão de OO e Introdução ao Design Patterns](https://docs.google.com/presentation/d/1J17QpkF1u5t7bP3WkYJW7VMx423SP-FhOCs4kFwuHGI/edit#slide=id.g73082a65e7_0_2)

Aula 3

**Padrão de projeto - Strategy**

[Padrão - Strategy](https://docs.google.com/presentation/d/1-h6hdSuYwVTP938cSejT_4noP1oN0WxYRqCXb60CVkA/edit#slide=id.p)

Exemplo 1



| public class ExemploSemStrategy {  public static void main(String[] args) {  Distancia distancia=new Distancia(100);  try {  System.out.println(obterFrete(distancia,"XPTO"));  } catch (IllegalArgumentException e) {  System.out.println(e.getMessage());  }    }    public static double obterFrete(Distancia distancia,String empresa){  if(empresa.equalsIgnoreCase("abc")){  return distancia.getDistancia()\*0.1;  }else if(empresa.equalsIgnoreCase("cba")){  return distancia.getDistancia()\*0.2;  }else if(empresa.equalsIgnoreCase("xpto")){  return distancia.getDistancia()\*0.3;  }else{  throw new IllegalArgumentException("empresa inválida!");  }  }    }  public class Distancia {  private double distancia;  public Distancia(double distancia) {  this.distancia = distancia;  }  public double getDistancia() {  return distancia;  }    } |
| --- |

Aplicando Strategy

| public interface Empresas {  public abstract double calcularFrete(Distancia distancia,ListaEmpresas empresa);  }  public class ExemploSemStrategy {  public static void main(String[] args) {  Distancia distancia = new Distancia(100);  System.out.println(obterFrete(distancia, ListaEmpresas.values()[2]));  }  public static double obterFrete(Distancia distancia, ListaEmpresas empresa) {  Empresas escolhida;  double valor=0;  if (empresa.equals(ListaEmpresas.ABC)) {  escolhida=new ABC();  valor=escolhida.calcularFrete(distancia, empresa);  } else if (empresa.equals(ListaEmpresas.CBA)) {  escolhida=new CBA();  valor=escolhida.calcularFrete(distancia, empresa);  } else if (empresa.equals(ListaEmpresas.XPTO)) {  escolhida=new XPTO();  valor=escolhida.calcularFrete(distancia, empresa);  }    return valor;  }  }  public class Distancia {  private double distancia;  public Distancia(double distancia) {  this.distancia = distancia;  }  public double getDistancia() {  return distancia;  }    }  public enum ListaEmpresas{  ABC,  CBA,  XPTO;  }    public class ABC implements Empresas{  @Override  public double calcularFrete(Distancia distancia, ListaEmpresas empresa) {  return distancia.getDistancia()\*0.1;  }    }  public class CBA implements Empresas{  @Override  public double calcularFrete(Distancia distancia, ListaEmpresas empresa) {  return distancia.getDistancia()\*0.2;  }    }  public class XPTO implements Empresas{  @Override  public double calcularFrete(Distancia distancia, ListaEmpresas empresa) {  return distancia.getDistancia()\*0.3;  }    } |
| --- |

Eliminando o bloco if..else

| public enum ListaEmpresas implements Empresas{  ABC{  @Override  public double calcularFrete(Distancia distancia, ListaEmpresas empresa) {  return distancia.getDistancia()\*0.1;  }    },  CBA{  @Override  public double calcularFrete(Distancia distancia, ListaEmpresas empresa) {  return distancia.getDistancia()\*0.2;  }    },  XPTO{  @Override  public double calcularFrete(Distancia distancia, ListaEmpresas empresa) {  return distancia.getDistancia()\*0.3;  }    };    }  public class ExemploSemStrategy {  public static void main(String[] args) {  Distancia distancia = new Distancia(130);  System.out.println(obterFrete(distancia, ListaEmpresas.values()[2]));  }  public static double obterFrete(Distancia distancia, ListaEmpresas empresa) {    return empresa.calcularFrete(distancia, empresa);    }  }  Observação: Com esse aprimoramento não são necessárias às classes das empresas: ABC, CBA, XPTO. |
| --- |

