

**Design patterns:**

Façade – This design pattern was used in order to separate the main gui from the underlying complexities of the rest of the program.

Factory – This design pattern was used in order to generate objects of type Customer, Booking and Guest. This refers to the classes CustomerFactory, BookingFactory and GuestFactory which create the respective objects.

**Class listings**

**MainWindow**

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace assessment2

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// Author: Sean Faughey

/// GUI logic and verification of input

/// Date last modified: 06/12/2016

/// </summary>

public partial class MainWindow : Window

{

Facade facade = new Facade(); //this program uses the facade design pattern, create a new facade

//constructor

public MainWindow()

{

InitializeComponent();

tbPassportNumber.MaxLength = 10; //forces the user to type no more than 10 characters for the passport

}

private void btnAddCustomer\_Click(object sender, RoutedEventArgs e) //method to add a customer

{

if (tbCustomerName.Text != "" && tbCustomerAddress.Text != "") { //Check that there is input in all necessary fields

facade.createCustomer(tbCustomerName.Text, tbCustomerAddress.Text); //access the createCustomer method in facade

}

else

{

MessageBox.Show("You must input all necessary customer fields!"); //message the user

}

}

private void btnAddBooking\_Click(object sender, RoutedEventArgs e) //method to add a booking

{

try

{

DateTime arrivalDate = DateTime.ParseExact(tbArrivalDate.Text, "dd/MM/yyyy", null); //change arrival date from string to time

DateTime departureDate = DateTime.ParseExact(tbDepartureDate.Text, "dd/MM/yyyy", null); //change departure date from string to time

bool carHire = (bool)cbCarHire.IsChecked; //checks if customer wants a car hire

bool eveningMeals = (bool)cbEveningMeals.IsChecked; //checks if customer wants evening meals

bool breakfast = (bool)cbBreakfast.IsChecked; //checks if customer wants breakfast

DateTime expectedDate;

if ((bool)cbCarHire.IsChecked) //if the car hire box is checked

{

if (!DateTime.TryParseExact(tbFrom.Text, "dd/MM/yyyy", new CultureInfo("en-US"), //if either form or until box is unfilled

DateTimeStyles.None, out expectedDate) || !DateTime.TryParseExact(tbUntil.Text, "dd/MM/yyyy", new CultureInfo("en-US"),

DateTimeStyles.None, out expectedDate))

{

MessageBox.Show("You must enter from and until dates for the car hire."); //tell the user to fill them

return; //exit method

}

else

{

if (tbDriver.Text != "") { //if the user hasn given a driver name

DateTime until = DateTime.ParseExact(tbUntil.Text, "dd/MM/yyyy", null);

DateTime from = DateTime.ParseExact(tbFrom.Text, "dd/MM/yyyy", null);

facade.createBooking(arrivalDate, departureDate, //access the createBooking() method in facade

eveningMeals, breakfast, carHire, Int32.Parse(tbCustomerReferenceNumber.Text), tbDietaryRequirements.Text, tbDietaryRequirements\_Breakfast.Text, from, until, tbDriver.Text);

}

else //if the user hasnt entered a driver name

{

MessageBox.Show("Please enter a driver name"); //output error message

}

}

}

if (!(bool)cbCarHire.IsChecked) //if the customer doesnt want a car

{

facade.createBooking(arrivalDate, departureDate, //access the createBooking() method in facade

eveningMeals, breakfast, carHire, Int32.Parse(tbCustomerReferenceNumber.Text), tbDietaryRequirements.Text, tbDietaryRequirements\_Breakfast.Text, DateTime.MinValue, DateTime.MinValue, tbDriver.Text);

}

}

catch (FormatException exc) { //catches exception when user has incorrectly entered information and outputs error message

MessageBox.Show(exc.Message);

MessageBox.Show("Please make sure that you have entered all dates in the correct format");

}

}

private void btnAddGuest\_Click(object sender, RoutedEventArgs e) //method to add a guest

{

int bookingResult;

Int32.TryParse(tbBookingReferenceNumber.Text, out bookingResult); //parse the booking reference number, 0 if not valid

if (bookingResult > 0)

{ //if the booking reference number is valid

if (facade.readBooking(bookingResult) != null) //and the booking exists

{

int age;

if (Int32.TryParse(tbAge.Text, out age)) { //check that the age has been input correctly

if (0 <= age && age <= 101) {

Booking booking = facade.readBooking(bookingResult); //retrieve the current booking

if (booking.GuestArray.Count < 4) { //check there is space on the booking for more guests

facade.createGuest(tbGuestName.Text, tbPassportNumber.Text, Int32.Parse(tbAge.Text)); //access the createGuest method in facade, adding the guest to the database

facade.amendBooking(booking.ArrivalDate, booking.DepartureDate, booking.BookingReferenceNumber, booking.EveningMeals, booking.Breakfast, booking.CarHire, booking.EveningDietaryRequirements,

booking.BreakfastDietaryRequirements, booking.CarHireStart, booking.CarHireEnd, booking.CustomerReferenceNumber, tbPassportNumber.Text, booking.Driver); //add the guest to the booking

}

else

{

MessageBox.Show("There are already 4 guests on this holiday."); //if there is no space on the booking

}

}

else

{

MessageBox.Show("Please insert an age between 0 and 101"); //if the age is out of bounds

}

}

else

{

MessageBox.Show("Please insert an age between 0 and 101"); //if the age is incorrectly input

}

}

else

{

MessageBox.Show("This is not a valid booking reference number!"); //if the bookingReferenceNumber doesnt exist

}

}

else

{

MessageBox.Show("Please input a booking reference number!"); //if the bookingReferenceNumber is incorrectly input

}

}

//finds the customer given the customer reference number

private void btnReadCustomer\_Click(object sender, RoutedEventArgs e)

{

int customerResult; //initialize customer result

Int32.TryParse(tbCustomerReferenceNumber.Text, out customerResult); //parse customer reference number if "" then = 0

if (customerResult > 0) //if customer reference number > 0

{

if (facade.readCustomer(customerResult) != null) { //read the customer using the customer reference number

tbCustomerName.Text = facade.readCustomer(Int32.Parse(tbCustomerReferenceNumber.Text)).Name; //access the facade.readCustomer method and update the GUI

tbCustomerAddress.Text = facade.readCustomer(Int32.Parse(tbCustomerReferenceNumber.Text)).Address; //^^

}

else

{

MessageBox.Show("This is not a valid customer reference number."); //if the customerReferenceNumber doesnt exist

}

}

else

{

MessageBox.Show("This is not a valid customer reference number."); //if the customerReferenceNumber doesnt is out of bounds or not an int

}

}

//finds the booking given the booking reference number

private void btnReadBooking\_Click(object sender, RoutedEventArgs e)

