Universidade Anhanguera-Uniderp - Unidade Vila Mariana

Marcel dos Santos Gubany – R.A. 7117508501

Marcus Yoshihiro Takano – RA: 6574300998

Plínio Balduino – R.A. 6950459640

Roberta Vieira Justino Balduino – R.A. 6950459481

ATPS – PROGRAMAÇÃO ORIENTADA A OBJETOS

Desenvolvimento de um sistema de hospedagem em Java

São Paulo/SP

09/2014

Índice

1 Apresentação 4

1.1 Orientação a objetos 4

1.2 Princípios SOLID 4

1.3 Ferramentas 5

1.3.1 Java 5

1.3.2 Maven 5

1.3.3 Eclipse 6

2 Relatório 1 7

2.1 Descrição 7

2.2 Código fonte 7

2.2.1 App.java 7

2.2.2 Booking.java 8

2.2.3 DefaultPeriodCalculator.java 9

2.2.4 ReceiptPrinter 10

2.2.5 KeyboardReader.java 11

2.2.6 DefaultPeriodCalculatorTest 12

2.2.7 ReceiptPrinterTest 12

3 Relatório 2 14

3.1 Descrição 14

3.2 Código fonte 14

3.2.1 App.java 14

3.2.2 Booking.java 16

3.2.3 ReceiptPrinter 18

3.2.4 Room.java 18

3.2.5 RoomList.java 19

3.2.6 KeyboardReader.java 20

4 Relatório 3 22

4.1 Descrição 22

4.2 Código fonte 22

4.2.1 Room.java 22

4.2.2 StandardRoom.java 22

4.2.3 Cottage.java 23

4.2.4 RoomList.java 23

4.2.5 RoomType.java 25

5.2.6 Booking.java 25

4.2.7 App.java 27

5 Relatório 4 30

5.1 Descrição 30

5.2 Código fonte 30

5.2.1 App.java 30

5.2.2 RoomType.java 32

5.2.3 RoomList.java 33

5.2.4 FancyRoom.java 34

# 1 Apresentação

O objetivo desta atividade é apresentar uma aplicação escrita na linguagem Java que sirva para controlar as hospedagens de quartos da localidade turística de Maraú, situada no Estado da Bahia.

A aplicação funciona através do console, mas pode ser facilmente estendida para trabalhar através de uma interface web ou mesmo uma interface gráfica para desktop independente de sistema operacional.

Foram utilizados os princípios SOLID, disseminados por Robert “Uncle Bob” Martin, autor do obrigatório livro “Clean Code”, onde são listadas as boas práticas no uso de programação orientada a objetos.

Foram utilizadas as versões 1.8.0 do Java e 3.2.1 do Maven. O código não é compatível com versões anteriores do Java, tendo em vista o uso da nova API de datas para o cálculo de diárias.

## 1.1 Orientação a objetos

A Programação Orientada a Objetos é uma das mais populares formas de se abordar problemas computacionais e foi implementada pela primeira vez como a linguagem Simula, em 1967.

O paradigma da orientação a objetos estabelece que um problema pode ser resolvido fazendo uso de objetos especializados que trabalham em conjunto para chegar a uma solução.

Dentro do contexto do Java, um objeto é a instância de uma classe, que por sua vez é uma abstração de algo existente no mundo real.

## 1.2 Princípios SOLID

SOLID é um acrônimo em inglês para os cinco princípios básicos da orientação a objetos descritos por Robert “Uncle Bob” Martin.

São eles:

S – Single Responsability Principle – Uma classe deve ter uma e apenas uma responsabilidade, ou seja, apenas uma motivação para ser modificada durante a evolução do sistema.

O – Open/Closed Principle – Objetos e classes devem ser abertos para extensão, mas fechados para modificação, ou seja, através de injeção de dependências ou uso consciente de herança, o comportamento de um objeto ou classe pode ser aprimorado, sem que seu código fonte original seja alterado.

L – Princípio de Liskov – Objetos podem ser substituídos por instâncias de seus subtipos sem que isso quebre a aplicação. Esse princípio indica que podemos utilizar uma classe Cachorro onde é esperado Animal, sabendo eu um Cachorro é um Animal, sem que isso invalide a aplicação

I – Interface Segregation Principle – Classes e objetos devem ter poucos comportamentos. Classes com muitos comportamentos acabam sendo difíceis de dar manutenção e passam a ter contato com todo o sistema.

D – Dependency Inversion Principle – Devemos depender de abstrações, e não de implementações específicas, já que as primeiras mudam menos e podem ser aplicadas em diferentes casos, aumentando o reuso de código.

## 1.3 Ferramentas

### 1.3.1 Java

Java é uma linguagem criada pela Sun, atualmente propriedade da Oracle, que se propôs a ser substituta do C++, além de ser multiplataforma.

Apesar de ter herdado vários dos problemas de design do C++, é muito utilizada em ambientes corporativos e para o ensino de Orientação a Objetos.

Para esta atividade está sendo utilizada a versão 1.8.0 do Java, conhecida como Java 8.

### 1.3.2 Maven

Maven é um gerenciador de dependências e automatizador de tarefas criado para facilitar o desenvolvimento de aplicações em Java.