**Exemplo 2 - Dada a classe, aplique o pattern Strategy.**

| public class Strategy {    public static void main(String[] args) {  char[] lista = {'a', 'a', 'c', 'e', 'f', 'h', 't', 'u', 'i', 'a', 'e', 'b'};  Strategy strategy;  strategy = new Strategy();  int n = 2;  switch (n) {  case 1:  strategy.converteMaiuculos(lista);  break;  case 2:  strategy.contarVogal(lista);  break;  default:  strategy.mostrarCaracteres(lista);  break;  }  }    public void mostrarCaracteres(char[] lista){  for(char c:lista){  System.out.print(c+" ");  }    }    public void converteMaiuculos(char[] lista) {  for (int i = 0; i < lista.length; i++) {  lista[i] = Character.toUpperCase(lista[i]);  }  }    public void contarVogal(char[] lista) {  char[] vogal = {'a', 'e', 'i', 'o', 'u'};  int cont = 0;  for (int i = 0; i < lista.length; i++) {  for (int j = 0; j < vogal.length; j++) {  if (lista[i] == vogal[j]) {  cont++;  }  }  }  System.out.println(cont);  }    } |
| --- |

**Exercícios**

1. Uma empresa estabeleceu uma regra para cálculo dos salários dos seus funcionários. Os salários serão calculados com base em um fator, que é diferenciado para cada departamento. Essa regra tem o objetivo de incrementar o plano de carreiras da empresa.

* Gerência - bônus 6%
* Desenvolvimento - bônus 5%
* Suporte - bônus 4%
* Demais departamentos - bônus 3%

cálculo do salário

* salário+salário\*bônus

Aplicar o pattern Strategy para esse caso.

1. Defina padrões de projeto.
2. Leia o trecho de uma possível conversa entre dois programadores e identifique os designs patterns contidos nela.

| **Programador 1:**  “Nessa sprint vou refatorar aquele trecho utilizando um método fábrica. O que você acha?”  **Programador 2:**  “Boa ideia! Falando nisso...  Podemos utilizar a composição para implementar aquela hierarquia, aproveitando a sua natureza recursiva!” |
| --- |

1. Quais são as características que o padrão singleton deve ter?
2. Para o código a seguir:

| public class Cenario {  private static Cenario cenario=null;    private Cenario(){}  public static Cenario getCenario(){  if(cenario==null)  cenario=new Cenario();    return cenario;  }  } |
| --- |

Escreva um método configuracoes que deve exibir uma mensagem. Esse método deve ser invocado pela instância cenario. Teste para o ambiente multitarefa, analise e mostre se há ou não, “quebra” no padrão singleton.

1. Com base no exemplo de Strategy, quais são os problemas de se utilizar o bloco if..else?
2. Com base no exemplo da aula, determine as classes que correspondem às estratégias e as classes que correspondem ao contexto.
3. Considerando que existem vários impostos (ip1,ip2, ip3) sobre produtos e serviços. E que a taxas são:

* ip1 - 10%
* ip2 - 15%
* ip3 - 25%

Escreva um projeto Java, tendo como base o padrão de projeto strategy, que determine o cálculo de orçamento que envolva serviço ou produto considerando as incidências dos impostos listados acima.

*Cálculo: orcamento\*imposto*

1. Dados os métodos de ordenação a seguir, escreva um projeto java que implemente o padrão Strategy. Escreva uma classe que teste os métodos utilizados.

| public static void insertion(float v[]){  int i,j;  float temp;  for(i=1;i<n;i++){  temp=v[i];  j=i;  while(j>0 && v[j-1]>=temp){  v[j]=v[j-1];  j--;  }  v[j]=temp;  }  } |
| --- |

| public static void selection(int v[]){  int i,j,min,aux;  for(i=0;i<MAX-1;i++){  min=i;  for(j=i+1;j<MAX;j++)  if(v[j]<v[min])  min=j;    aux=v[i];  v[i]=v[min];  v[min]=aux;  }  } |
| --- |

