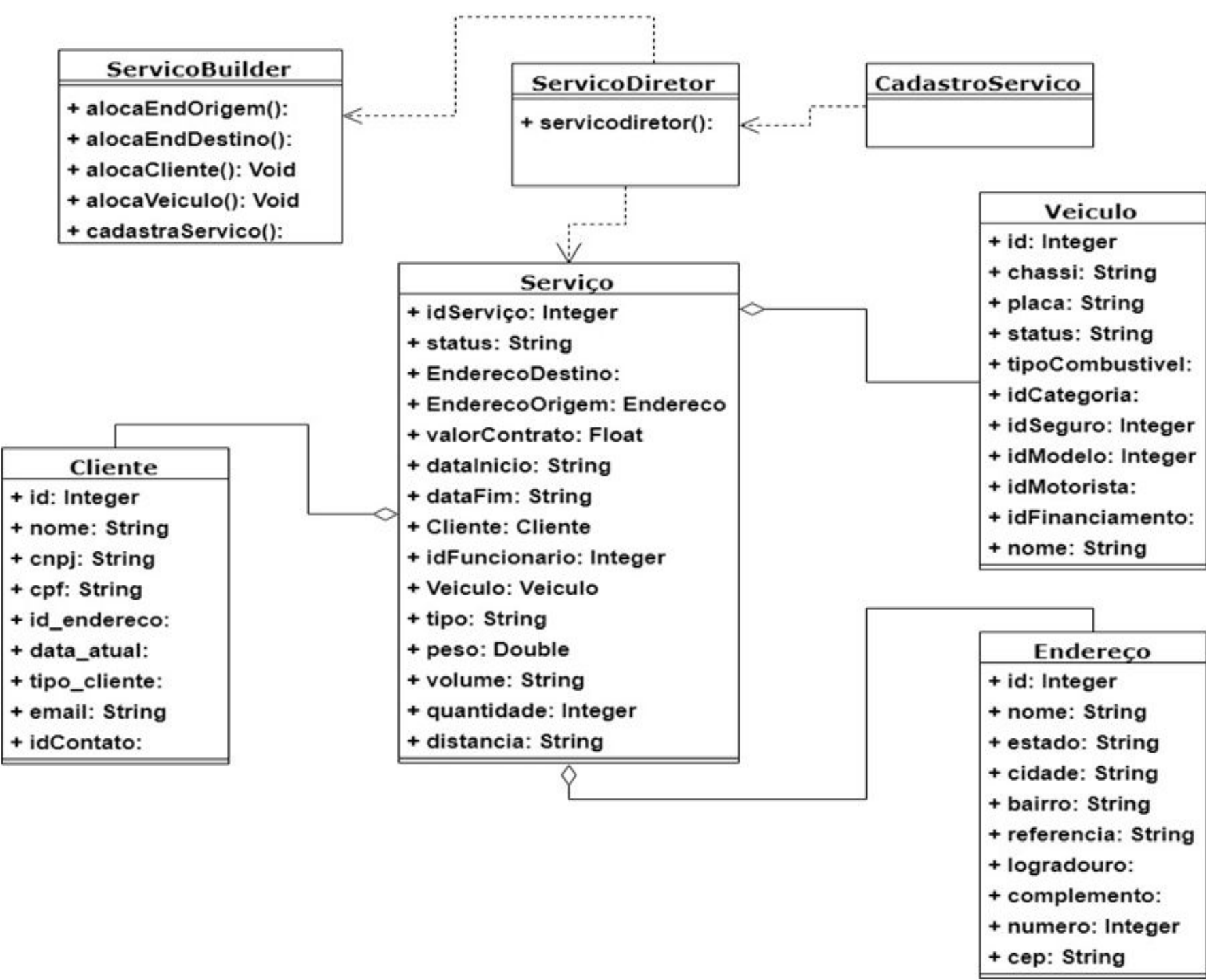


# CARGA PESADA

Wallace Silva, Leonardo Laia Arpini e Bruno Moreto

# PADRÃO BUILDER

- Utilizamos o padrão builder na criação dos serviços, já que é utilizado vários objetos na criação do próprio serviço.