Com o tempo surgiram diversas alternativas que se propõe a simplificar o trabalho feito pelo Maven, todas com resultados questionáveis.

### 1.3.3 Eclipse

Ambiente de desenvolvimento gratuito para a linguagem Java. Apesar de ser mais avançado que seu concorrente NetBeans, hoje em dia tem perdido popularidade para o concorrente mais novo IntelliJ.

# 2 Relatório 1

## 2.1 Descrição

De acordo com o enunciado:

“Fazer uma classe para representar o aluguel de um quarto (dica: armazene as informações de data e horário de entrada e saída separadas, ou seja, um campo inteiro para o dia de entrada, um campo inteiro para o mês de entrada, um campo inteiro para o ano de entrada, um campo inteiro para o horário de entrada etc.). Esta classe deve conter métodos para calcular o número de diárias e o valor a ser pago pelo locatário.”

De acordo com o “Princípio da Responsabilidade Única”, tomamos a liberdade de não deixar as informações de hospedagem, cálculos de diárias e impressão de recibo na mesma classe.

Foi decidido de comum acordo que é uma boa prática observar que cada objeto deve ter apenas uma motivação e ser alterado em apenas um ponto.

Não concordamos também com a quebra das datas de entrada e saída em campos separados, uma vez que o Java provê tipos específicos para armazenamento de data (*Date* e *Calendar*), além de toda uma API para cálculos de datas, adicionada na versão 8 do Java.

O pacote principal da aplicação é *ead2.poo.atps.marau*, existindo pacotes internos de acordo com a necessidade de organização do código.

## 2.2 Código fonte

### 2.2.1 App.java

/\*\*

\* Classe que inicia a aplicação.

\*

\*/

package ead2.poo.atps.marau;

import java.io.IOException;

import java.text.ParseException;

import ead2.poo.atps.marau.util.KeyboardReader;

public class App {

public static void main(String[] args) throws Exception {

App app = new App();

app.addBooking();

}

/\*\*

\* Método que abre uma nova hospedagem e imprime o respectivo recibo

\*/

private void addBooking() throws IOException, ParseException {

Booking booking = getBoookingData();

ReceiptPrinter.print(booking);

}

private Booking getBoookingData() throws IOException, ParseException {

System.out.println("= Reserva de quarto =");

System.out.println();

String renterName = KeyboardReader.readLineAsString("Nome do locatario: ");

String checkin = KeyboardReader.readLineAsString("Data de entrada : ", "dd/MM/yyyy hh:mm");

String checkout = KeyboardReader.readLineAsString("Data de saída : ", "dd/MM/yyyy hh:mm");

int guests = KeyboardReader.readLineAsInteger("Acompanhantes : ");

double dailyRate = KeyboardReader.readLineAsDouble("Valor da diária : ");

return new Booking(renterName, checkin, checkout, guests, dailyRate);

}

}

### 2.2.2 Booking.java

/\*\*

\*

\* Classe responsável por armazenar as informações de hospedagem

\*

\*/

package ead2.poo.atps.marau;

import java.text.ParseException;

public class Booking {

private String renterName;

private String checkin;

private String checkout;

private int guests;

private double dailyRate;

private int dailyCount;

public Booking(final String renterName, final String checkin, final String checkout, final int guests, final double dailyRate) throws ParseException {

this.renterName = renterName;

this.checkin = checkin;

this.checkout = checkout;

this.guests = guests;

this.dailyRate = dailyRate;

this.dailyCount = DefaultPeriodCalculator.countDays(this.checkin, this.checkout);

}

public String getRenterName() {

return renterName;

}

public String getCheckin() {

return checkin;

}

public String getCheckout() {

return checkout;

}

public int getDailyCount() {

return dailyCount;

}

public double getTotalRate() {

return dailyCount \* dailyRate;

}

}

### 2.2.3 DefaultPeriodCalculator.java

/\*\*

\*

\* Classe responsável pelo cálculo de dias de hospedagem

\*

\*/

package ead2.poo.atps.marau;

import java.text.DateFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.time.Duration;

import java.time.LocalDate;

import java.time.Period;

import java.time.ZoneId;

import java.time.temporal.ChronoUnit;

import java.util.Date;

import java.util.TimeZone;

public class DefaultPeriodCalculator {

private static final DateFormat DATE\_FORMATTER;

private static final ZoneId UTC\_ZONE = ZoneId.of("UTC");

static {

TimeZone.setDefault(TimeZone.getTimeZone("UTC"));

DATE\_FORMATTER = new SimpleDateFormat("dd/MM/yyyy hh:mm");

}

/\*\*

\* countDays

\* Retorna a quantidade de diárias entre a entrada e a saída do hóspede

\*

\* Para isso a função recebe as datas em formato texto, como

\* dd/mm/aaaa hh:mm", e as converte para o tipo Date. Em seguida são

\* subtraídas doze horas da data de entrada e são somadas doze horas na data

\* de saída. Então as horas são descartadas e é calculada a diferença de

\* dias entre saída e entrada.