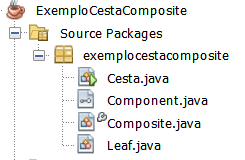
| public static void bubble(int c[]){  int i,j;  float temp;  for(j=0;j<T-1;j++)  for(i=0;i<T-1-j;i++)  if(c[i]>c[i+1]){  temp=c[i];  c[i]=c[i+1];  c[i+1]=temp;  }  } |
| --- |

1. Com base no exercício 3, quais são as classes de contexto e as estratégias?

**Padrão Composite**

[Padrão Composite](https://docs.google.com/presentation/d/1D6JbgiH1zeM-02YGIHOgPMShxKzvINocLxdIDiTqtP0/edit?usp=sharing)

Exemplo 1: Simular uma cesta de café da manhã. O objetivo é determinar o preço total da cesta.



| public interface Component {  public double getPreco();  }  import java.util.ArrayList;  import java.util.List;  public class Composite implements Component{  private List<Component> lista=new ArrayList<>();  private String nome;  public Composite() {  }  public Composite(String nome) {  this.nome = nome;  }  /\*\*  \* Adicionar componentes na lista.  \* @param c Component  \*/  public void adicionar(Component c){  this.lista.add(c);  }    public void remover(Component c){  this.lista.remove(c);  }    @Override  public double getPreco() {  double total=0;  for(Component v:lista){  total+=v.getPreco();  }  return total;  }  @Override  public String toString() {  return "Composite{" + "lista=" + lista + ", nome=" + nome + '}';  }  }  public class Leaf implements Component{  private double preco;  private String nome;    public Leaf() {  }  public Leaf(double preco, String nome) {  this.preco = preco;  this.nome = nome;  }    @Override  public double getPreco() {  return this.preco;  }  @Override  public String toString() {  return "Leaf{" + "preco=" + preco + ", nome=" + nome + '}';  }    }  public class Cesta {  public static void main(String[] args) {  Composite cesta=new Composite("Café da manhã");  Leaf produto1=new Leaf(12.45, "Xícara");  Leaf produto2=new Leaf(8.6, "Café");  Leaf produto3=new Leaf(5.23, "Biscoito");  Leaf produto4=new Leaf(1.62, "Geléia de morango");  Leaf produto5=new Leaf(7.98, "Geléia de damasco");    Composite caixa=new Composite("Caixa de geléias");  caixa.adicionar(produto4);  caixa.adicionar(produto5);  cesta.adicionar(produto1);  cesta.adicionar(produto2);  cesta.adicionar(produto3);  cesta.adicionar(caixa);  System.out.println("Total: "+cesta.getPreco());  }  } |
| --- |

**Exercícios**

1. Implemente o padrão composite para representar a estrutura de uma empresa composta por Diretoria (Composite), Departamento (Leaf). Considere a interface DepartamentoComponent com os métodos getCusto() e getFuncionarios() - Retorna a quantidade de funcionários de um departamento.
2. Tomando como base o exemplo da cesta de café, aprimore-o escrevendo uma classe folha para tratar produtos em promoção, onde o Cliente deve entrar com o preço e com o desconto.
3. Crie um projeto de sistema de arquivos, onde os leafs terão os atributos nome\_arquivo (String) e tamanho (float). A classe Composite, além das operações padrão, deverá determinar o tamanho total dos arquivos no sistema.

**Factory Method**

[Apresentação](https://docs.google.com/presentation/d/1QENgmKqHkbMQrB1yQE-8TU5XddWOgHWUkNhJBvUUSNo/edit?usp=sharing)