{

int bookingResult;

Int32.TryParse(tbBookingReferenceNumber.Text, out bookingResult); //parse the booking reference number, 0 if not valid

if (bookingResult > 0) { //if the booking reference number is valid

if (facade.readBooking(bookingResult) != null) //and the booking exists

{

Booking booking = facade.readBooking(bookingResult);

tbArrivalDate.Text = booking.ArrivalDate.Date.ToShortDateString(); //update arrival date

tbDepartureDate.Text = booking.DepartureDate.Date.ToShortDateString(); //update departure date

cbEveningMeals.IsChecked = booking.EveningMeals; //update evening meals

cbBreakfast.IsChecked = booking.Breakfast; //update breakfast

cbCarHire.IsChecked = booking.CarHire; //update car hire

tbDietaryRequirements.Text = booking.EveningDietaryRequirements; //update Evening Meals dietary requirements

tbDietaryRequirements\_Breakfast.Text = booking.BreakfastDietaryRequirements; //update Breakfast dietary requirements

tbDriver.Text = booking.Driver;

tbCustomerReferenceNumber.Text = booking.CustomerReferenceNumber.ToString();

if (!booking.CarHire) { //if the customer doesnt want a car clear the car related fields

tbFrom.Text = ""; //set the fields to blank

tbUntil.Text = "";

tbDriver.Text = "";

}

else

{

tbFrom.Text = booking.CarHireStart.ToShortDateString(); //update car hire start

tbUntil.Text = booking.CarHireEnd.Date.ToShortDateString(); //update car hire end

}

}

else

{

MessageBox.Show("This is not a valid booking reference number!"); //inform the user this is not a valid booking

}

}

}

//finds the guest given the passport number

private void btnReadGuest\_Click(object sender, RoutedEventArgs e)

{

if (facade.readGuest(tbPassportNumber.Text) != null) //and the guest exists

{

tbGuestName.Text = facade.readGuest(tbPassportNumber.Text).Name; //update the guest name

tbAge.Text = facade.readGuest(tbPassportNumber.Text).Age.ToString(); //update the guest age

}

else

{

MessageBox.Show("This is not a valid passport number!");

}

}

//deletes the customer given the customerReferenceNumber

private void btnDeleteCustomer\_Click(object sender, RoutedEventArgs e)

{

bool objectDeleted = facade.deleteObject(Int32.Parse(tbCustomerReferenceNumber.Text), "customer", "0"); //access deleteObject telling it we are deleting a customer

if (!objectDeleted) { //if the customer was not deleted

MessageBox.Show("The customer was not deleted"); //message to the user

}

}

//deletes a booking

private void btnDeleteBooking\_Click(object sender, RoutedEventArgs e)

{

bool objectDeleted = facade.deleteObject(Int32.Parse(tbBookingReferenceNumber.Text), "booking", "0"); //access deleteObject telling it we are deleting a booking

if (!objectDeleted) { //if the booking was not deleted

MessageBox.Show("The booking was not deleted!"); //message to the user

}

}

//deletes a guest

private void btnDeleteGuest\_Click(object sender, RoutedEventArgs e)

{

int bookingResult;

Int32.TryParse(tbBookingReferenceNumber.Text, out bookingResult); //parse the booking reference number, 0 if not valid

if (bookingResult > 0)

{ //if the booking reference number is valid

Booking booking = facade.readBooking(bookingResult); //read the booking with the valid booking reference number

if (booking != null) //and the booking exists

{

facade.deleteGuest(tbPassportNumber.Text, booking); //delete the guest

}

else

{

MessageBox.Show("The guest was not deleted"); //if the guest was not deleted, message the user

}

}

}

//amends the customer

private void btnAmendCustomer\_Click(object sender, RoutedEventArgs e)

{

try

{

if (tbCustomerAddress.Text != "" && tbCustomerName.Text != "") //if all necessary fields are input, except the customerReferenceNumber

{

bool amendedCustomer = facade.amendCustomer(Int32.Parse(tbCustomerReferenceNumber.Text), "customer", tbCustomerName.Text, tbCustomerAddress.Text); //access amend customer method

if (!amendedCustomer) //if the customer was not amended

{

MessageBox.Show("The customer was not amended!"); //message to the user

}

}

else //if the necessary fields are blank

{

MessageBox.Show("Please make sure all data was entered correctly"); //message the user

}

}

catch (FormatException exc) { //if the customerReferenceNumber is blank

MessageBox.Show("You must enter all necessary fields"); //message the user

}

}

//amends the booking

private void btnAmendBooking\_Click(object sender, RoutedEventArgs e)

{

bool amendedBooking = false;

int result;

if (Int32.TryParse(tbBookingReferenceNumber.Text, out result)) //if the booking reference number is a number

{

Booking booking = facade.readBooking(result); //read the booking

DateTime arrival;

DateTime departure;

DateTime from;

DateTime until;

if (DateTime.TryParseExact(tbArrivalDate.Text, "dd/MM/yyyy", new CultureInfo("en-US"), //if the arrival date box is filled

DateTimeStyles.None, out arrival))

{

if (DateTime.TryParseExact(tbDepartureDate.Text, "dd/MM/yyyy", new CultureInfo("en-US"), //and if the departure box is filled

DateTimeStyles.None, out departure))

{

DateTime.TryParseExact(tbUntil.Text, "dd/MM/yyyy", new CultureInfo("en-US"), DateTimeStyles.None, out until); //check if the car hire end date is filled in correctly

DateTime.TryParseExact(tbFrom.Text, "dd/MM/yyyy", new CultureInfo("en-US"), DateTimeStyles.None, out from); //check if the car hire start date is filled in correctly

if ((bool)cbCarHire.IsChecked) //if the customer wants a car hire

{

if (from > DateTime.MinValue && until > DateTime.MinValue && tbDriver.Text != "") //if the car hire start and end dates were valid

{

amendedBooking = facade.amendBooking(arrival, departure, booking.BookingReferenceNumber, (bool)cbEveningMeals.IsChecked, (bool)cbBreakfast.IsChecked, (bool)cbCarHire.IsChecked,

tbDietaryRequirements.Text, tbDietaryRequirements\_Breakfast.Text, from, until, booking.CustomerReferenceNumber, "", tbDriver.Text); //amend the booking

}

}

else //if the cutsomer doesnt want a car

{

amendedBooking = facade.amendBooking(arrival, departure, booking.BookingReferenceNumber, (bool)cbEveningMeals.IsChecked, (bool)cbBreakfast.IsChecked, (bool)cbCarHire.IsChecked,

tbDietaryRequirements.Text, tbDietaryRequirements\_Breakfast.Text, DateTime.MinValue, DateTime.MinValue, booking.CustomerReferenceNumber, "", ""); //amend the booking with minimum values and blank driver