# PADRÃO BUILDER

```
public class ServicoBuilder {  
  
    Servico servico;  
    Cliente cliente;  
    Endereco enderecoOrig;  
    Endereco enderecoDest;  
    Veiculo veiculo;  
  
    public void setServico(Servico servico) {  
        this.servico = servico;  
    }  
  
    public void setCliente(Cliente cliente) {  
        this.cliente = cliente;  
    }  
  
    public void setEnderecoOrig(Endereco enderecoOrig) {  
        this.enderecoOrig = enderecoOrig;  
    }  
}
```

# PADRÃO BUILDER

```
public void setEnderecoDest(Endereco enderecoDest) {  
    this.enderecoDest = enderecoDest;  
}
```

```
public void setVeiculo(Veiculo veiculo) {  
    this.veiculo = veiculo;  
}
```

```
public ServicoBuilder() {  
}
```

```
public void alocaEndOrigem() {  
    servico.setEnderecoOrigem(enderecoOrig);  
}
```

```
public void alocaEndDestino() {  
    servico.setEnderecoDestino(enderecoDest);  
}
```

# PADRÃO BUILDER

```
public void alocaEndDestino() {  
    servico.setEnderecoDestino(enderecoDest);  
}  
  
public void alocaCliente () {  
    servico.setCliente(cliente);  
}  
  
public void alocaVeiculo() {  
    servico.setVeiculo(veiculo);  
}  
  
public Servico cadastraServico() throws SQLException {  
    AplicacoesBD aplic = new AplicacoesBD();  
    aplic.cadastraServico(servico);  
    return servico;  
}
```

# PADRÃO BUILDER - DIRETOR

```
import java.sql.SQLException;
import modelo.Servico;

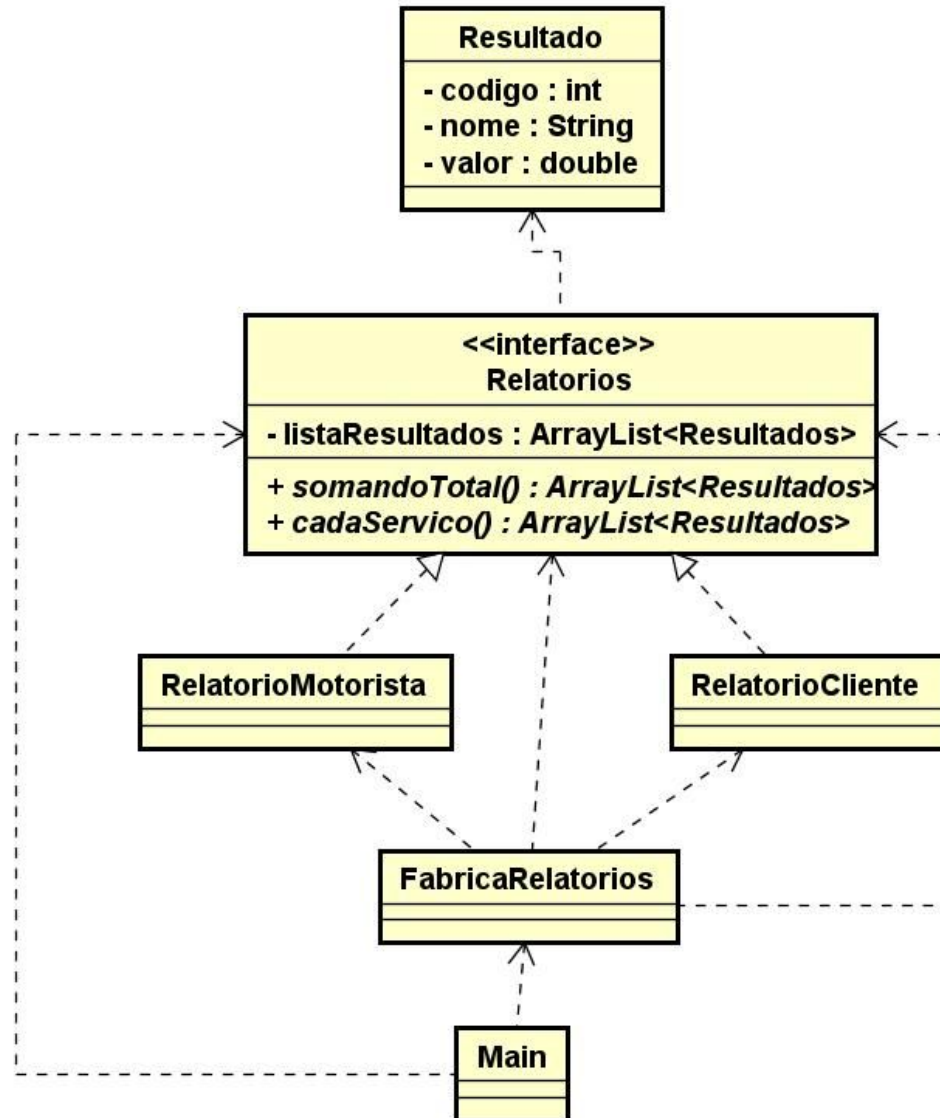
/**
 *
 * @author walla
 */
public class ServicoDiretor {
    public Servico ServicoDiretor(ServicoBuilder build) throws SQLException{
        build.alocaCliente();
        build.alocaEndDestino();
        build.alocaEndOrigem();
        build.alocaVeiculo();
        return build.cadastraServico();
    }
}
```