Exemplo

| //Product  public interface Carro {  public double getPreco();  public String getModelo();  }  //ConcreteProduct  public class Fox implements Carro{    private double preco;  private String modelo;  public Fox() {  }  public Fox(double preco, String modelo) {  this.preco = preco;  this.modelo = modelo;  }    @Override  public double getPreco() {  return this.preco;  }  @Override  public String getModelo() {  return this.modelo;  }  @Override  public String toString() {  return "Fox{" + "preco=" + preco + ", modelo=" + modelo + '}';  }    }  public class Jetta implements Carro {  private double preco;  private String modelo;  public Jetta() {  }  public Jetta(double preco, String modelo) {  this.preco = preco;  this.modelo = modelo;  }    @Override  public double getPreco() {  return this.preco;  }  @Override  public String getModelo() {  return this.modelo;  }  @Override  public String toString() {  return "Jetta{" + "preco=" + preco + ", modelo=" + modelo + '}';  }    }  public enum Lista {  FOX,JETTA;  }  public class FabricaVW {  public static Carro factoryMethod(Lista lista) {  Carro c = null;  if (lista.equals(Lista.FOX)) {  c = new Fox(60000, "Cross fox");  } else if (lista.equals(Lista.JETTA)) {  c = new Jetta(120000, "Jetta TSI");  }  return c;  }  }  public class FabricaCarro {  public static void main(String[] args) {  Carro c=FabricaVW.factoryMethod(Lista.values()[0]);  System.out.println(c);  }    } |
| --- |

Modificando o projeto para incluir a fábrica de carro Fiat.

| public interface Fabrica {  public Carro factoryMethod(Lista lista);  }  //ConcreteProduct  public class Argo implements Carro {  private double preco;  private String modelo;  public Argo() {  }  public Argo(double preco, String modelo) {  this.preco = preco;  this.modelo = modelo;  }      @Override  public double getPreco() {  return this.preco;  }  @Override  public String getModelo() {  return this.modelo;  }  @Override  public String toString() {  return "Argo{" + "preco=" + preco + ", modelo=" + modelo + '}';  }    }  //ConcreteProduct  public class Uno implements Carro {  private double preco;  private String modelo;  public Uno() {  }  public Uno(double preco, String modelo) {  this.preco = preco;  this.modelo = modelo;  }      @Override  public double getPreco() {  return this.preco;  }  @Override  public String getModelo() {  return this.modelo;  }  @Override  public String toString() {  return "Uno{" + "preco=" + preco + ", modelo=" + modelo + '}';  }      }  //ConcreteCreator  public class FabricaFiat implements Fabrica{  @Override  public Carro factoryMethod(Lista lista) {  Carro c = null;  if (lista.equals(Lista.ARGO)) {  c = new Argo(40000, "Argo II");  } else if (lista.equals(Lista.UNO)) {  c = new Uno(12000, "Uno Milli");  }  return c;  }  }  //ConcreteCreator  public class FabricaVW implements Fabrica{  @Override  public Carro factoryMethod(Lista lista) {  Carro c = null;  if (lista.equals(Lista.FOX)) {  c = new Fox(60000, "Cross fox");  } else if (lista.equals(Lista.JETTA)) {  c = new Jetta(120000, "Jetta TSI");  }  return c;  }  }  public enum Lista {  FOX,JETTA,ARGO,UNO;  }  public class FabricaCarro {  public static void main(String[] args) {  Fabrica f=new FabricaFiat();  Carro c=f.factoryMethod(Lista.values()[3]);  System.out.println(c);  }    } |
| --- |

**Exercícios**

1. Aplicar o pattern Factory Method no projeto batalha de tanques.

[Batalha de tanques](https://drive.google.com/drive/folders/19WYu9H9x2pBHxkjdDyJZGcxbQOij4hph?usp=sharing)

1. Aplique o pattern factory Method para retornar a quantidade de arestas correspondentes à figura. Por exemplo: quadrado - 4 arestas, triângulo - 3 arestas.
2. Considere uma instituição de ensino que possui duas modalidades de graduação: presencial e EAD, e uma modalidade de pós-graduação. Ao fazer a matrícula em um dos cursos, o aluno deve informar o local que deseja estudar (se for presencial) ou o pólo de apoio (se for EAD). E dependendo da modalidade que o aluno se matricule, deve retornar a informação do diploma requisitado.

| Curso | Informar | Requisito - Diploma |
| --- | --- | --- |
| Graduação presencial | Unidade | Ensino médio |
| Graduação EAD | Pólo | Ensino médio |
| Pós-graduação | Unidade | Graduação |

Escreva um projeto aplicando o pattern Factory Method considerando as regras definidas para o projeto.