}

}

}

}

if (!amendedBooking) { //if the booking was not amended

MessageBox.Show("The booking was not amended!"); //message the user

}

}

//amends the guest

private void btnAmendGuest\_Click(object sender, RoutedEventArgs e)

{

try //check the information was input correctly

{

bool amendedGuest = facade.amendGuest(tbGuestName.Text, Int32.Parse(tbAge.Text), tbPassportNumber.Text);

if (!amendedGuest) //if the guest was not amended

{

MessageBox.Show("The guest was not amended."); //message the user

}

}

catch (FormatException exc)

{ //if the information was not input correctly

MessageBox.Show("Make sure all data was correctly entered"); //message the user

}

}

//produces an invoice given the booking reference number

private void btnInvoice\_Click(object sender, RoutedEventArgs e)

{

int bookingReferenceNumber = 0;

Int32.TryParse(tbBookingReferenceNumber.Text, out bookingReferenceNumber); //check the bookingReferenceNumber was input correctly, if not bookingRefernceNumber = 0

facade.createInvoice(bookingReferenceNumber); //access the createInvoice method in the facade

}

}

}

**Façade**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

//Author: Sean Faughey

//Description: This class follows the facade design pattern in order to seperate the main functionality of the solution from the GUI

//Date last modified: 06/12/2016

//Design pattern: Facade

namespace assessment2

{

public class Facade

{

CustomerFactory customerFactory = new CustomerFactory(); //produces a customer factory

BookingFactory bookingFactory = new BookingFactory(); //produces a booking factory

GuestFactory guestFactory = new GuestFactory(); //produces a guest factory

SerializeData serializer = new SerializeData("testBinaryFile.txt"); //produces a serializer in order to store data

public void createCustomer(string name, string address) //method to create a customer

{

int customerReferenceNumber = serializer.findFirstAvailableNumber("customer"); //finds the first available number for a customer reference number

serializer.serializeObject(customerFactory.createCustomer(name, address, customerReferenceNumber)); //access the serializer and store the customer using the customer factory

}

//method to create a booking

public void createBooking(DateTime arrivalDate, DateTime departureDate, bool eveningMeals, bool breakfast, bool carHire, int customerReferenceNumber, string dietEvening, string dietBreakfast, DateTime from, DateTime until, string driver)

{

serializer.serializeObject(bookingFactory.createBooking( //access the serializer and store the booking using the booking factory

arrivalDate, departureDate, serializer.findFirstAvailableNumber("booking"), eveningMeals, breakfast, carHire, customerReferenceNumber, dietEvening, dietBreakfast, from, until, driver));

}

//method to create a guest

public void createGuest(string guestName, string passportNumber, int age)

{

serializer.serializeObject(guestFactory.createGuest(guestName, passportNumber, age)); //access the serializer and store the guest using the guest factory

}

//method to retreive a customer

public Customer readCustomer(int customerReferenceNumber)

{

Customer customer = null;

if (customerReferenceNumber != 0) { //if given a customer reference number

customer = (Customer) serializer.deserializeObject(customerReferenceNumber, "customer", "0"); //find the customer usnig the serializer

}

return customer;

}

//method to retreive a booking

public Booking readBooking(int bookingReferenceNumber)

{

Booking booking = null;

if (bookingReferenceNumber != 0) //if given a booking reference number

{

booking = (Booking)serializer.deserializeObject(bookingReferenceNumber, "booking", "0"); //find the booking using the serializer

}

return booking;

}

//method to retreieve a guest

public Guest readGuest(string passportNumber)

{

Guest guest = null;

if (passportNumber != "") { //if given a passport number

guest = (Guest)serializer.deserializeObject(0, "guest", passportNumber); //find the guest using the passport number

}

return guest;

}

//method to delete an object from the file

public bool deleteObject(int objectIdentifier, string objectType, string passportNumber)

{

return serializer.deleteObject(objectIdentifier, objectType, passportNumber); //delete an object from file using the serializer

}

//method to delete a guest from the relevant booking

public void deleteGuest(string passportNumber, Booking booking)

{

booking.GuestArray.Remove(passportNumber); //remove the guest from the booking given

serializer.amendBookingRemovePassport(booking); //use the serializer to amend the booking which the guest is attributed to

serializer.deleteObject(0, "guest", passportNumber); //use the serializer to remove the guest from file

}

//method to amend a customer's information

public bool amendCustomer(int objectIdentifier, string objectType, string name, string address)

{

return serializer.amendCustomer(objectIdentifier, name, address); //use the serializer to amend the information

}

//method to amend a booking's information

public bool amendBooking(DateTime arrivalDate, DateTime departureDate, int bookingReferenceNumber, bool eveningMeals, bool breakfast, bool carHire, string eveningDiet, string breakfastDiet, DateTime from, DateTime until, int customerReferenceNumber, string passportNumber, string driver)

{

//use the serializer to amend the booking information

return serializer.amendBooking(arrivalDate, departureDate, bookingReferenceNumber, eveningMeals, breakfast, carHire, eveningDiet, breakfastDiet, from, until, customerReferenceNumber, passportNumber, driver);

}

//methoid to amend a guest's information

public bool amendGuest(string name, int age, string passportNumber)

{

//use the serializer to amend the guest information

return serializer.amendGuest(name, age, passportNumber);

}

//method to create an invoice of a booking

public void createInvoice(int bookingReferenceNumber) {

//create a new invoice object by reading the booking, passing it a serializer to read from

Invoice invoice = new Invoice(readBooking(bookingReferenceNumber), serializer);

invoice.Show(); //show a new Invoice window

}

}

}

**Booking**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Author name: Sean Faughey

//Description: this is a booking object meant ot represent a booking in the napire holiday village.

// Holds information regarding the booking including who made the booking (customer) and who is in the booking (guests)

//This class is hidden from the GUI using the facade design pattern (Facade class), and is created using the BookingFactory class.