\*/

public static int countDays(final String checkin, final String checkout) throws ParseException {

// calcular inicio da primeira diaria de fato

LocalDate checkinDay = dailyStart(DATE\_FORMATTER.parse(checkin));

// calcular fim da ultima diaria de fato

LocalDate checkoutDay = dailyEnd(DATE\_FORMATTER.parse(checkout));

// calcular diferença entre as duas datas

Period period = Period.between(LocalDate.from(checkinDay), LocalDate.from(checkoutDay));

return period.getDays();

}

private static LocalDate dailyStart(Date checkin) {

return checkin.toInstant()

.minus(Duration.ofHours(12))

.truncatedTo(ChronoUnit.DAYS)

.atZone(UTC\_ZONE)

.toLocalDate();

}

private static LocalDate dailyEnd(Date checkout) {

return checkout.toInstant()

.plus(Duration.ofHours(12))

.truncatedTo(ChronoUnit.DAYS)

.atZone(UTC\_ZONE)

.toLocalDate();

}

}

### 2.2.4 ReceiptPrinter

package ead2.poo.atps.marau;

import java.io.PrintStream;

public class ReceiptPrinter {

private static PrintStream out = System.out;

public static void setPrintStream(final PrintStream out) {

ReceiptPrinter.out = out;

}

public static void print(Booking booking) {

out.println("========================================");

out.printf("Locatário: %s%n", booking.getRenterName());

out.printf("Data e horário de entrada: %s%n", booking.getCheckin());

out.printf("Data e horário de saída: %s%n", booking.getCheckout());

out.printf("Número de diárias: %d%n", booking.getDailyCount());

out.printf("Total devido: %.2f%n", booking.getTotalRate());

out.println("========================================");

}

}

### 2.2.5 KeyboardReader.java

package ead2.poo.atps.marau.util;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.text.ParseException;

import java.text.SimpleDateFormat;

public final class KeyboardReader {

private KeyboardReader() {

}

public static String readLineAsString(final String text) throws IOException {

System.out.print(text);

BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));

return keyboard.readLine();

}

public static int readLineAsInteger(final String text) throws IOException {

int result = 0;

while(true) {

String line = readLineAsString(text);

try {

result = Integer.parseInt(line);

break;

} catch (NumberFormatException e) {

System.out.println("Informe um número inteiro");

}

}

return result;

}

public static double readLineAsDouble(String text) throws IOException {

double result = 0;

while(true) {

String line = readLineAsString(text);

try {

result = Double.parseDouble(line.replace(',', '.'));

break;

} catch (NumberFormatException e) {

System.out.println("Formato incorreto. Você pode usar ponto ou vírgula para separar os centavos");

}

}

return result;

}

public static String readLineAsString(String text, String format) throws IOException {

String line;

while(true) {

line = readLineAsString(text);

try {

new SimpleDateFormat(format).parse(line);

break;

} catch (ParseException e) {

System.out.println("Data inválida. Use o formato " + format);

}

}

return line;

}

}

### 2.2.6 DefaultPeriodCalculatorTest

package ead2.poo.atps.marau;

import static org.junit.Assert.assertEquals;

import java.text.ParseException;

import org.junit.Test;

public class DefaultPeriodCalculatorTest {

@Test

public void firstPeriodSample() throws ParseException {

assertEquals(3, DefaultPeriodCalculator.countDays("12/07/2012 13:42", "15/07/2012 11:50"));

}

@Test

public void secondPeriodSample() throws ParseException {

assertEquals(4, DefaultPeriodCalculator.countDays("12/07/2012 17:00", "15/07/2012 16:50"));

}

}

### 2.2.7 ReceiptPrinterTest

package ead2.poo.atps.marau;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.OutputStream;

import java.io.PrintStream;

import java.text.ParseException;

import org.junit.Before;

import org.junit.Test;

public class ReceiptPrinterTest {

@Before

public void setUp() throws FileNotFoundException {

ReceiptPrinter.setPrintStream(new NullPrintStream());

}

@Test

public void printingTest() throws ParseException {

Booking booking = new Booking("Jeremias", "12/07/2012 13:42", "15/07/2012 11:50", 0, 200);

ReceiptPrinter.print(booking);

}

private static class NullPrintStream extends PrintStream {

public NullPrintStream() {

super(new OutputStream() {

@Override

public void write(int value) throws IOException {

// do nothing

}

});

}

}

}

# 3 Relatório 2

## 3.1 Descrição

De acordo com o enunciado, “Como as hospitalidades e quartos oferecidos aos turistas têm o mesmo padrão, os moradores decidiram estabelecer o mesmo valor de diária. Sabendo-se disso, modifiquem o programa da etapa anterior para que o atributo referente ao valor da diária seja compartilhado entre todas as instâncias da classe que representa o aluguel de um quarto”

Apenas os arquivos que foram modificados ou incluídos serão exibidos neste relatório.

## 3.2 Código fonte

### 3.2.1 App.java

/\*\*

\* Classe que inicia a aplicação.