**Padrão State**

[Apresentação](https://docs.google.com/presentation/d/1MYE_yqriHRglgktk7HCCjCzQCqP6gl2wuqCl5Hidzbg/edit?usp=sharing)

Solução

| public interface State {  public void atrasada();  public void concluida();  public void pendente();  }  public class Context {  private State state = new Pendente(this);  public State getState() {  return state;  }  void setState(State state) {  this.state = state;  System.out.println(getState());  }  public void requisitarConcluida(){  state.concluida();  }    public void requisitarAtrasada(){  state.atrasada();  }    public void requisitarPendente(){  state.pendente();  }  }  public class Atrasada implements State{    private final String nome="Atrasada";  private Context tarefa;  public Atrasada(Context tarefa) {  this.tarefa = tarefa;  }    @Override  public void atrasada() {  System.out.println("Já está atrasada");  }  @Override  public void concluida() {  tarefa.setState(new Concluida(tarefa));  }  @Override  public void pendente() {  System.out.println("Já está atrasada");  }  @Override  public String toString() {  return "Novo estado da tarefa: "+nome;  }  }  public class Concluida implements State {  private String nome = "Concluída";  private Context tarefa;  public Concluida(Context tarefa) {  this.tarefa = tarefa;  }  @Override  public void atrasada() {  System.out.println("Já está concluída");  }  @Override  public void concluida() {  System.out.println("Já está concluída");  }  @Override  public void pendente() {  tarefa.setState(new Pendente(tarefa));  }  @Override  public String toString() {  return "Novo estado da tarefa: " + nome;  }  }  public class Pendente implements State {  private final String nome = "Pendente";  private Context tarefa;  public Pendente(Context tarefa) {  this.tarefa = tarefa;  }  @Override  public void atrasada() {  tarefa.setState(new Atrasada(tarefa));  }  @Override  public void concluida() {  tarefa.setState(new Concluida(tarefa));  }  @Override  public void pendente() {  System.out.println("Já está pendente");  }  @Override  public String toString() {  return "Novo estado da tarefa: " + nome;  }  }  public class GerenciadorTarefas {  public static void main(String[] args) throws ParseException {  Context tarefa=new Context();  tarefa.requisitarAtrasada();  tarefa.requisitarConcluida();  tarefa.requisitarPendente();  tarefa.requisitarConcluida();  tarefa.requisitarConcluida();  }  } |
| --- |

**Exercícios**

1. Elabore um projeto de game tipo Plataforma onde o personagem possui os estados: esperando, correndo, pulando e abaixando. Implemente o padrão state para esse contexto.
2. Elabore um projeto utilizando o pattern state para modelar os estados possíveis de uma compra de produto online: pagamento confirmado, pedido em preparação, cancelar pedido, pedido enviado.
3. Aprimore o projeto implementado em aula para instanciar apenas uma tarefa, utilize o padrão singleton.

**Padrão Observer**

[Apresentação - Padrão Observer](https://docs.google.com/presentation/d/1AFZ2CoVCtVsBgtS88JDGpmquF_mKVnk4OjgOG5ck5xw/edit?usp=sharing)

Exercícios

1. Modifique o exemplo passado em aula para possibilitar a remoção de objetos observadores, simulando a destruição de uma nave inimiga.

Implemente o padrão singleton para garantir que instancie apenas um jogador.

1. Aplique o padrão Observer para uma agência que informa vagas de empregos para assinantes do serviço. Use o exemplo como base para realizar esse exercício.