//Last modified: 06/12/2016

namespace assessment2

{

[Serializable] //allows booking objects to be added to a file using a serializer

public class Booking

{

private DateTime arrivalDate; //represents the date of arrival of the guests

private int bookingReferenceNumber; //represents the booking reference number, a unique identifier of bookings

private int customerReferenceNumber; //represents the customer reference number of the customer who made the booking

private DateTime departureDate; //represents the date of departure of the guests

private bool eveningMeals; //whether the customer chose to include evening meals

private bool breakfast; //whether the customer chose to include breakfast

private bool carHire; //whether the customer chose to include car hire

private string eveningDietaryRequirements; //any dietary requirements of the evening meals

private string breakfastDietaryRequirements; //any breakfast dietary requirements

private DateTime carHireStart; //date of start of the car hire

private DateTime carHireEnd; //date of the car hire end

private string driver; //name of the driver of the car

private List<string> guestArray; //list of guests on the booking

//constructor

public Booking(DateTime arrivalDate, DateTime departureDate, int bookingReferenceNumber,

bool eveningMeals, bool breakfast, bool carHire, int customerReferenceNumber, string eveningDietaryRequirements,

string breakfastDietaryRequirements, DateTime carHireStart, DateTime carHireEnd, string driver)

{

guestArray = new List<string>();

this.arrivalDate = arrivalDate;

this.bookingReferenceNumber = bookingReferenceNumber;

this.departureDate = departureDate;

this.eveningMeals = eveningMeals;

this.breakfast = breakfast;

this.carHire = carHire;

this.eveningDietaryRequirements = eveningDietaryRequirements;

this.breakfastDietaryRequirements = breakfastDietaryRequirements;

this.carHireStart = carHireStart;

this.carHireEnd = carHireEnd;

this.customerReferenceNumber = customerReferenceNumber;

this.driver = driver;

}

//getters and setters for all of the properties

public int BookingReferenceNumber

{

get

{

return this.bookingReferenceNumber;

}

set

{

if (value == null)

{

throw new Exception("This is wrong");

}

this.bookingReferenceNumber = value;

}

}

public DateTime ArrivalDate

{

get

{

return this.arrivalDate;

}

set

{

if (value == null)

{

throw new Exception("This is wrong!");

}

this.arrivalDate = value;

}

}

public DateTime DepartureDate

{

get

{

return this.departureDate;

}

set

{

if (value == null)

{

throw new Exception("This is wrong");

}

this.departureDate = value;

}

}

public bool Breakfast

{

get

{

return this.breakfast;

}

set

{

if (value == null)

{

throw new Exception("this is wrong");

}

this.breakfast = value;

}

}

public bool CarHire

{

get

{

return this.carHire;

}

set

{

if (value == null)

{

throw new Exception("wrnog");

}

this.carHire = value;

}

}

public bool EveningMeals

{

get

{

return this.eveningMeals;

}

set

{

if (value == null)

{

throw new Exception("wrong");

}

this.eveningMeals = value;

}

}

public int CustomerReferenceNumber

{

get

{

return this.customerReferenceNumber;

}

set

{

if (value == null)

{

throw new Exception("cant happen pal");

}

this.customerReferenceNumber = value;

}

}

public List<string> GuestArray

{

get

{

return this.guestArray;

}

}

public string EveningDietaryRequirements

{

get

{

return this.eveningDietaryRequirements;

}

set

{

this.eveningDietaryRequirements = value;

}

}

public string BreakfastDietaryRequirements

{

get

{

return this.breakfastDietaryRequirements;

}

set

{

this.breakfastDietaryRequirements = value;

}

}

public DateTime CarHireStart

{

get

{

return this.carHireStart;

}

set

{

this.carHireStart = value;

}

}

public DateTime CarHireEnd

{

get

{

return this.carHireEnd;

}

set

{

this.carHireEnd = value;

}

}

public string Driver

{

get

{

return this.driver;

}

set

{

this.driver = value;

}

}

}

}

**BookingFactory**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Author name: Sean Faughey

//Description: This is a factory which follows the factory design pattern in order to create bookings with unique identifiers

//This class is hidden from the GUI using the facade design pattern (Facade class)

//Last modified: 06/12/2016

namespace assessment2

{

class BookingFactory

{

//creates a booking object and returns it

public Booking createBooking(DateTime arrivalDate, DateTime departureDate, int bookingReferenceNumber, bool eveningMeals, bool breakfast, bool carHire, int customerReferenceNumber, string eveningDietaryRequirements,

string breakfastDietaryRequirements, DateTime carHireStart, DateTime carHireEnd, string driver)

{

Booking booking = new Booking(arrivalDate, departureDate, bookingReferenceNumber,

eveningMeals, breakfast, carHire, customerReferenceNumber, eveningDietaryRequirements, breakfastDietaryRequirements, carHireStart, carHireEnd, driver);

return booking;

}

}

}

**Customer**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Author name: Sean Faughey

//Description: This is a customer object meant to represent a customer in the Napier Holiday Village

// Holds information regarding the customer

//This class is hidden from the GUI using the facade design pattern (Facade class), and is created using the CustomerFactory class.

//Last modified: 06/12/2016

namespace assessment2

{

[Serializable] //allows the customer to be added to a file using the serializer

public class Customer

{

private string address; //represents the customer's address

private int customerReferenceNumber; //unique identifier of a customer

private string name; //represents the customer's name

//constructor

public Customer(string name, string address, int customerReferenceNumber)

{

this.address = address;

this.customerReferenceNumber = customerReferenceNumber;

this.name = name;

}

//gets and setters of the private properties

public int CustomerReferenceNumber {

get

{

return this.customerReferenceNumber;

}

set

{

if (value == null) {

throw new Exception("This is wrong");

}

this.customerReferenceNumber = value;

}

}

public string Address

{

get

{

return this.address;

}

set

{

if (value == null) {

throw new Exception("This is wrong.");

}

this.address = value;

}

}

public string Name

{

get

{

return this.name;

}

set

{

if (value == null) {

throw new Exception("This is wrong.");

}

this.name = value;

}

}

}

}

**CustomerFactory**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Author name: Sean Faughey

//Description: This is a factory which follows the factory design pattern in order to create Customers with unique identifiers

//This class is hidden from the GUI using the facade design pattern (Facade class)

//Last modified: 03/12/2016

namespace assessment2

{

class CustomerFactory

{

//method to create a customer and return it

public Customer createCustomer(string name, string address, int customerReferenceNumber)

{

Customer customer = new Customer(name, address, customerReferenceNumber);

return customer;

}

}

}

**Guest**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Author name: Sean Faughey

//Description: This is a Guest object meant to represent a guest in the Napier holiday Village

// Holds information regarding the Guest

//This class is hidden from the GUI using the facade design pattern (Facade class), and is created using the GuestFactory class.