# PADRÃO FÁBRICA

- Utilizamos o padrão fábrica na criação dos nossos relatórios, para evitar a criação de vários objetos diferentes para cada relatório e para facilitar na criação de novos relatórios.



pkg



# PADRÃO FABRICA

```
public interface Relatorio {  
    ArrayList <Resultado> lista_resultado = new ArrayList<Resultado>();  
    public ArrayList relatorioTotal() throws SQLException;  
    public ArrayList relatorioTodos() throws SQLException;  
}
```

# PADRÃO FABRICA

```
public class RelatorioMotorista implements Relatorio{


    @Override
    public ArrayList relatorioTotal() throws SQLException{
        Connection c;
        Statement stmt;
        c = ConexaoBD.getInstance();
        stmt = c.createStatement();
        ResultSet rs;
        rs = stmt.executeQuery("select  mt.id, fun.nome, sum(valor_contrato) as valor_total\n" +
            "from servico as sr inner join motorista as mt on(sr.id_motorista = mt.id)\n" +
            "inner join funcionario as fun on(fun.id = mt.id)\n" +
            "group by mt.id, fun.nome;");
        while (rs.next()) {
            Resultado resultado = new Resultado();
            resultado.setCodigo(rs.getInt("ID"));
            resultado.setNome(rs.getString("NOME"));
            resultado.setValor(rs.getFloat("VALOR_TOTAL"));
            lista_resultado.add(resultado);
        }
        rs.close();
        stmt.close();
        c.close();
        return lista_resultado;
    }
}
```


# PADRÃO FABRICA


```
@Override
public ArrayList relatorioTodos() throws SQLException {
    Connection c;
    Statement stmt;
    c = ConexaoBD.getInstance();
    stmt = c.createStatement();
    ResultSet rs;
    rs = stmt.executeQuery("select mt.id, fun.nome, valor_contrato as valor\n" +
        "from servico as sr inner join motorista as mt on(sr.id_motorista = mt.id)\n" +
        "inner join funcionario as fun on(fun.id = mt.id);");
    while (rs.next()) {
        Resultado resultado = new Resultado();
        resultado.setCodigo(rs.getInt("ID"));
        resultado.setNome(rs.getString("NOME"));
        resultado.setValor(rs.getFloat("VALOR"));
        lista_resultado.add(resultado);
    }
    rs.close();
    stmt.close();
    c.close();
    return lista_resultado;
}
```


# PADRÃO FABRICA


```
public class FabricaRelatorios {  
    Relatorio relatorios;  
  
    public Relatorio geraRelatorio (String tipo){  
  
        if (tipo == "motorista"){  
            relatorios = new RelatorioMotorista();  
        }  
        else if (tipo == "cliente"){  
            relatorios = new RelatorioCliente();  
        }  
        return relatorios;  
    }  
}
```

