TRABALHO POO-HOTEL ESTADIA CONFORTAVEL

-Alunos:

João Guilherme R R Da Cunha

Enzo Andrade Alves

David Freire Carvalho

-Link do Git Hub:

https://github.com/JGuih15/hotelPOO

1.O que cada aluno fez no projeto:

Este projeto foi desenvolvido colaborativamente por um grupo de três pessoas, com as seguintes responsabilidades:

Membro 1 - Arquitetura e Modelos de Domínio

Responsável pela estruturação das classes principais do sistema, implementação da hierarquia de quartos utilizando herança e polimorfismo, e definição das entidades fundamentais (Hospede, Reserva, Servico).

Membro 2 - Lógica de Controle e Regras de Negócio

Responsável pela implementação da classe ControleHotel, tratamento de exceções personalizadas, gerenciamento do fluxo de reservas, check-in/check-out, e implementação do programa de fidelidade.

Membro 3 - Interface de Usuário e Persistência

Responsável pela criação do menu interativo na classe Main, implementação da camada de persistência de dados via serialização, e testes de integração do sistema completo.

2.Funcionalidades implementadas:

Funcionalidades Principais:

1. Gerenciamento de Quartos

- 10 quartos disponíveis divididos em três categorias:

    - 4 Quartos Standard (101-104): Diária de R$ 100,00
    - 4 Quartos Luxo (201-204): Diária de R$ 225,00 (preço base R$ 150 + 50%)
    - 2 Quartos Suíte (301-302): Diária de R$ 500,00 (preço base R$ 250 + 100%)
- Listagem de quartos disponíveis em tempo real
- Controle automático de ocupação

2. Sistema de Check-in

- Cadastro automático de novos hóspedes (CPF + Nome)
- Busca de hóspedes já cadastrados
- Validação de disponibilidade de quartos
- Definição da quantidade de dias de estadia
- Tratamento de erros para reservas inválidas

3. Serviços de Quarto (Room Service)

O sistema oferece um menu com 4 serviços:

- Café da Manhã Completo: R$ 35,00
- Jantar Executivo: R$ 60,00
- Lavanderia (5 peças): R$ 45,00
- Massagem Terapêutica: R$ 120,00

Benefício VIP: Hóspedes VIP recebem 10% de desconto em todos os serviços.

4. Programa de Fidelidade

Sistema completo de pontuação:

- Acúmulo: 1 ponto para cada R$ 10,00 gastos
- Resgate: 10 pontos = R$ 1,00 de desconto
- Limite de desconto: Máximo de 50% do valor da fatura
- Status VIP: Concedido automaticamente ao atingir 500 pontos
- Consulta de pontos disponíveis por CPF

5. Sistema de Check-out e Pagamento

Processo completo de finalização:

1. Cálculo automático do valor total (diárias + serviços)
2. Aplicação de desconto VIP (10% nos serviços)
3. Exibição de fatura preliminar detalhada
4. Opção de resgate de pontos de fidelidade
5. Cálculo do valor final com descontos aplicados
6. Liberação automática do quarto
7. Acúmulo de novos pontos baseado no valor gasto

6. Consultas e Relatórios

- Listagem de todas as reservas ativas
- Consulta individual de programa de fidelidade por hóspede
- Histórico de estadias (armazenado)

7. Persistência de Dados

- Salvamento automático após cada operação crítica
- Utilização de serialização de objetos Java
- Arquivo binário dados_hotel.dat
- Recuperação automática ao reiniciar o sistema
- Todo o estado do hotel é preservado entre execuções

3.Experiencia do aluno:

Tivemos dificuldades em relação ao encapsulamento, mas a principal dificuldade foi na classe geradora de pdf(relatório), tivemos problemas ao importar a biblioteca de leitura e geração.

Entretante, durante o projeto, para resolver o problema com encapsulamentos conseguimos aprimorar nossa noção em torno desse pilar de POO. Infelizmente não conseguimos resolver o problema do gerador de pdf, então retiramos ele do código.

4.Bibliotecas utilizadas e motivos:

java.util - Estruturas de Dados e Utilitários:

Armazenar coleções dinâmicas de objetos (quartos, reservas, serviços)

Permitir adição/remoção de elementos durante a execução

Iteração sobre elementos com facilidade

java.io - Entrada/Saída e Persistência:

Permitir que objetos sejam convertidos em bytes

Essencial para salvar o estado completo do sistema em arquivo

Todas as classes de modelo implementam para manter integridade

java.time - Manipulação de Datas:

Registrar a data de check-in de forma moderna e segura

API mais robusta que java.util.Date (legado)

Imutabilidade garante thread-safety

Facilita cálculos de datas se necessário no futuro

5. **Referencias**:

-Nos baseamos principalmente em experiencias individuas de aprendizagem em sala de aula, cuja as quais, juntas conseguiram resolver a maioria dos problemas

-Usamos também IAS para orientação em problemas, cujo os quais, não conseguimos resolver.