\*

\*/

package ead2.poo.atps.marau;

import java.io.IOException;

import java.text.ParseException;

import ead2.poo.atps.marau.util.KeyboardReader;

public final class App {

private static final char EXIT = '0';

private static final char CHECKIN = '1';

private static final char CHECKOUT = '2';

private static final char UPDATE\_DAILY\_RATE = '3';

private final RoomList rooms;

public App(RoomList rooms) {

this.rooms = rooms;

}

public static void main(String[] args) throws Exception {

RoomList rooms = new RoomList();

rooms.setSize(25);

App app = new App(rooms);

app.systemParameters();

app.menu();

}

private void menu() throws IOException, ParseException {

int option = 99;

do {

option = printMenu();

switch (option) {

case EXIT:

System.out.println();

System.out.println("Até logo.");

break;

case CHECKIN:

doCheckin();

break;

case CHECKOUT:

doCheckout();

break;

case UPDATE\_DAILY\_RATE:

systemParameters();

break;

default:

System.out.println("Opção inválida");

KeyboardReader.readKey();

break;

}

} while (option != EXIT);

}

private int printMenu() throws IOException {

clearScreen();

System.out.println("= Menu do sistema =");

System.out.println(" 1 - Registrar entrada");

System.out.println(" 2 - Registrar saída");

System.out.println(" 3 - Atualizar valor de diária");

System.out.println(" -----------------------------");

System.out.println(" 0 - Sair");

System.out.println();

System.out.printf(" = Diária padrão: %.2f%n", Booking.STANDARD\_DAILY\_RATE);

System.out.println();

System.out.println(" = Ocupação =");

System.out.println(rooms);

return KeyboardReader.readKey();

}

private void clearScreen() {

for(int i = 0; i < 1000; i++) {

System.out.println();

}

}

private void systemParameters() throws IOException {

System.out.println("= Parâmetros do sistema =");

Booking.STANDARD\_DAILY\_RATE = KeyboardReader.readLineAsDouble("Valor padrão da diária : ");

System.out.println();

}

/\*\*

\* Método que abre uma nova hospedagem

\*/

private void doCheckin() throws IOException, ParseException {

rooms.addBooking(getBoookingData());

}

private void doCheckout() throws IOException, ParseException {

System.out.println();

System.out.println("= Saída do quarto =");

int roomNumber = KeyboardReader.readLineAsInteger("Informe o número do quarto (1 - " + rooms.count() + "): ");

if(!rooms.isEmpty(roomNumber)) {

String checkout = KeyboardReader.readLineAsString("Data de saída : ", "dd/MM/yyyy hh:mm");

Booking booking = rooms.checkout(roomNumber, checkout);

ReceiptPrinter.print(booking, roomNumber);

} else {

System.out.println("Quarto está vazio");

}

KeyboardReader.readKey();

}

private Booking getBoookingData() throws IOException, ParseException {

System.out.println("= Reserva de quarto =");

String renterName = KeyboardReader.readLineAsString("Nome do locatario: ");

String checkin = KeyboardReader.readLineAsString("Data de entrada : ", "dd/MM/yyyy hh:mm");

int guests = KeyboardReader.readLineAsInteger("Acompanhantes : ");

System.out.println();

return new Booking(renterName, checkin, guests);

}

}

### 3.2.2 Booking.java

/\*\*

\*

\* Classe responsável por armazenar as informações de hospedagem

\*

\*/

package ead2.poo.atps.marau;

import java.text.ParseException;

public class Booking {

private final String renterName;

private final String checkin;

private String checkout;

private final int guests;

private int dailyCount;

public static double STANDARD\_DAILY\_RATE;

public Booking(final String renterName, final String checkin, final int guests) throws ParseException {

this(renterName, checkin, null, guests);

}

public Booking(final String renterName, final String checkin, final String checkout, final int guests) throws ParseException {

this.renterName = renterName;

this.checkin = checkin;

this.checkout = checkout;

this.guests = guests;

if(isCheckedOut()) {

this.dailyCount = DefaultPeriodCalculator.countDays(this.checkin, this.checkout);

}

}

public String getRenterName() {

return renterName;

}

public String getCheckin() {

return checkin;

}

public String getCheckout() {

return checkout;

}

public int getDailyCount() {

return dailyCount;

}

public double getTotalRate() {

return dailyCount \* STANDARD\_DAILY\_RATE;

}

public boolean isCheckedOut() {

return checkout != null;

}

public void setCheckout(String checkout) throws ParseException {

this.checkout = checkout;

this.dailyCount = DefaultPeriodCalculator.countDays(this.checkin, this.checkout);

}

}

### 3.2.3 ReceiptPrinter

package ead2.poo.atps.marau;

import java.io.PrintStream;

public class ReceiptPrinter {

private static PrintStream out = System.out;

public static void setPrintStream(final PrintStream out) {

ReceiptPrinter.out = out;

}

public static void print(Booking booking, int roomNumber) {

out.println("========================================");

out.printf("Locatário: %s%n", booking.getRenterName());

out.printf("Quarto número: %d%n" , roomNumber);

out.printf("Data e horário de entrada: %s%n", booking.getCheckin());

out.printf("Data e horário de saída: %s%n", booking.getCheckout());

out.printf("Número de diárias: %d%n", booking.getDailyCount());

out.printf("Total devido: %.2f%n", booking.getTotalRate());

out.println("========================================");