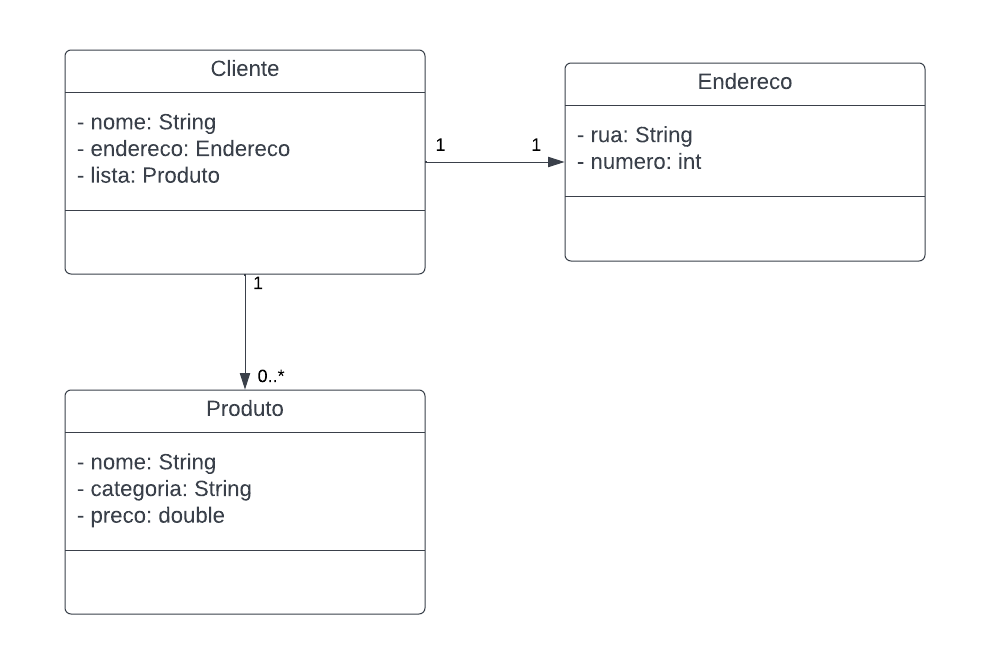
**Padrão Facade (Fachada)**

[Apresentação do padrão Facade](https://docs.google.com/presentation/d/18CpeT9Cwu5FJTtQPPf628afs9w3LZz5o-kVRjjzMlVI/edit#slide=id.gf3ccbcfd87_0_0)

| package lista;  public class UsaLista {  public static void main(String[] args) {  Facade facade=new Facade();  facade.inserir("Gerson","TADS",123);  facade.inserir("Anna","TADS",4321);  facade.inserir("Sandra","BSI",789);  facade.remover(4321);  Metodos.exibir();  }  }  public class Facade {  public void inserir(String nome, String curso, int ra) {  Aluno aluno = new Aluno(nome, curso, ra);  Metodos.inserirNo(aluno);  }      public void remover(int ra){  boolean teste = Metodos.remover(ra);  if (teste) {  System.out.println("Removido com sucesso!");  } else {  System.out.println("Não removeu!");  }  }    }  public class Metodos {  private static No inicio = null, atual, aux;  private static int quantidadeNo = 0;  public static void inserirNo(Object objeto) {  if (inicio == null) {  No novoNo = new No(null, null, objeto);  inicio = novoNo;  aux = inicio;  } else {  No novoNo = new No(null, aux, objeto);  atual = novoNo;  aux.setProx(atual);  aux = atual;  }  quantidadeNo++;  }  public static void exibir() {  No imprimir = inicio;  while (imprimir != null) {  System.out.println(imprimir);  imprimir = imprimir.getProx();  }  }  public static Object buscar(Object objeto) {  No pesq = inicio;  int ra=(int)objeto;  while (pesq != null) {  Aluno alunoPesq=(Aluno)pesq.getInfo();  if(alunoPesq.getRa()==ra) {  return pesq;  }  pesq = pesq.getProx();  }  return null;  }    public static boolean remover(Object object) {  No rem = (No)buscar(object);  if (rem != null && inicio != null) {  //Remover do início  if (rem == inicio) {  if (quantidadeNo > 1) {  inicio = inicio.getProx();  rem.setProx(null);  inicio.setAnt(null);  }else{//Se tiver apenas um elemento na lista  inicio=null;  }  } else if (rem == atual) {//Removendo o último nó  atual = rem.getAnt();  atual.getProx().setAnt(null);  atual.setProx(null);  } else {//Remoção de um nó qualquer  rem.getAnt().setProx(rem.getProx());  rem.getProx().setAnt(rem.getAnt());  rem.setAnt(null);  rem.setProx(null);  }  quantidadeNo--;  return true;  }  return false;  }  }  package lista;    public class No {  private No prox,ant;  private Object info;  public No() {  }  public No(No prox, No ant, Object info) {  this.prox = prox;  this.ant = ant;  this.info = info;  }  public No getProx() {  return prox;  }  public void setProx(No prox) {  this.prox = prox;  }  public No getAnt() {  return ant;  }  public void setAnt(No ant) {  this.ant = ant;  }  public Object getInfo() {  return info;  }  public void setInfo(Object info) {  this.info = info;  }  @Override  public String toString() {  return "info=" + info;  }    }  package lista;  public class UsaLista {  public static void main(String[] args) {  Facade facade=new Facade();  facade.inserir("Gerson","TADS",123);  facade.inserir("Anna","TADS",4321);  facade.inserir("Sandra","BSI",789);  facade.remover(4321);  Metodos.exibir();  }  } |
| --- |