//Last modified: 03/12/2016

namespace assessment2

{

[Serializable] //allows the guest to be written to file using the serializer

public class Guest

{

private int age; //integer to represent the guest's age

private string name; //represents the guest's name

private string passportNumber; //unique identifier of the guest representing their passport number

//constructor

public Guest(string name, string passportNumber, int age)

{

this.age = age;

this.name = name;

this.passportNumber = passportNumber;

}

//getters and setters of private properties

public string PassportNumber

{

get

{

return this.passportNumber;

}

set

{

if (value == null)

{

throw new Exception("This is wrong");

}

this.passportNumber = value;

}

}

public string Name

{

get

{

return this.name;

}

set

{

if (value == null) {

throw new Exception("This is wrong");

}

this.name = value;

}

}

public int Age

{

get

{

return this.age;

}

set

{

if (value == null) {

throw new Exception("This is wrong");

}

this.age = value;

}

}

}

}

**GuestFactory**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Author name: Sean Faughey

//Description: This is a factory which follows the factory design pattern in order to create guests with unique identifiers

//This class is hidden from the GUI using the facade design pattern (Facade class)

//Last modified: 06/12/2016

namespace assessment2

{

class GuestFactory

{

public Guest createGuest(string guestName, string passportNumber, int age) //create a guests and return it

{

Guest guest = new Guest(guestName, passportNumber, age);

return guest;

}

}

}

**Invoice**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

//Author name: Sean Faughey

//Description: This class deals with the GUI for the Invoice window

//Last modified: 06/12/2016

namespace assessment2

{

/// <summary>

/// Interaction logic for Invoice.xaml

/// </summary>

public partial class Invoice : Window

{

private Booking booking; //a booking object to be displayed on screen

public Invoice(Booking booking, SerializeData serializer) //a constructor

{

InitializeComponent();

this.booking = booking;

InvoiceBooking invoice = new InvoiceBooking(booking, serializer); //create a new invoicebooking object which deals with calculations

tbBookingReferenceNumber.Text = booking.BookingReferenceNumber.ToString(); //show the booking reference number on screen

tbCostPerNight.Text = invoice.basicCost().ToString(); //show the cost per night

tbExtrasCost.Text = invoice.extrasCost().ToString(); //show the extras cost

}

}

}

**InvoiceBooking**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

//Author name: Sean Faughey

//Description: This class deals with calculations of a booking invoice

//Last modified: 06/12/2016

namespace assessment2

{

class InvoiceBooking

{

private Booking booking; //represents a booking to be represented

private SerializeData serializer; //a serializer in order to read from file

private List<string> listOfGuests; //a list of all the guests of the booking

public InvoiceBooking(Booking booking, SerializeData serializer) //constructor

{

this.booking = booking;

this.serializer = serializer;

listOfGuests = booking.GuestArray; //gets the guests from the booking

}

//a method to calculate the total cost of a booking

public int totalCost()

{

int totalCost = basicCost() + extrasCost(); //cost per night per guest + extras per night

return totalCost;

}

//a method to calculate the basic cost of a booking before extras

public int basicCost()

{

int basicCost = 0;

int totalDays = (booking.DepartureDate - booking.ArrivalDate).Days; //calculates the number of nights the booking si for

foreach (String passport in listOfGuests) { //for every guest

Guest guest = (Guest)serializer.deserializeObject(0, "guest", passport); //read the guest from file

if (guest.Age < 18) { //if the guest is under 18 add 30 to the basic cost

basicCost = basicCost + 30;

}

if (guest.Age >= 18) { //if the guest is 18 or over add 50 to the basic cost

basicCost = basicCost + 50;

}

}

return basicCost; //return the basic cost

}

//method to calculate the cos tof all the extras per night

public int extrasCost()

{

int extrasCost = 0;

if (booking.CarHire) { //if the booking has a car hire

int carHireDays = (booking.CarHireEnd - booking.CarHireStart).Days; //calculate for how long

extrasCost = carHireDays \* 50; //multiply nights by price (50)

}

if (booking.Breakfast) { //if the customer chose to have breakfast included

int breakfastCost = listOfGuests.Count \* 5; //count the number of guests and multiply by the cost (5)

extrasCost = extrasCost + breakfastCost; //add the breakfast cost to the total extras cost

}

if (booking.EveningMeals) { //if the customer chose to have evening meals included

int eveningMealsCost = listOfGuests.Count \* 15; //count the number of guests and multiply by the cost (15)

extrasCost = extrasCost + eveningMealsCost; //add the evening meals cost to the total extras cost

}

return extrasCost; //return the overall cost of extras

}

}

}

**SerializeData**

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace assessment2

{

public class SerializeData

{

FileStream stream = null; //open a new FileStream

BinaryFormatter bformatter = null; //open a new BinaryFormatter

String txtFileName = ""; //create a blank filename to be input

ArrayList objectArray = new ArrayList(); //create a blank list which will contain all objects

public SerializeData(String fileName)

{

txtFileName = fileName; //supply the file name

bformatter = new BinaryFormatter();

}

//method to check whether a file exists and if not create one

public void createFile()

{

if (!File.Exists(txtFileName)) { //check if the file exists, if not

stream = File.Open(txtFileName, FileMode.Create); //create the file

closeStream();

}

}

//this method writes an object to the file

public void serializeObject(Object objectToSerialize)

{

createFile(); //check if the file exists

objectArray = deserializeArray(); //read in all of the objects to an array

objectArray.Add(objectToSerialize); //add this object to the array

using (var stream = File.OpenWrite(txtFileName))

{

bformatter.Serialize(stream, objectArray); //write the new array to the file

}

}

//method to find the first available number for unique auto-incrementing identifiers

public int findFirstAvailableNumber(string objectToCount)

{

int counter = 0;

deserializeArray(); //read in all objects

for (int i = 0; i < objectArray.Count; i++ ) //for every object

{

if (objectToCount == "customer")

{ //if the object we're looking to find the first available number for is a customer

if (objectArray[i] is Customer) //and the object in the array is a customer

{

counter++; //count of customers in the array

}

}

if (objectToCount == "booking") //if the object we're looking to find the first available number for is a booking

{ //and the object in the array is a booking

if (objectArray[i] is Booking)

{

counter++; //count of bookings in the array

}

}

if (objectToCount == "guest") //if the object we're looking to find the first available number for is a guest

{ //and the object in the arra is a guest

if (objectArray[i] is Guest)

{

counter++; //count of guests in the array

}

}

}

return counter+1; //return the next number up from the number of objects in the array

}

//method to read in the entire file

public ArrayList deserializeArray()

{

createFile(); //check if the file exists

var list = new ArrayList();

using (var stream = File.OpenRead(txtFileName))

{

if (stream.Length != 0) { //if the file is not blank

list = (ArrayList)bformatter.Deserialize(stream); //read the file and make the list = the contents

}

}

objectArray = list;

return list; //return a list representing the list stored in the file

}

//method to read a specific object from file

public Object deserializeObject(int objectIdentifier, string objectType, string passportNumber)

{

createFile(); //check to see if the file exists

Object objectToReturn = null;

objectArray = deserializeArray(); //read in the entire file

try

{

for (int i = 0; i < objectArray.Count; i++) //for every object in the list

{

if (objectType == "customer") //if we're reading a customer

{

if (objectArray[i] is Customer) //and the object in the array is a customer

{

Customer customer = (Customer)objectArray[i]; //read the customer

if (customer.CustomerReferenceNumber == objectIdentifier) //if its the customer we're looking for