}

}

### 3.2.4 Room.java

package ead2.poo.atps.marau;

import java.text.ParseException;

public class Room {

public final static Room EMPTY = new Room(null);

private Booking booking;

public Room(Booking booking) {

this.booking = booking;

}

public boolean isEmpty() {

return booking == null;

}

public String toString() {

return isEmpty() ? "\_Vazio\_" : "Ocupado";

}

public Booking checkout(String checkout) throws ParseException {

booking.setCheckout(checkout);

return booking;

}

}

### 3.2.5 RoomList.java

package ead2.poo.atps.marau;

import java.text.ParseException;

import java.util.ArrayList;

import java.util.List;

public class RoomList {

private final List<Room> rooms;

public RoomList() {

rooms = new ArrayList<Room>();

}

public void setSize(final int roomQuantity) {

for(int room = 0; room < roomQuantity; room++) {

rooms.add(Room.EMPTY);

}

}

public int addBooking(Booking booking) {

int roomNumber = findFreeRoom();

rooms.set(roomNumber - 1, new Room(booking));

return roomNumber;

}

private int findFreeRoom() {

int i = -1;

boolean found = false;

for(i = 0; !found && i < rooms.size(); i++) {

if(rooms.get(i).isEmpty()) {

found = true;

}

}

return found ? i : -1;

}

public String toString() {

StringBuffer buffer = new StringBuffer();

for(int room = 0; room < count(); room++) {

buffer.append("[Quarto ")

.append(room < 9 ? "0" : "")

.append(room + 1)

.append(": ")

.append(rooms.get(room))

.append("] ")

.append(room > 0 && room % 5 == 0 ? "\n" : "");

}

return buffer.toString();

}

public int count() {

return rooms.size();

}

public boolean isEmpty(final int roomNumber) {

return (roomNumber - 1 > 0 && roomNumber - 1 < count() ? rooms.get(roomNumber - 1).isEmpty() : true);

}

public Booking checkout(int roomNumber, String checkout) throws ParseException {

Booking booking = rooms.get(roomNumber - 1).checkout(checkout);

rooms.set(roomNumber - 1, Room.EMPTY);

return booking;

}

}

### 3.2.6 KeyboardReader.java

package ead2.poo.atps.marau.util;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.text.ParseException;

import java.text.SimpleDateFormat;

public final class KeyboardReader {

private KeyboardReader() {

}

public static String readLineAsString(final String text) throws IOException {

System.out.print(text);

BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));

return keyboard.readLine();

}

public static int readLineAsInteger(final String text) throws IOException {

int result = 0;

while(true) {

String line = readLineAsString(text);

try {

result = Integer.parseInt(line);

break;

} catch (NumberFormatException e) {

System.out.println("Informe um número inteiro");

}

}

return result;

}

public static double readLineAsDouble(String text) throws IOException {

double result = 0;

while(true) {

String line = readLineAsString(text);

try {

result = Double.parseDouble(line.replace(',', '.'));

break;

} catch (NumberFormatException e) {

System.out.println("Formato incorreto. Você pode usar ponto ou vírgula para separar os centavos");

}

}

return result;

}

public static String readLineAsString(String text, String format) throws IOException {

String line;

while(true) {

line = readLineAsString(text);

try {

new SimpleDateFormat(format).parse(line);

break;

} catch (ParseException e) {

System.out.println("Data inválida. Use o formato " + format);

}

}

return line;

}

public static int readKey() throws IOException {

BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));

return keyboard.read();

}

}

# 4 Relatório 3

## 4.1 Descrição

De acordo com o enunciado: “Com o aumento no fluxo de turistas, a Associação de Moradores de Maraú decidiu construir uma pousada com 50 quartos (numerados de 1 a 50) e 10 chalés (numerados de 51 a 60). O aluguel dos quartos tem o mesmo esquema de cobrança definido nas etapas anteriores. Já o aluguel dos chalés tem um custo da tarifa 20% maior que o dos quartos e permite que sejam incluídas camas extras. Cada cama extra tem o custo de 10% do valor de uma diária. Modificar o programa feito na etapa anterior para criar uma classe que represente o aluguel de um chalé que tem os mesmos dados dos quartos e, além destes, armazena o número de camas extras solicitadas. Criar uma classe para representar a pousada e nela defina um único vetor capaz de armazenar as instancias das acomodações que estejam ocupadas”.

## 4.2 Código fonte

### 4.2.1 Room.java

package ead2.poo.atps.marau.rooms;

import java.text.ParseException;

import ead2.poo.atps.marau.Booking;

public abstract class Room {

private Booking booking;

protected Room(Booking booking) {

this.booking = booking;

if(!isEmpty()) {

booking.setRoom(this);

}

}

public final boolean isEmpty() {

return booking == null;

}

@Override

public final String toString() {

return getRoomType() + (isEmpty() ? " \_Vazio\_" : " Ocupado");

}

public final Booking checkout(String checkout) throws ParseException {

booking.setCheckout(checkout);

return booking;

}

public abstract RoomType getRoomType();

public abstract double luxuryRate();

}

### 4.2.2 StandardRoom.java

package ead2.poo.atps.marau.rooms;

import ead2.poo.atps.marau.Booking;

public class StandardRoom extends Room {

public final static Room EMPTY = new StandardRoom(null);

public StandardRoom(Booking booking) {

super(booking);