**Exercício**

1. Considere o diagrama de classes da UML.



Pede-se:

1. Escreva as três classes com os atributos apresentados, encapsule-os, crie dois construtores (no mínimo) e crie um método toString().
2. Escreva a classe principal e uma classe para simplificar a interface.

**Dica:** Escreva a classe principal e instancie as classes mencionadas, atribuindo valores aos objetos. A partir daí tente visualizar métodos que possam agrupar as instruções e daí escreva a classe Facade.

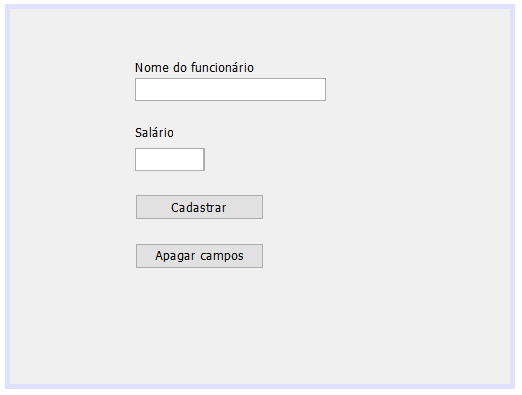
**Classes internas, classes anônimas e expressões lambdas**

[**Apresentação**](https://docs.google.com/presentation/d/13RIQ-epfgKWJ7H3CWNPz8brMr-3_ozdxuas32wWqXb4/edit?usp=sharing)

**Padrão Arquitetural MVC**

[Apresentação MVC](https://docs.google.com/presentation/d/1krIpyBIjBKExCD1xtBKD6z9vt7nnU2XyJHHvM2SqLtI/edit#slide=id.g8040275d86_0_57)

1. Elabore uma aplicação desktop no padrão arquitetural MVC, para cadastrar funcionários, escreva uma classe para o layout apresentado, uma classe Controle e uma classe Funcionario correspondente. Verificar valores de salários entre R$1.300,00 e R$10.000,00. Para valores fora da faixa, lançar exceção.



[Conexão BD usando Framework](https://docs.google.com/presentation/d/1Id8nwa3GQSbw9-RiuzW9cuK7zCNmLn8-0uKPdQ92YLs/edit?usp=sharing)

**JPQL - Java Persistence Query Language**

Realiza consultas em entidades persistentes de forma independente da base de dados.

Exemplos de consulta com Select

**Usando a cláusula WHERE**

public Query consultarId(Long id) {

Query sql=em.createQuery("SELECT c FROM Cliente c where c.id=:id");

sql.setParameter("id", id);

return sql;

}

**Usando a cláusula LIKE**

public List listarNome(String nome){

Query sql=em.createQuery("SELECT c FROM Cliente c WHERE c.nome LIKE "+"'%"+nome+"%'");

return sql.getResultList();

}

**Select simples**

public List<Cliente> listar() {

return em.createQuery("SELECT c FROM Cliente As c").getResultList();

}

**Delete**

public void apagar(Long id){

em.getTransaction().begin();

