# Sistema CRM – Padrão Facade

**Disciplina:** Engenharia de Software II – 2025.1
**Trabalho:** Padrões de Projeto no Acoplamento e Coesão
**Aluno:** Dário Ygor de Sousa Leal
**Professor:** Prof. João Paulo
**Data:** 17/08/2025

# Introdução

Empresas que utilizam CRM lidam com atividades como captação e qualificação de leads, conversão em clientes, registro de interações e gestão de oportunidades. Expor diretamente todos os serviços à camada cliente aumenta o acoplamento e dificulta a evolução. Este trabalho demonstra como o padrão *Facade* encapsula a orquestração entre subsistemas (validação, pontuação, lead, cliente, oportunidade, comunicação e auditoria), reduzindo acoplamento e elevando coesão. As funcionalidades são simuladas por mensagens no console, suficientes para fins didáticos.

# Desenvolvimento

## Diagrama UML (Português)

A Figura 1 apresenta a visão geral do sistema CRM com o padrão Facade aplicado.

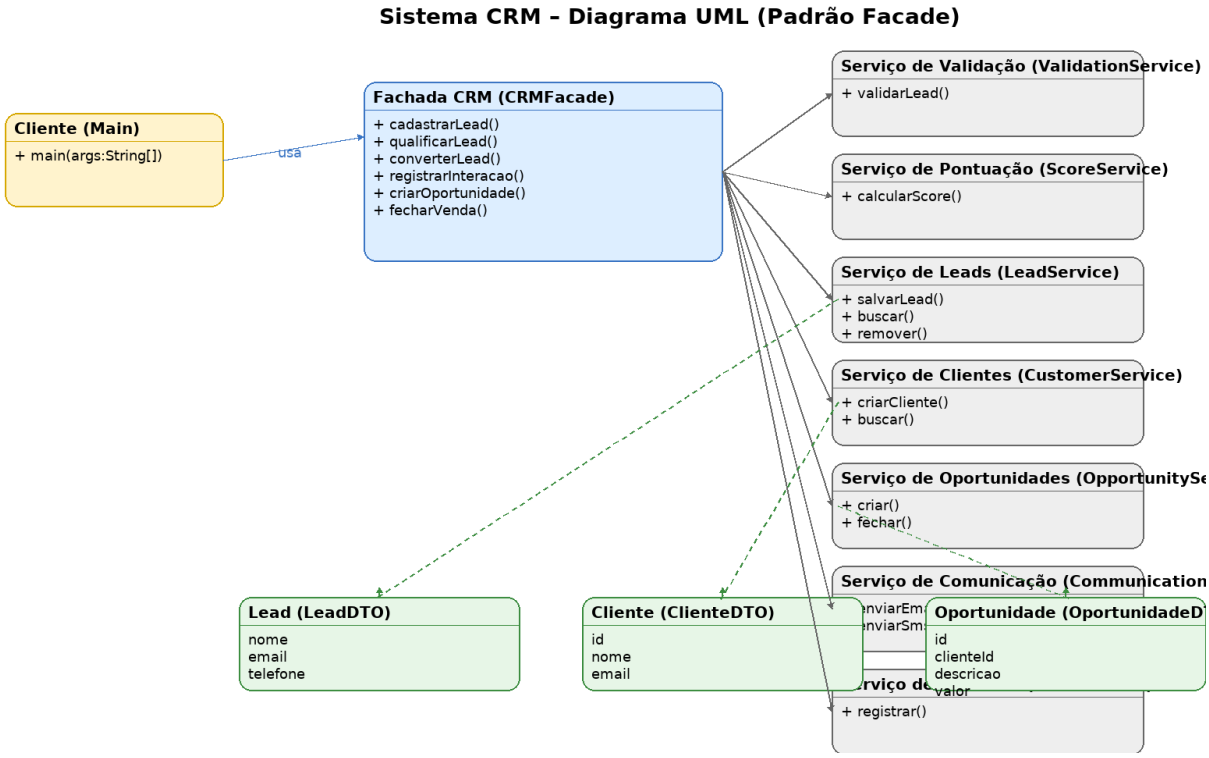


*Figura 1 – Diagrama UML do CRM com Facade.*

## Fachada (CRMFacade)

```
public class CRMFacade {
    private final ValidationService validation;
    private final ScoreService score;
```

```
    private final LeadService lead;
    private final CustomerService customer;
    private final OpportunityService opportunity;
    private final CommunicationService comms;
    private final AuditService audit;

    public CRMFacade(ValidationService validation, ScoreService score,
                     LeadService lead, CustomerService customer,
                     OpportunityService opportunity, CommunicationService comms,
                     AuditService audit) {
        this.validation = validation;
        this.score = score;
        this.lead = lead;
        this.customer = customer;
        this.opportunity = opportunity;
        this.comms = comms;
        this.audit = audit;
    }

    public String cadastrarLead(LeadDTO novoLead) {
        validation.validarLead(novoLead);
        int sc = score.calcularScore(novoLead);
        String leadId = lead.salvarLead(novoLead);
        audit.registrar("CADASTRAR_LEAD", "leadId=" + leadId + ", score=" + sc);
        comms.enviarEmail(novoLead.email, "Bem-vindo!",
            "Olá " + novoLead.nome + ", recebemos seu contato. Em breve falaremos com você.");
        return leadId;
    }

    public boolean qualificarLead(String leadId, int minimoScore) {
        LeadDTO l = lead.buscar(leadId);
        if (l == null) return false;
        int sc = score.calcularScore(l);
        boolean qualificado = sc >= minimoScore;
        audit.registrar("QUALIFICAR_LEAD", "leadId=" + leadId + ", score=" + sc +
                ", qualificado=" + qualificado);
        return qualificado;
    }

    public String converterLeadEmCliente(String leadId) {