}

@Override

public RoomType getRoomType() {

return RoomType.STANDARD;

}

@Override

public double luxuryRate() {

return 1.0;

}

}

### 4.2.3 Cottage.java

package ead2.poo.atps.marau.rooms;

import ead2.poo.atps.marau.Booking;

public class Cottage extends Room {

public final static Room EMPTY = new Cottage(null);

private Booking booking;

public Cottage(Booking booking) {

super(booking);

this.booking = booking;

}

@Override

public RoomType getRoomType() {

return RoomType.COTTAGE;

}

@Override

public double luxuryRate() {

return 1.2 + (0.1 \* (double)booking.getGuests());

}

}

### 4.2.4 RoomList.java

package ead2.poo.atps.marau.rooms;

import java.io.IOException;

import java.text.ParseException;

import java.util.ArrayList;

import java.util.List;

import ead2.poo.atps.marau.Booking;

import ead2.poo.atps.marau.util.KeyboardReader;

public class RoomList {

private final List<Room> rooms;

public RoomList() {

rooms = new ArrayList<Room>();

}

public void setSize(final int standardRooms) {

for(int room = 0; room < standardRooms; room++) {

rooms.add(StandardRoom.EMPTY);

}

}

public void setSize(int standardRooms, int cottages) {

setSize(standardRooms);

for(int room = standardRooms; room < (standardRooms + cottages); room++) {

rooms.add(Cottage.EMPTY);

}

}

public int addBooking(Booking booking) throws IOException {

int roomNumber;

do {

roomNumber = KeyboardReader.readLineAsInteger("Número do quarto : ");

if(!isEmpty(roomNumber)) {

System.out.println("O quarto está ocupado");

}

}while(!isEmpty(roomNumber));

Room room = null;

switch (rooms.get(roomNumber - 1).getRoomType()) {

case STANDARD:

room = new StandardRoom(booking);

break;

case COTTAGE:

room = new Cottage(booking);

break;

}

rooms.set(roomNumber - 1, room);

return roomNumber;

}

public String toString() {

StringBuffer buffer = new StringBuffer();

for(int room = 0; room < count(); room++) {

buffer.append("[")

.append(room < 9 ? "0" : "")

.append(room + 1)

.append(": ")

.append(rooms.get(room))

.append("] ")

.append(room > 0 && (room + 1) % 5 == 0 ? "\n" : "");

}

return buffer.toString();

}

public int count() {

return rooms.size();

}

public boolean isEmpty(final int roomNumber) {

return (roomNumber - 1 > 0 && roomNumber - 1 < count() ? rooms.get(roomNumber - 1).isEmpty() : true);

}

public Booking checkout(int roomNumber, String checkout) throws ParseException {

Booking booking = rooms.get(roomNumber - 1).checkout(checkout);

rooms.set(roomNumber - 1, StandardRoom.EMPTY);

return booking;

}

}

### 4.2.5 RoomType.java

package ead2.poo.atps.marau.rooms;

public enum RoomType {

STANDARD {

@Override

public String toString() {

return "Padrão";

}

},

COTTAGE {

@Override

public String toString() {

return "Chalé ";

}

}

}

### 5.2.6 Booking.java

/\*\*

\*

\* Classe responsável por armazenar as informações de hospedagem

\*

\*/

package ead2.poo.atps.marau;

import java.text.ParseException;

import ead2.poo.atps.marau.rooms.Room;

public class Booking {

private final String renterName;

private final String checkin;

private String checkout;

private final int guests;

private int dailyCount;

private Room room;

public static double STANDARD\_DAILY\_RATE;

public Booking(final String renterName, final String checkin, final int guests) throws ParseException {

this(renterName, checkin, null, guests);

}

public Booking(final String renterName, final String checkin, final String checkout, final int guests) throws ParseException {

this.renterName = renterName;

this.checkin = checkin;

this.checkout = checkout;

this.guests = guests;

if(isCheckedOut()) {

this.dailyCount = DefaultPeriodCalculator.countDays(this.checkin, this.checkout);

}

}

public String getRenterName() {

return renterName;

}

public String getCheckin() {

return checkin;

}

public String getCheckout() {

return checkout;

}

public int getDailyCount() {

return dailyCount;

}

public double getTotalRate() {

return dailyCount \* STANDARD\_DAILY\_RATE \* room.luxuryRate();

}

public boolean isCheckedOut() {

return checkout != null;

}

public void setCheckout(String checkout) throws ParseException {

this.checkout = checkout;

this.dailyCount = DefaultPeriodCalculator.countDays(this.checkin, this.checkout);

}

public void setRoom(Room room) {

this.room = room;

}

public int getGuests() {

return guests;

}

}

### 4.2.7 App.java

/\*\*

\* Classe que inicia a aplicação.