 Home

 Serviços

 Veículos

 Funcionários

 Clientes

Relatorios

RELATORIOS

☒ TOTAL DE SERVIÇOS

☐ SERVIÇOS SEPARADOS

☒ CLIENTES

☐ MOTORISTAS

GERAR

ID	Nome	Valor
1	ADRIANI	100.0

versao 1.1

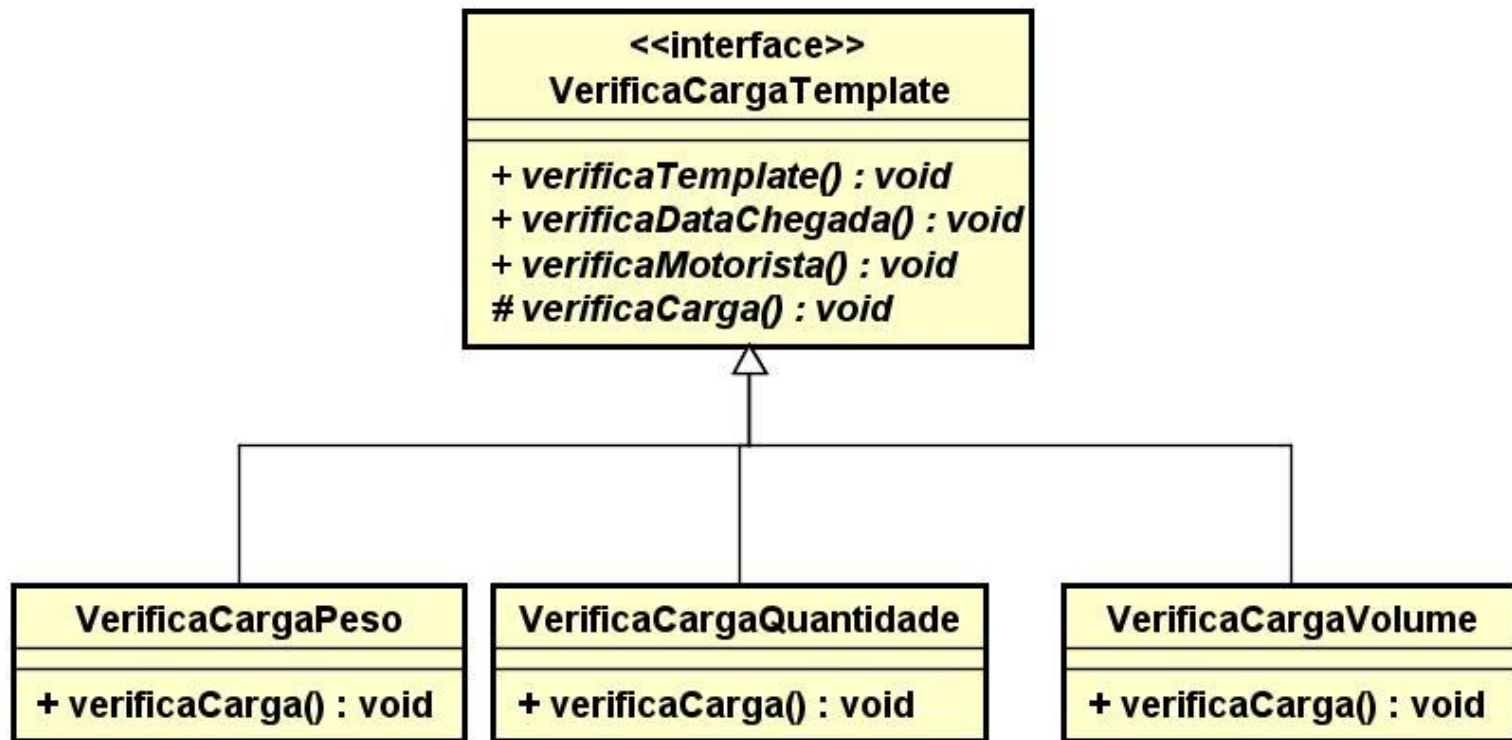
```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {  
    // TODO add your handling code here:  
    ArrayList <Resultado> lista = null;  
    Relatorio relatorio = null;  
    FabricaRelatorios fabrica = new FabricaRelatorios();  
    TABELA tab = new TABELA();  
  
    if(CLIENTE.isSelected()){  
        relatorio = fabrica.geraRelatorio("cliente");  
    }  
    else if(MOTORISTAS.isSelected()){  
        relatorio = fabrica.geraRelatorio("motorista");  
    }  
  
    if(SOMANDO.isSelected()){  
        try {  
            lista = relatorio.relatorioTotalizandoValor();  
        } catch (SQLException ex) {  
            Logger.getLogger(Relatorios.class.getName()).log(Level.SEVERE, null, ex);  
        }  
    }  
    else if(CADA.isSelected()){  
        try {  
            lista = relatorio.relatorioDeCadaServico();  
        } catch (SQLException ex) {  