6.Screenshot(Classe de controle de hotel):

```
package control;

import exceptions.ReservaInvalidaException;
import model.*;
import utils.PersistenciaDados;


import java.io.Serializable;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class ControleHotel implements Serializable {
  private static final long serialVersionUID = 1L;
  private List<Quarto> quartos;
  private Map<String, Hospede> hospedes; // CPF como chave
  private List<Reserva> reservasAtivas;
  private List<Reserva> historicoEstadias;
  private List<Servico> menuServicos;

  public ControleHotel() {
    this.quartos = new ArrayList<>();
    this.hospedes = new HashMap<>();
    this.reservasAtivas = new ArrayList<>();
    this.historicoEstadias = new ArrayList<>();
    this.menuServicos = new ArrayList<>();
    inicializarDados();
  }

  private void inicializarDados() {
    for (int i = 101; i <= 104; i++) {
      quartos.add(new QuartoStandard(String.valueOf(i)));
    }
    for (int i = 201; i <= 204; i++) {
      quartos.add(new QuartoLuxo(String.valueOf(i)));
    }
```

```
    for (int i = 301; i <= 302; i++) {
      quartos.add(new QuartoSuite(String.valueOf(i)));
    }

    menuServicos.add(new Servico("Café da Manhã Completo", 35.00));
    menuServicos.add(new Servico("Jantar Executivo", 60.00));
    menuServicos.add(new Servico("Lavanderia (5 peças)", 45.00));
    menuServicos.add(new Servico("Massagem Terapêutica", 120.00));
  }

  public void cadastrarHospede(Hospede h) {
    if (!hospedes.containsKey(h.getCpf())) {
      hospedes.put(h.getCpf(), h);
      System.out.println("[INFO] Hóspede " + h.getNome() + " cadastrado com
sucesso.");
    } else {
      System.err.println("[ERRO] Hóspede com CPF " + h.getCpf() + " já
cadastrado.");
    }
  }

  public Quarto buscarQuarto(String numero) {
    return quartos.stream()
        .filter(q -> q.getNumero().equals(numero))
        .findFirst()
        .orElse(null);
  }

  public Hospede buscarHospede(String cpf) {
    return hospedes.get(cpf);
  }

  public Reserva buscarReservaPorQuarto(String numeroQuarto) {
    return reservasAtivas.stream()
        .filter(r -> r.getQuarto().getNumero().equals(numeroQuarto))
        .findFirst()
        .orElse(null);
  }


  public void realizarCheckin(Hospede h, Quarto q, int diasEstadia) throws
```

```
ReservaInvalidaException {
    if (q.isOcupado()) {
      throw new ReservaInvalidaException("Quarto " + q.getNumero() + " está
ocupado.");
    }
    if (diasEstadia <= 0) {
      throw new ReservaInvalidaException("O número de dias da estadia deve
ser maior que zero.");
    }

    if (!hospedes.containsKey(h.getCpf())) {
      cadastrarHospede(h);
    }

    Reserva novaReserva = new Reserva(h, q, diasEstadia);
    reservasAtivas.add(novaReserva);
    PersistenciaDados.salvar(this); // Salva o estado após o check-in
    System.out.println("\n[SUCESSO] Check-in realizado para " + h.getNome()
+ " no Quarto " + q.getNumero());
  }

  public void adicionarServicoAQuarto(String numeroQuarto, Servico servico)
throws ReservaInvalidaException {
    Reserva r = buscarReservaPorQuarto(numeroQuarto);
    if (r == null) {
      throw new ReservaInvalidaException("Não há reserva ativa no Quarto " +
numeroQuarto);
    }

    r.adicionarServico(servico);
    PersistenciaDados.salvar(this);
    System.out.println("[SUCESSO] Serviço '" + servico.getNome() + "'
adicionado ao Quarto " + numeroQuarto);
  }
  public double realizarCheckout(String numeroQuarto, int pontosParaTrocar)
throws ReservaInvalidaException {
    Reserva reserva = buscarReservaPorQuarto(numeroQuarto);

    if (reserva == null) {
      throw new ReservaInvalidaException("Nenhuma reserva ativa
encontrada no Quarto " + numeroQuarto);
```

```
    }

    double valorFinal = reserva.calcularValorTotal();
    Hospede hospede = reserva.getHospede();

    if (pontosParaTrocar > 0) {
      int descontoMaximo = (int) (valorFinal * 0.5);
      int pontosMaximo = descontoMaximo * 10;

      int pontosUteis = Math.min(pontosParaTrocar,
hospede.getPontosFidelidade());
      pontosUteis = Math.min(pontosUteis, pontosMaximo);

      if (hospede.trocarPontosPorDesconto(pontosUteis)) {
        double valorDesconto = pontosUteis / 10.0;
        valorFinal -= valorDesconto;
        System.out.printf("[FIDELIDADE] Desconto de R$ %.2f aplicado com
%d pontos.\n", valorDesconto, pontosUteis);
      } else {
        System.out.println("[FIDELIDADE] Pontos insuficientes ou valor final
muito baixo para troca.");
      }
    }

    hospede.acumularPontos(valorFinal);

    reserva.getQuarto().setOcupado(false);
    reservasAtivas.remove(reserva);
    historicoEstadias.add(reserva);

    PersistenciaDados.salvar(this);


    System.out.println("[SUCESSO] Check-out do Quarto " + numeroQuarto + "
finalizado.");

    return valorFinal;
  }

  public List<Quarto> getQuartosDisponiveis() {
    return quartos.stream()
```

```
            .filter(q -> !q.isOcupado())
            .collect(Collectors.toList());
    }

    public List<Reserva> getReservasAtivas() {
        return reservasAtivas;
    }

    public List<Servico> getMenuServicos() {
        return menuServicos;
    }

    public Map<String, Hospede> getHospedes() {
        return hospedes;
    }

    public List<Reserva> getHistoricoEstadias() {
        return historicoEstadias;
    }
}
```