Query sql=em.createQuery("DELETE FROM Cliente c WHERE c.id=:id");

sql.setParameter("id",id);

sql.executeUpdate();

em.getTransaction().commit();

}

**Update**

public void atualizar(double salario, int id) {

em.getTransaction().begin();

Query query = em.createQuery("UPDATE Funcionario f SET f.salario = :salario WHERE f.id = :id");

query.setParameter("salario", salario);

query.setParameter("id", id);

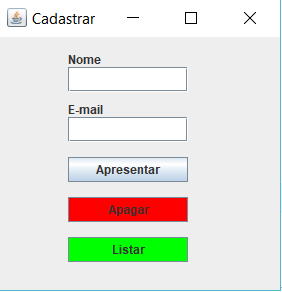
query.executeUpdate();

em.getTransaction().commit();

}

Exercícios

1. Inclua essas consultas no projeto passado nos slides.
2. Considerando o projeto MVC desktop e a tela apresentada, faça com que os dados inseridos sejam persistidos na BD via EclipseLink.



**Exercícios de Revisão**

1. Foi desenvolvido um projeto Java web (Faça o download do projeto no Blackboard: Exercício MVC ) e sabe-se que as layers no MVC são:

1. View;

2. Control;

3. Model;

Esse projeto tem os seguintes arquivos:

a) Um formulário HTML;

b) Um servlet que recebe requisições e responde às requisições;

c) Uma classe chamada Pessoa (Classe Java com os métodos getter’s/setter’s e um construtor padrão);

d) Uma classe PessoaDAO para a conexão com uma base de dados.

Relacione os arquivos às camadas corretas.

Considere o exemplo apresentado em aula, relacione os arquivos na tabela e corrija o relacionamento da coluna um (Camada) com a coluna três (Responsabilidade), posicionando cada responsabilidade à camada correta.

| Camada | Arquivos | Responsabilidade |
| --- | --- | --- |
| Controle |  | Apresentar resultados, conteúdos, mensagens etc. |
| Modelo |  | Recebe as requisições e direciona as ações para o destino correto. |
| Visão |  | Contém as fórmulas de cálculos, as regras de negócios, assim como as restrições. |

1. Com base no exercício anterior, escreva uma instrução para cada item:
2. Instrução que evidencia o acoplamento entre as camadas visão e controle.
3. Instrução que evidencia o acoplamento entre as camadas controle e modelo.
4. Uma classe interna pode ter modificado private? Mostre um exemplo prático e funcional.
5. Responda às questões referentes à classe anônima.
6. O que é uma classe anônima?
7. Escreva um exemplo prático e funcional.
8. O que são expressões lambdas? Escreva um exemplo prático e funcional.
9. Uma interface funcional deve ter obrigatoriamente a anotação @FunctionalInterface? O que caracteriza uma interface funcional? Cite duas interfaces funcionais nativas do Java. Explique o que é um interface funcional mencionando o uso da referida anotação.
10. É correto afirmar que um padrão arquitetural tem como base um design pattern, do catálogo GoF? Explique, relacionando arquitetura de software e design pattern.
11. Elabore um projeto Java que realize o mapeamento objeto-relacional através de um framework adequado (Sugestão: Utilize o EclipseLink). O seu projeto deve conter:
12. A unidade de persistência deve estar setada com a estratégia de criar uma tabela na base de dados estabelecida.
13. A classe da entidade Funcionario: id (int), nome, cargo (String), salario (double). Insira as anotações necessárias para torná-la visível à unidade de persistência e estabeleça o auto incremento para o id.
14. A classe FuncionarioDAO, com o CRUD, isto é, os métodos inserir, listar, remover e atualizar (Atualizar salário por id).
15. Na classe principal, teste todos os métodos de FuncionarioDAO. Em particular, o método listar deve ter o seu conteúdo iterado pelo método forEach da interface Iterable. Para isso, faça de três maneiras: 1) Escreva uma classe solicitada em forEach; 2) Utilize uma classe anônima; 3) Utilize uma expressão lambda.
16. Modifique a forma de estabelecer a conexão (FuncionarioDAO) para que forneça uma conexão apenas.