```

# PADRÃO TEMPLATE

- Utilizamos o padrão template na criação das nossas verificações de carga, pois assim podemos criar um padrão(template) da ordem que a verificação da carga ocorre ao mesmo tempo em que podemos alterar os métodos da verificação que se diferem nos tipos de carga(volume, quantidade, peso) nas subclasses.





```
public abstract class VerificaCargaTemplate {

    AplicacoesBD aplicacao = new AplicacoesBD();
    ArrayList<Servico> servicoL;
    ArrayList<Funcionario> funcionarioL;
    SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy");
    int resultado;

    public int getResultado() {
        return resultado;
    }

    public final void VerificaTemplate(Servico servico, double carga, String dataServ, String nomeMoto) throws ParseException{
        String data = servico.getDataFim();
        Date dataFim = formato.parse(data);
        Date dataForm = formato.parse(dataServ);
        this.verificaDataChegada(dataFim, dataForm);

        String funcionarioID = Integer.toString(servico.getIdFuncionario());
        funcionarioL = aplicacao.ProcuraFuncionarioID(funcionarioID);
        Funcionario funcionario = funcionarioL.get(0);
        String nome = funcionario.getNome();
        this.verificaMotorista(nome, nomeMoto);
        //this.verificaEnderecoEntrega();
        this.verificaCarga(servico, carga);
    }
}
```

```

protected void verificaDataChegada(Date dataFim, Date data) throws ParseException {
    //String data = JOptionPane.showInputDialog("Insira a data atual.\n");
    //Date dataFormatada = formato.parse(data);
    if (data.after(dataFim)){
        JOptionPane.showMessageDialog(null,"Entrega em atraso");
    }else{
        JOptionPane.showMessageDialog(null,"Entrega dentro do prazo");
        resultado++;
    }
}
}

```

```

protected void verificaMotorista(String nome, String nomeMoto) {
    //String nomeMoto = JOptionPane.showInputDialog("Insira o nome do motorista.\n");
    if(nomeMoto.equalsIgnoreCase(nome)){
        JOptionPane.showMessageDialog(null,"Motorista verificado.");
        resultado++;
    }else{
        JOptionPane.showMessageDialog(null,"Motorista diferentes, verificar com central");
    }
}
}

```

```

//protected void verificaEnderecoEntrega() {}

```

```

protected abstract boolean verificaCarga(Servico servico, double carga);

```

```

}

```

```
public class VerificaCargaVolume extends VerificaCargaTemplate{

    ArrayList<Servico> servicoL;
    @Override
    protected boolean verificaCarga(Servico servico, double volume) {
        double volumeServ = servico.getVolume();
        //double volume = Integer.parseInt(JOptionPane.showInputDialog("Digite o volume constatado na chegada\n"));
        if (volume == volumeServ){
            JOptionPane.showMessageDialog(null,"Volume iguais, carga confirmada.\n");
            resultado++;
            return true;
        } else {
            JOptionPane.showMessageDialog(null,"Volume diferentes, carga rejeitada.\nRelatar Central");
        }
        return false;
    }
}
```

```
public class VerificaCargaQuantidade extends VerificaCargaTemplate{

    ArrayList<Servico> servicoL;
    protected boolean verificaCarga(Servico servico, double quantidade) {
        int quantidadeServ = servico.getQuantidade();
        //int quantidade = Integer.parseInt(JOptionPane.showInputDialog("Digite a carga constatado na chegada\n"));
        if (quantidade == quantidadeServ){
            JOptionPane.showMessageDialog(null,"Quantidade iguais, carga confirmada.\n");
            resultado++;
            return true;
        } else {
            JOptionPane.showMessageDialog(null,"Quantidades diferentes, carga rejeitada.\nRelatar Central.");
        }
        return false;
    }
}
```

```
public class VerificaCargaPeso extends VerificaCargaTemplate{

    ArrayList<Servico> servicoL;
    @Override
    protected boolean verificaCarga(Servico servico, double peso) {

        double pesoServ = servico.getQuantidade();
        //double peso = Integer.parseInt(JOptionPane.showInputDialog("Digite o peso constatado na chegada\n"));
        if (peso == pesoServ){
            JOptionPane.showMessageDialog(null,"Peso iguais, carga confirmada.\n");
            resultado++;
            return true;
        } else {
            JOptionPane.showMessageDialog(null,"Peso diferentes, carga rejeitada.\nRelatar Central.");
        }
        return false;
    }
}
```