{

objectToReturn = customer; //return the customer

break; //exit loop

}

}

}

if (objectType == "booking") //if we're read a booking

{

if (objectArray[i] is Booking) //and the object in the array is a booking

{

Booking booking = (Booking)objectArray[i]; //read the booking

if (booking.BookingReferenceNumber == objectIdentifier) //if its the booking we're looking for

{

objectToReturn = booking; //return the booking

break; //exit the loop

}

}

}

if (objectType == "guest") //if we're reading a guest

{

if (objectArray[i] is Guest) //and the object in the array is a guest

{

Guest guest = (Guest)objectArray[i]; //read the guest

if (guest.PassportNumber == passportNumber) //if its the guest we're looking for

{

objectToReturn = guest; //return the guest

break; //exit loop

}

}

}

}

}

catch(SerializationException e)

{

Console.WriteLine(e);

}

return objectToReturn; //return the customer/booking/guest

}

//method to delete an object from file

public bool deleteObject(int objectIdentifier, string objectType, string passportNumber)

{

createFile(); //check to see if the file exists

deserializeArray(); //read in the entire file to an array

bool foundObject = false; //set boolean foundobject to be false

for (int i = 0; i < objectArray.Count; i++) //for every object in the array

{

if (objectType == "customer" && objectArray[i] is Customer) { //if we're looking for a customer and the object in the array is a customer

Customer objectToCheck = (Customer) objectArray[i]; //read the customer

if (objectToCheck.CustomerReferenceNumber == objectIdentifier) //if its the customer we're looking for

{

bool customerHasBookings = false; //set boolean to false

foreach (Object thing in objectArray) //for every object in the array

{

if (thing is Booking) //if it is a booking

{

Booking booking = (Booking)thing; //read the booking

if (booking.CustomerReferenceNumber == objectIdentifier) //if the booking is related to the customer we are looking to delete

{

MessageBox.Show("This customer has bookings, cannot delete."); //message the user that we cannot delete the customer

customerHasBookings = true; //set customer has bookings to be true

break; //exi the loop

}

}

}

if (!customerHasBookings) //if the customer doesnt have any bookings

{

objectArray.RemoveAt(i); //remove the customer from the array

foundObject = true; //set foundObject to true

break; //exit the loop

}

}

}

if (objectType == "booking" && objectArray[i] is Booking) //if we're looking to delete a booking and the object in the array is a booking

{

Booking objectToCheck = (Booking)objectArray[i]; //read the booking

if (objectToCheck.BookingReferenceNumber == objectIdentifier) //if its the specific booking we're looking for

{

objectArray.RemoveAt(i); //remove the booking

foundObject = true; //set found object to be true

break; //exit the loop

}

}

if (objectType == "guest" && objectArray[i] is Guest) //if we're looking to delete a guest and the object in the array is a guest

{

Guest objectToCheck = (Guest)objectArray[i]; //read the guest

if (objectToCheck.PassportNumber == passportNumber) //if its the specific guest we're looking for

{

objectArray.RemoveAt(i); //remove the guest form the array

foundObject = true; //set found object to be true

break; //exit the loop

}

}

}

stream = File.Open(txtFileName, FileMode.Create); //open stream to the file, ready to be overwritten

bformatter.Serialize(stream, objectArray); //overwrite the file

closeStream(); //close the stream to the file, we're finished writing

return foundObject; //return whether we deleted the file

}

//metohd to amend a specific customer

public bool amendCustomer(int objectIdentifier, string name, string address)

{

bool amendedCustomer = false; //boolean whether we have amended the customer

ArrayList objectArray = deserializeArray(); //read in the entire file

for (int i = 0; i < objectArray.Count; i++) //for every object in the file

{

if (objectArray[i] is Customer) //if the object is a customer

{

Customer objectToCheck = (Customer)objectArray[i]; //read in the customer

if (objectToCheck.CustomerReferenceNumber == objectIdentifier) //if its the customer we're looking for

{

objectToCheck.Name = name; //amend the name

objectToCheck.Address = address; //amend the address

using (var stream = File.OpenWrite(txtFileName))

{

bformatter.Serialize(stream, objectArray); //overwrite the new array to the file

amendedCustomer = true; //boolean = true, we have amended the customer

}

}

}

}

return amendedCustomer; //return whether we ahve amended the customer

}

//method to amend a specific booking

public bool amendBooking(DateTime arrivalDate, DateTime departureDate, int bookingReferenceNumber, bool eveningMeals, bool breakfast, bool carHire, string eveningDiet, string breakfastDiet, DateTime from, DateTime until, int customerReferenceNumber, string passportNumber, string driver)

{

bool amendedBooking = false; //boolean stating whethe rwe have amended the booking

ArrayList objectArray = deserializeArray(); //read in the entire file

for (int i = 0; i < objectArray.Count; i++) //for every object in the file

{

if (objectArray[i] is Booking) //if the object is a booking

{

Booking objectToCheck = (Booking)objectArray[i]; //read in the booking

if (objectToCheck.BookingReferenceNumber == bookingReferenceNumber) //if its the specific booking we were looking for

{

objectToCheck.ArrivalDate = arrivalDate; //ammend the booking's information

objectToCheck.DepartureDate = departureDate;

objectToCheck.EveningMeals = eveningMeals;

objectToCheck.Breakfast = breakfast;

objectToCheck.CarHire = carHire;

objectToCheck.EveningDietaryRequirements = eveningDiet;

objectToCheck.BreakfastDietaryRequirements = breakfastDiet;

objectToCheck.CarHireStart = from;

objectToCheck.CarHireEnd = until;

objectToCheck.Driver = driver;

if (customerReferenceNumber != 0) //if we have provided a customer reference number - we dont want to set this to 0!