        LeadDTO l = lead.buscar(leadId);
        if (l == null) throw new IllegalArgumentException("Lead não encontrado");
        String clienteId = customer.criarCliente(l);
        audit.registrar("CONVERTER_LEAD", "leadId=" + leadId + " -> clienteId=" + clienteId);
        comms.enviarEmail(l.email, "Conta criada",
            "Olá " + l.nome + ", criamos sua conta de cliente. Obrigado!");
        lead.remover(leadId);
        return clienteId;
    }

    public void registrarInteracao(String clienteId, String canal, String conteudo) {
        ClienteDTO c = customer.buscar(clienteId);
        if (c == null) throw new IllegalArgumentException("Cliente inválido");
        audit.registrar("INTERACAO", "clienteId=" + clienteId + ", canal=" + canal);
        if ("EMAIL".equalsIgnoreCase(canal)) {
            comms.enviarEmail(c.email, "Resposta do Suporte", conteudo);
        } else if ("SMS".equalsIgnoreCase(canal)) {
            comms.enviarSms(c.email, conteudo); // simulando
        } else {
            System.out.println("[Interação] Canal: " + canal + " | " + conteudo);
        }
    }

    public String criarOportunidade(String clienteId, String descricao, double valor) {
        String opId = opportunity.criar(clienteId, descricao, valor);
        audit.registrar("CRIAR_OPORTUNIDADE", "clienteId=" + clienteId + ", opId=" + opId);
        return opId;
    }

    public void fecharVenda(String oportunidadeId) {
        opportunity.fechar(oportunidadeId);
        audit.registrar("FECHAR_VENDA", "opId=" + oportunidadeId);
    }
}
```

A **Fachada** concentra a orquestração dos serviços. O cliente chama operações de alto nível como *cadastrarLead*, *qualificarLead*, *converterLead*, entre outras, sem conhecer a ordem das chamadas internas nem os serviços envolvidos.

## DTOs (LeadDTO, ClienteDTO, OportunidadeDTO)

```
public class LeadDTO {
    public String nome;
    public String email;
    public String telefone;
    public LeadDTO(String nome, String email, String telefone) {
        this.nome = nome; this.email = email; this.telefone = telefone;
    }
}

public class ClienteDTO {
    public String id; public String nome; public String email;
    public ClienteDTO(String id, String nome, String email) {
        this.id = id; this.nome = nome; this.email = email;
    }
}

public class OportunidadeDTO {
    public String id; public String clienteId; public String descricao; public double valor;
    public OportunidadeDTO(String id, String clienteId, String descricao, double valor) {
        this.id = id; this.clienteId = clienteId; this.descricao = descricao; this.valor = valor;
    }
}
```

Os **DTOs** transportam dados simples entre os serviços e a fachada, mantendo baixo acoplamento com as regras de negócio.