## Cadastro de Serviço

Cliente

Veículo

Endereço

BUCA

Ca

Tipo

liquid

Dados

Data

Valor Contrato

×

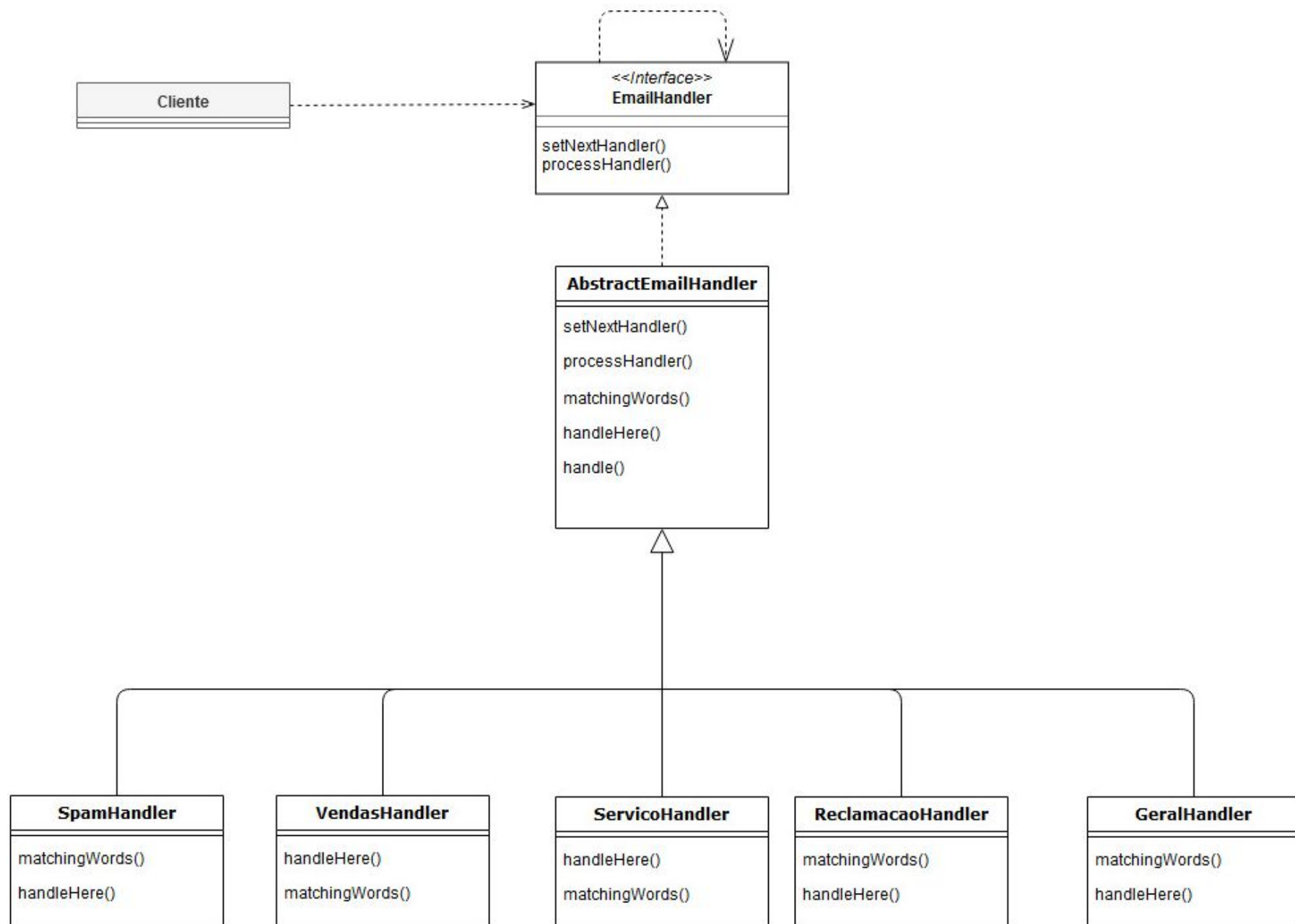
### Verificação de Carga

<b>Nome Motorista</b>	<b>Data Chegada</b>
<input type="text"/>	<input type="text" value="--"/>
<b>Carga</b>	
<input type="text"/>	
<input type="button" value="Verificar"/>	<input type="button" value="Voltar"/>

# PADRÃO CADEIA DE RESPONSABILIDADE

- Utilizamos o padrão cadeia de responsabilidade para criação dos tratamentos de e-mails, para evitar que as áreas percam tempo recebendo e-mails que não sejam de seu interesse, fazendo com que o trabalho seja feito de forma mais eficiente.





```
package cadeiaResponsabilidade;

public interface EmailHandler {
    public void setNextHandler(EmailHandler handler);
    public void processHandler(String email);
}
```

---

```
public abstract class AbstractEmailHandler implements EmailHandler{
    private EmailHandler nextHandler;

    @Override
    public void setNextHandler(EmailHandler handler){
        this.nextHandler = handler;
    }

    @Override
    public void processHandler(String email){
        boolean wordFound = false;

        for (String word : matchingWords()){
            if (email.indexOf(word) >= 0){
                wordFound = true;
                break;
            }
        }
        if (wordFound){
            handleHere(email);
        }
        else{
            nextHandler.processHandler(email);
        }
    }

    protected abstract String[] matchingWords();
    protected abstract void handleHere(String email);
}
```

```

    public static void handle (String email){
        EmailHandler spam = new SpamHandler();
        EmailHandler vendas = new VendasHandler();
        EmailHandler servico = new ServicoHandler();
        EmailHandler reclamacao = new ReclamacaoHandler();
        EmailHandler geral = new GeralHandler();