{

objectToCheck.CustomerReferenceNumber = customerReferenceNumber; //amend the customer reference number

}

if (passportNumber != "") //if we have provided a guest which we want to add

{

if (objectToCheck.GuestArray.Count == 0) { //if there are no guests on this booking

objectToCheck.GuestArray.Add(passportNumber); //add the guest

}

else //if there are guests on this booking

{

bool foundIt = false; //boolean to see if this guest already exists

for (int count = 0; count < objectToCheck.GuestArray.Count; count++) //for every guest on the booking

{

if (objectToCheck.GuestArray[count] == passportNumber) //if the guest we're looking to add is on the booking

{

foundIt = true; //boolean = true

}

}

if (!foundIt) //if boolean = false - we never found the guest

{

objectToCheck.GuestArray.Add(passportNumber); //add the guest

}

}

}

using (var stream = File.OpenWrite(txtFileName))

{

bformatter.Serialize(stream, objectArray); //overwrite the array to the file

amendedBooking = true; //return will = true

}

}

}

}

return amendedBooking; //return whether the booking was amended

}

//method to amend the booking when deleting a guest

public void amendBookingRemovePassport(Booking booking)

{

ArrayList objectArray = deserializeArray(); //read in the entire file

for (int i = 0; i < objectArray.Count; i++) //for every object in the file

{

if (objectArray[i] is Booking) //if the object is a booking

{

Booking objectToCheck = (Booking) objectArray[i]; //read the booking

if (objectToCheck.BookingReferenceNumber == booking.BookingReferenceNumber ) //if its the booking we're looking for

{

objectArray[i] = booking; //this booking = the booking provided

using (var stream = File.OpenWrite(txtFileName))

{

bformatter.Serialize(stream, objectArray); //overwrite this booking to file

}

}

}

}

}

//method to amend a guest

public bool amendGuest(string name, int age, string objectIdentifier)

{

bool amendedGuest = false; //boolean whether we have amended the guest

ArrayList objectArray = deserializeArray(); //read in the entire file

for (int i = 0; i < objectArray.Count; i++) //for every object in the file

{

if (objectArray[i] is Guest) //if the object is a guest

{

Guest objectToCheck = (Guest)objectArray[i]; //read in the guest

if (objectToCheck.PassportNumber == objectIdentifier) //if the guest is the specific guest we are looking for

{

objectToCheck.Name = name; //amend the guest name

objectToCheck.Age = age; //amend the guest age

using (var stream = File.OpenWrite(txtFileName))

{

bformatter.Serialize(stream, objectArray); //overwrite the guest to the file

amendedGuest = true; //boolean stating whether we have amended the guest is true

}

}

}

}

return amendedGuest; //return the boolean

}

//method to close the file stream, allowing it to be accessible later

public void closeStream()

{

stream.Close(); //close the stream

}

}

}

**TestAssessment2**

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using assessment2;

using System.IO;

//Author name: Sean Faughey

//Description: This is a test class which tests all of the methods within the solution allowing you to add, change or delete a customer, booking or guest

//Last modified: 06/12/2016

namespace assessment2Test

{

[TestClass]

public class TestAssessment2

{

//this method test the functionality of the createCustomer method in the facade by adding a customer,

//reading in the customer using the customerReferenceNumber and comparing the properties

[TestMethod]

public void TestAddCustomer()

{

File.WriteAllText("testBinaryFile.txt", String.Empty); //make sure the file is empty

Facade facade = new Facade(); //create a new facade

Customer customer = new Customer("john", "address", 1); //expected customer

facade.createCustomer("john", "address"); //create a customer

Assert.AreEqual(customer.Name, facade.readCustomer(1).Name); //check that the properties of the expected customer and the written customer are the same

Assert.AreEqual(customer.Address, facade.readCustomer(1).Address);

Assert.AreEqual(customer.CustomerReferenceNumber, facade.readCustomer(1).CustomerReferenceNumber);

}

//this method tests the functionality of the createBooking method in the facade by adding a booking,

//reading in the booking using the bookingReferenceNumber and comparing the properties

[TestMethod]

public void TestAddBooking()

{

DateTime arrivalDate = DateTime.Parse("01/01/2001");

DateTime departureDate = DateTime.Parse("02/02/2001");

DateTime carHireStart = DateTime.Parse("01/01/2001");

DateTime carHireEnd = DateTime.Parse("02/02/2001");

File.WriteAllText("testBinaryFile.txt", String.Empty); //check the the file is empty when we start

Facade facade = new Facade(); //create a new facade

Customer customer = new Customer("john", "address", 1); //test customer

Booking booking = new Booking(arrivalDate, departureDate, 1, true, true, true, 1, "", "", carHireStart, carHireEnd, "john"); //expected booking

facade.createBooking(arrivalDate, departureDate, true, true, true, 1, "", "", carHireStart, carHireEnd, "john"); //create the booking

Assert.AreEqual(booking.ArrivalDate, facade.readBooking(1).ArrivalDate); //checks that all of the information is correct

Assert.AreEqual(booking.DepartureDate, facade.readBooking(1).DepartureDate);

Assert.AreEqual(booking.EveningMeals, facade.readBooking(1).EveningMeals);

Assert.AreEqual(booking.Breakfast, facade.readBooking(1).Breakfast);

Assert.AreEqual(booking.CarHire, facade.readBooking(1).CarHire);

Assert.AreEqual(booking.CustomerReferenceNumber, facade.readBooking(1).CustomerReferenceNumber);

Assert.AreEqual(booking.EveningDietaryRequirements, facade.readBooking(1).EveningDietaryRequirements);

Assert.AreEqual(booking.BreakfastDietaryRequirements, facade.readBooking(1).BreakfastDietaryRequirements);

Assert.AreEqual(booking.CarHireStart, facade.readBooking(1).CarHireStart);

Assert.AreEqual(booking.CarHireEnd, facade.readBooking(1).CarHireEnd);

Assert.AreEqual(booking.Driver, facade.readBooking(1).Driver);

}

//this method tests the functionality of the createGuest method in the facade by adding a guest,

//reading in the guest using the passportNumber and comparing the properties

[TestMethod]

public void TestAddGuest()

{

File.WriteAllText("testBinaryFile.txt", String.Empty); //make sure the file is empty when we start

Facade facade = new Facade(); //create a new facade

Guest guest = new Guest("john", "123abc", 21); //expected guest

facade.createGuest("john", "123abc", 21); //create the guest

Assert.AreEqual(guest.Name, facade.readGuest("123abc").Name); //checks that all the information is correct

Assert.AreEqual(guest.Age, facade.readGuest("123abc").Age);

Assert.AreEqual(guest.PassportNumber, facade.readGuest("123abc").PassportNumber);

}

//this method tests the deleteCustomer method in the facade by creating a customer, checking that the customer is not null using the readCustomer method

//then calling the deleteObject method with the customerReferenceNumber and customer object identifier

//and checking whether the readCustomer method then returns null

[TestMethod]

public void TestDeleteCustomer()

{

File.WriteAllText("testBinaryFile.txt", String.Empty); //check to see that the file is empty

Facade facade = new Facade(); //create a new facade

facade.createCustomer("john", "address"); //create the customer

Assert.IsNotNull(facade.readCustomer(1)); //check the customer was created

facade.deleteObject(1, "customer", ""); //delete the customer

Assert.IsNull(facade.readCustomer(1)); //ccheck the customer was deleted

}

//this method tests the deleteBooking method in the facade by creating a booking, checking that the booking is not null using the readBooking method