\*

\*/

package ead2.poo.atps.marau;

import java.io.IOException;

import java.text.ParseException;

import ead2.poo.atps.marau.rooms.RoomList;

import ead2.poo.atps.marau.util.KeyboardReader;

public final class App {

private static final char EXIT = '0';

private static final char CHECKIN = '1';

private static final char CHECKOUT = '2';

private static final char UPDATE\_DAILY\_RATE = '3';

private final RoomList rooms;

public App(RoomList rooms) {

this.rooms = rooms;

}

public static void main(String[] args) throws Exception {

RoomList rooms = new RoomList();

rooms.setSize(50, 10);

App app = new App(rooms);

app.systemParameters();

app.menu();

}

private void menu() throws IOException, ParseException {

int option = 99;

do {

option = printMenu();

switch (option) {

case EXIT:

System.out.println();

System.out.println("Até logo.");

break;

case CHECKIN:

doCheckin();

break;

case CHECKOUT:

doCheckout();

break;

case UPDATE\_DAILY\_RATE:

systemParameters();

break;

default:

System.out.println("Opção inválida");

KeyboardReader.readKey();

break;

}

} while (option != EXIT);

}

private int printMenu() throws IOException {

clearScreen();

System.out.println("= Menu do sistema =");

System.out.println(" 1 - Registrar entrada");

System.out.println(" 2 - Registrar saída");

System.out.println(" 3 - Atualizar valor de diária");

System.out.println(" -----------------------------");

System.out.println(" 0 - Sair");

System.out.println();

System.out.printf(" = Diária padrão: %.2f%n", Booking.STANDARD\_DAILY\_RATE);

System.out.println();

System.out.println(" = Ocupação =");

System.out.println(rooms);

return KeyboardReader.readKey();

}

private void clearScreen() {

for(int i = 0; i < 1000; i++) {

System.out.println();

}

}

private void systemParameters() throws IOException {

System.out.println("= Parâmetros do sistema =");

Booking.STANDARD\_DAILY\_RATE = KeyboardReader.readLineAsDouble("Valor padrão da diária : ");

System.out.println();

}

/\*\*

\* Método que abre uma nova hospedagem

\*/

private void doCheckin() throws IOException, ParseException {

rooms.addBooking(getBoookingData());

}

private void doCheckout() throws IOException, ParseException {

System.out.println();

System.out.println("= Saída do quarto =");

int roomNumber = KeyboardReader.readLineAsInteger("Informe o número do quarto (1 - " + rooms.count() + "): ");

if(!rooms.isEmpty(roomNumber)) {

String checkout = KeyboardReader.readLineAsString("Data de saída : ", "dd/MM/yyyy hh:mm");

Booking booking = rooms.checkout(roomNumber, checkout);

ReceiptPrinter.print(booking, roomNumber);

} else {

System.out.println("Quarto está vazio");

}

KeyboardReader.readKey();

}

private Booking getBoookingData() throws IOException, ParseException {

System.out.println("= Reserva de quarto =");

String renterName = KeyboardReader.readLineAsString("Nome do locatario: ");

String checkin = KeyboardReader.readLineAsString("Data de entrada : ", "dd/MM/yyyy hh:mm");

int guests = KeyboardReader.readLineAsInteger("Camas extra : ");

System.out.println();

return new Booking(renterName, checkin, guests);

}

}

# 5 Relatório 4

## 5.1 Descrição

Do enunciado: “Com os lucros gerados, a pousada foi remodelada, de tal forma que metade dos quartos passou a ter banheira. Entretanto, o lucro não era suficiente para promover reformas nos chalés. Os quartos que possuem banheira têm tarifa 18% superior aos que não possuem e são numerados de 1 a 25. Sabendo-se disso, alterem o programa da etapa anterior, criando uma classe Aluguel que será a mais genérica e da qual as outras classes devem descender. Esta classe deve ter um método abstrato que deve retornar o valor a ser cobrado pelas diárias”.

## 5.2 Código fonte

### 5.2.1 App.java

/\*\*

\* Classe que inicia a aplicação.

\*

\*/

package ead2.poo.atps.marau;

import java.io.IOException;

import java.text.ParseException;

import ead2.poo.atps.marau.rooms.RoomList;

import ead2.poo.atps.marau.util.KeyboardReader;

public final class App {

private static final char EXIT = '0';

private static final char CHECKIN = '1';

private static final char CHECKOUT = '2';

private static final char UPDATE\_DAILY\_RATE = '3';

private final RoomList rooms;

public App(RoomList rooms) {

this.rooms = rooms;

}

public static void main(String[] args) throws Exception {

RoomList rooms = new RoomList();

rooms.setSize(25, 25, 10);

App app = new App(rooms);

app.systemParameters();

app.menu();

}

private void menu() throws IOException, ParseException {

int option = 99;

do {

option = printMenu();

switch (option) {

case EXIT:

System.out.println();

System.out.println("Até logo.");

break;

case CHECKIN:

doCheckin();

break;

case CHECKOUT:

doCheckout();

break;

case UPDATE\_DAILY\_RATE:

systemParameters();

break;

default:

System.out.println("Opção inválida");

KeyboardReader.readKey();

break;

}

} while (option != EXIT);

}

private int printMenu() throws IOException {

clearScreen();

System.out.println("= Menu do sistema =");

System.out.println(" 1 - Registrar entrada");

System.out.println(" 2 - Registrar saída");

System.out.println(" 3 - Atualizar valor de diária");