        spam.setNextHandler(vendas);
        vendas.setNextHandler(servico);
        servico.setNextHandler(reclamacao);
        reclamacao.setNextHandler(geral);

        spam.processHandler(email);
    }
}

```

```

public class SpamHandler extends AbstractEmailHandler{
    @Override
    protected String[] matchingWords(){
        return new String[]{"medicina", "promoção", "grátis", "marketing", "multinível"};
    }

    @Override
    protected void handleHere(String email) {
        System.out.println("Email sendo enviado para caixa de SPAM");
    }
}

```

```
public class VendasHandler extends AbstractEmailHandler{  
    @Override  
    protected String[] matchingWords(){  
        return new String[]{"valor","contratar","orçamento"};  
    }  
  
    @Override  
    protected void handleHere(String email) {  
        System.out.println("Email sendo cuidado pelo departamento de vendas");  
    }  
}
```

---

```
public class ServicoHandler extends AbstractEmailHandler{  
    @Override  
    protected String[] matchingWords(){  
        return new String[]{"serviço","andamento"};  
    }  
  
    @Override  
    protected void handleHere(String email) {  
        System.out.println("Email sendo cuidado pelo Gerente de Serviços");  
    }  
}
```

```
public class ReclamacaoHandler extends AbstractEmailHandler{
    @Override
    protected String[] matchingWords(){
        return new String[]{"sugestão", "reclamação", "atrazo", "faltando"};
    }

    @Override
    protected void handleHere(String email) {
        System.out.println("Email sendo cuidado pelo departamento de reclamações");
    }
}
```

```
public class GeralHandler extends AbstractEmailHandler{
    @Override
    protected String[] matchingWords(){
        return new String[]{};
    }

    @Override
    protected void handleHere(String email) {
        System.out.println("Email sendo cuidado pelo call center");
    }
}
```

---