//then calling the deleteObject method with the bookingReferenceNumber and booking object identifier,

//and checking whether the readBooking method then returns null

[TestMethod]

public void TestDeleteBooking()

{

DateTime arrivalDate = DateTime.Parse("01/01/2001");

DateTime departureDate = DateTime.Parse("02/02/2001");

DateTime carHireStart = DateTime.Parse("01/01/2001");

DateTime carHireEnd = DateTime.Parse("02/02/2001");

File.WriteAllText("testBinaryFile.txt", String.Empty); //make sure we are working with an empty file

Facade facade = new Facade(); //create a facade

facade.createBooking(arrivalDate, departureDate, true, true, true, 1, "", "", carHireStart, carHireEnd, "john"); //create a booking

Assert.IsNotNull(facade.readBooking(1)); //check the booking exists

facade.deleteObject(1, "booking", ""); //delete the booking

Assert.IsNull(facade.readBooking(1)); //check the booking was deleted

}

//this method tests the deleteGuest method in the facade by creating a guest, checking that the guest is not null using the readGuest method

//and then calling the deleteGuest method with the passportNumber and object identifier,

//then the amendBooking method is called to remove the guest from the booking

//checks then occur to see whether that guest exists using the readGuest method and checking the number of guests the booking holds (should be 0)

[TestMethod]

public void TestDeleteGuest()

{

DateTime arrivalDate = DateTime.Parse("01/01/2001");

DateTime departureDate = DateTime.Parse("02/02/2001");

DateTime carHireStart = DateTime.Parse("01/01/2001");

DateTime carHireEnd = DateTime.Parse("02/02/2001");

File.WriteAllText("testBinaryFile.txt", String.Empty); //make sure we are working with an empty file

Facade facade = new Facade(); //create a new facade

facade.createBooking(arrivalDate, departureDate, true, true, true, 1, "", "", carHireStart, carHireEnd, "john"); //create a booking

Booking booking = facade.readBooking(1); //read in the booking

facade.createGuest("john", "123abc", 21); //create the guest

facade.amendBooking(arrivalDate, departureDate, 1, true, true, true, "", "", carHireStart, carHireEnd, 1, "123abc", "john"); //amend the booking with the guest

facade.deleteGuest("123abc", booking); //delete the guest

Assert.IsNull(facade.readGuest("123abc")); //check the guest does not exist on file

Assert.IsTrue(facade.readBooking(1).GuestArray.Count == 0); //check the reference to the guest in the booking has been deleted

}

//this method tests the amendCustomer method by creating a customer, and then amending the customer using the amendCustomer method

//the properties are then checked to not be equal to what they were prior to the amendCustomer method being called

[TestMethod]

public void TestAmendCustomer()

{

File.WriteAllText("testBinaryFile.txt", String.Empty); //make sure we are working with an empty file

Facade facade = new Facade(); //create a new facade

facade.createCustomer("john", "address"); //create a new customer

facade.amendCustomer(1, "customer", "notjohn", "notaddress"); //amend the customer

Assert.AreNotEqual("john", facade.readCustomer(1).Name); //check the customer's information has changed from the original information

Assert.AreNotEqual("address", facade.readCustomer(1).Address); //^^

}

//this method tests the amendBooking method by creating a booking, and then amending the booking using the amendBooking method

//the properties are then checked to not be equal to what they were prior to the amendBooking method being called

[TestMethod]

public void TestAmendBooking()

{

DateTime arrivalDate = DateTime.Parse("01/01/2001");

DateTime departureDate = DateTime.Parse("02/02/2001");

DateTime carHireStart = DateTime.Parse("01/01/2001");

DateTime carHireEnd = DateTime.Parse("02/02/2001");

DateTime arrivalDateAmended = DateTime.Parse("02/01/2001");

DateTime departureDateAmended = DateTime.Parse("03/01/2001");

DateTime carHireStartAmended = DateTime.MinValue;

DateTime carHireEndAmended = DateTime.MinValue;

File.WriteAllText("testBinaryFile.txt", String.Empty); //make sure we are working with a new file

Facade facade = new Facade(); //create a new facade

Customer customer = new Customer("john", "address", 1); //create a !expected customer

Booking booking = new Booking(arrivalDate, departureDate, 1, true, true, true, 1, "", "", carHireStart, carHireEnd, "john"); //create a !expected booking

facade.createBooking(arrivalDate, departureDate, true, true, true, 1, "", "", carHireStart, carHireEnd, "john"); //create a booking

facade.amendBooking(arrivalDateAmended, departureDateAmended, 1, false, false, false, "dietary requirement", "dietary requirement", carHireStartAmended, carHireEndAmended, 1, "", ""); //amend the booking

Assert.AreNotEqual(booking.ArrivalDate, facade.readBooking(1).ArrivalDate); //checks to see that the !expected booking and the amended booking are not equal

Assert.AreNotEqual(booking.DepartureDate, facade.readBooking(1).DepartureDate);

Assert.AreNotEqual(booking.EveningMeals, facade.readBooking(1).EveningMeals);

Assert.AreNotEqual(booking.Breakfast, facade.readBooking(1).Breakfast);

Assert.AreNotEqual(booking.CarHire, facade.readBooking(1).CarHire);

Assert.AreEqual(booking.CustomerReferenceNumber, facade.readBooking(1).CustomerReferenceNumber);

Assert.AreNotEqual(booking.EveningDietaryRequirements, facade.readBooking(1).EveningDietaryRequirements);

Assert.AreNotEqual(booking.BreakfastDietaryRequirements, facade.readBooking(1).BreakfastDietaryRequirements);

Assert.AreNotEqual(booking.CarHireStart, facade.readBooking(1).CarHireStart);

Assert.AreNotEqual(booking.CarHireEnd, facade.readBooking(1).CarHireEnd);

Assert.AreNotEqual(booking.Driver, facade.readBooking(1).Driver);

}

//this method tests the amendGuest method by creating a guest, and then amending the guest using the amendGuest method

//the proiperties are then checked to not be equal to what they were prior to the amendGuest method being called

[TestMethod]

public void TestAmendGuest()

{

File.WriteAllText("testBinaryFile.txt", String.Empty); //make sure we are working with a new file

Facade facade = new Facade(); //create a new facade

Guest guest = new Guest("john", "123abc", 21); //create a !expected guest

facade.createGuest("john", "123abc", 21); //create a guest to be amended

facade.amendGuest("notjohn", 22, "123abc"); //amend the guest

Assert.AreNotEqual(guest.Name, facade.readGuest("123abc").Name); //check the the !expected guest and the amended guest are not equal

Assert.AreNotEqual(guest.Age, facade.readGuest("123abc").Age);

}

}

}