System.out.println(" -----------------------------");

System.out.println(" 0 - Sair");

System.out.println();

System.out.printf(" = Diária padrão: %.2f%n", Booking.STANDARD\_DAILY\_RATE);

System.out.println();

System.out.println(" = Ocupação =");

System.out.println(rooms);

return KeyboardReader.readKey();

}

private void clearScreen() {

for(int i = 0; i < 1000; i++) {

System.out.println();

}

}

private void systemParameters() throws IOException {

System.out.println("= Parâmetros do sistema =");

Booking.STANDARD\_DAILY\_RATE = KeyboardReader.readLineAsDouble("Valor padrão da diária : ");

System.out.println();

}

/\*\*

\* Método que abre uma nova hospedagem

\*/

private void doCheckin() throws IOException, ParseException {

rooms.addBooking(getBoookingData());

}

private void doCheckout() throws IOException, ParseException {

System.out.println();

System.out.println("= Saída do quarto =");

int roomNumber = KeyboardReader.readLineAsInteger("Informe o número do quarto (1 - " + rooms.count() + "): ");

if(!rooms.isEmpty(roomNumber)) {

String checkout = KeyboardReader.readLineAsString("Data de saída : ", "dd/MM/yyyy hh:mm");

Booking booking = rooms.checkout(roomNumber, checkout);

ReceiptPrinter.print(booking, roomNumber);

} else {

System.out.println("Quarto está vazio");

}

KeyboardReader.readKey();

}

private Booking getBoookingData() throws IOException, ParseException {

System.out.println("= Reserva de quarto =");

String renterName = KeyboardReader.readLineAsString("Nome do locatario: ");

String checkin = KeyboardReader.readLineAsString("Data de entrada : ", "dd/MM/yyyy hh:mm");

int guests = KeyboardReader.readLineAsInteger("Camas extra : ");

System.out.println();

return new Booking(renterName, checkin, guests);

}

}

### 5.2.2 RoomType.java

package ead2.poo.atps.marau.rooms;

public enum RoomType {

STANDARD {

@Override

public String toString() {

return "Padrão ";

}

},

COTTAGE {

@Override

public String toString() {

return "Chalé ";

}

},

BATH {

@Override

public String toString() {

return "Com banheira";

}

}

}

### 5.2.3 RoomList.java

package ead2.poo.atps.marau.rooms;

import java.io.IOException;

import java.text.ParseException;

import java.util.ArrayList;

import java.util.List;

import ead2.poo.atps.marau.Booking;

import ead2.poo.atps.marau.util.KeyboardReader;

public class RoomList {

private final List<Room> rooms;

public RoomList() {

rooms = new ArrayList<Room>();

}

public void setSize(final int standardRooms) {

for(int room = 0; room < standardRooms; room++) {

rooms.add(StandardRoom.EMPTY);

}

}

public void setSize(int standardRooms, int cottages) {

setSize(standardRooms);

for(int room = 0; room < cottages; room++) {

rooms.add(CottageRoom.EMPTY);

}

}

public void setSize(int standardRooms, int fancyRooms, int cottages) {

for(int room = 0; room < fancyRooms; room++) {

rooms.add(FancyRoom.EMPTY);

}

setSize(standardRooms, cottages);

}

public int addBooking(Booking booking) throws IOException {

int roomNumber;

do {

roomNumber = KeyboardReader.readLineAsInteger("Número do quarto : ");

if(!isEmpty(roomNumber)) {

System.out.println("O quarto está ocupado");

}

}while(!isEmpty(roomNumber));

Room room = null;

switch (rooms.get(roomNumber - 1).getRoomType()) {

case STANDARD:

room = new StandardRoom(booking);

break;

case COTTAGE:

room = new CottageRoom(booking);

break;

case BATH:

room = new FancyRoom(booking);

break;

}

rooms.set(roomNumber - 1, room);

return roomNumber;

}

public String toString() {

StringBuffer buffer = new StringBuffer();

for(int room = 0; room < count(); room++) {

buffer.append("[")

.append(room < 9 ? "0" : "")

.append(room + 1)

.append(": ")

.append(rooms.get(room))

.append("] ")

.append(room > 0 && (room + 1) % 5 == 0 ? "\n" : "");

}

return buffer.toString();

}

public int count() {

return rooms.size();

}

public boolean isEmpty(final int roomNumber) {

return (roomNumber - 1 > 0 && roomNumber - 1 < count() ? rooms.get(roomNumber - 1).isEmpty() : true);

}

public Booking checkout(int roomNumber, String checkout) throws ParseException {

Booking booking = rooms.get(roomNumber - 1).checkout(checkout);

rooms.set(roomNumber - 1, StandardRoom.EMPTY);

return booking;

}

}

### 5.2.4 FancyRoom.java

package ead2.poo.atps.marau.rooms;

import ead2.poo.atps.marau.Booking;

public class FancyRoom extends Room {

public static final Room EMPTY = new FancyRoom(null);

protected FancyRoom(Booking booking) {

super(booking);

}

@Override

public RoomType getRoomType() {

return RoomType.BATH;

}

@Override

public double luxuryRate() {

return 1.18;

}

}