## Serviços (Validação, Pontuação, Lead, Cliente, Oportunidade, Comunicação, Auditoria)

```
public class ValidationService {
    public void validarLead(LeadDTO lead) {
        if (lead.nome == null || lead.nome.isBlank())
            throw new IllegalArgumentException("Nome inválido");
        if (lead.email == null || !lead.email.contains("@"))
            throw new IllegalArgumentException("Email inválido");
        System.out.println("[Validation] Lead válido: " + lead.nome);
    }
}

public class ScoreService {
    public int calcularScore(LeadDTO lead) {
        int score = 50;
        if (lead.email.endsWith("@empresa.com")) score += 30;
        if (lead.telefone != null && lead.telefone.startsWith("+55")) score += 20;
        System.out.println("[Score] Score do lead " + lead.nome + ": " + score);
        return score;
    }
}

import java.util.*;
public class LeadService {
    private final Map<String, LeadDTO> leads = new HashMap<>();
    public String salvarLead(LeadDTO lead) {
        String id = "L" + (leads.size()+1);
        leads.put(id, lead);
        System.out.println("[Lead] Lead salvo com id=" + id);
        return id;
    }
    public LeadDTO buscar(String leadId) { return leads.get(leadId); }
    public void remover(String leadId) { leads.remove(leadId); }
}

import java.util.*;
public class CustomerService {
    private final Map<String, ClienteDTO> clientes = new HashMap<>();
    public String criarCliente(LeadDTO lead) {
        String id = "C" + (clientes.size()+1);
        ClienteDTO c = new ClienteDTO(id, lead.nome, lead.email);
        clientes.put(id, c);
        System.out.println("[Cliente] Cliente criado id=" + id + " (" + c.nome + ")");
        return id;
    }
    public ClienteDTO buscar(String clienteId) { return clientes.get(clienteId); }
}
```

```
import java.util.*;
public class OpportunityService {
    private final Map<String, OportunidadeDTO> oportunidades = new HashMap<>();
    public String criar(String clienteId, String desc, double valor) {
        String id = "O" + (oportunidades.size()+1);
        OportunidadeDTO op = new OportunidadeDTO(id, clienteId, desc, valor);
        oportunidades.put(id, op);
        System.out.println("[Oportunidade] Criada id=" + id + " para cliente=" + clienteId);
        return id;
    }
    public void fechar(String oportunidadeId) {
        System.out.println("[Oportunidade] Oportunidade " + oportunidadeId + " FECHADA!");
    }
}

public class CommunicationService {
    public void enviarEmail(String para, String assunto, String corpo) {
        System.out.println("[Email] Para: " + para + " | Assunto: " + assunto + " | Corpo: " + corpo);
    }
    public void enviarSms(String para, String texto) {
        System.out.println("[SMS] Para: " + para + " | " + texto);
    }
}

public class AuditService {
    public void registrar(String evento, String detalhe) {
        System.out.println("[Audit] " + evento + " -> " + detalhe);
    }
}
```

Cada **serviço** possui responsabilidade única (coesão alta). A fachada coordena o uso desses serviços, evitando que a camada cliente dependa de múltiplas interfaces (redução de acoplamento).

## Cliente (Main)

```
public class Main {
    public static void main(String[] args) {
        CRMFacade facade = new CRMFacade(
            new ValidationService(),
            new ScoreService(),
            new LeadService(),
            new CustomerService(),
            new OpportunityService(),
            new CommunicationService(),
            new AuditService()
        );

        String leadId = facade.cadastrarLead(new LeadDTO("Alex", "alex@empresa.com", "+5599999999"));

        if (facade.qualificarLead(leadId, 80)) {
            String clienteId = facade.converterLeadEmCliente(leadId);
            facade.registrarInteracao(clienteId, "EMAIL", "Olá! Podemos agendar uma demonstração?");
            String opId = facade.criarOportunidade(clienteId, "Plano Anual CRM", 4999.90);
            facade.fecharVenda(opId);
        } else {
            System.out.println("[Main] Lead não alcançou score mínimo.");
        }
    }
}
```

A classe **Main** representa o cliente: utiliza apenas a *CRMFacade*. Qualquer mudança interna nos serviços não afeta o cliente, desde que a interface da fachada permaneça estável.

# Discussão: Acoplamento × Coesão

Sem a fachada, a camada cliente teria que conhecer *todos* os serviços e a sequência correta de chamadas, acarretando **alto acoplamento** e maior esforço de manutenção. Com a *CRMFacade*, existe uma **porta de entrada única**; o cliente depende apenas dela. Ao mesmo tempo, cada serviço mantém **coesão alta** por possuir responsabilidade bem definida (validação, pontuação, persistência etc.). Esse desenho facilita testes, evolução e substituição de serviços internos sem impacto no cliente.

# Considerações finais

Aplicar o padrão *Facade* em um cenário de CRM centraliza fluxos, simplifica a utilização pela camada cliente e preserva a coesão dos serviços. A solução resultante é mais evolutiva, de baixo acoplamento e com melhor testabilidade. Como trabalho futuro, pode-se integrar *Billing*, *Catálogo* e *Relatórios* por novas fachadas ou subfachadas, mantendo a mesma filosofia de orquestração e encapsulamento.

# Referências

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
Enunciado da atividade: Padrões de Projeto no Acoplamento e Coesão (IFPI, 2025.1).