**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

по дисциплине: «Объектно-ориентированное проектирование и   
программирование»

на тему: **«**Наследование классов**»**

Выполнил: студент гр. ИТИ-21

Мусафиров А.Ю.

Принял: преподаватель

Башаримов Ю. С.

Гомель 2023

**Цель работы**: изучить организацию наследования классов, узнать методы и механизмы наследования, разработать иерархию классов.

**Условие задания:**

1. Необходимо решить задачу, согласно варианту;

2. При создании классов руководствоваться *Code* *Convention*;

3. Весь код должен содержать элементы документирования;

4. При реализации иерархии классов использовать механизм наследования;

5. Разработать модульные тесты для верификации созданных классов;

6. Классы должен быть размещены в библиотеке классов;

7. Модульные тесты - в отдельном проекте;

**Задание:**

1. Разработать иерархию классов для хранения информации O приобретённых билетах на автобус, двигающийся по маршруту Гомель - Речица - Светлогорск - Жлобин - Бобруйск. Количество мест в автобусе ограничено и зависит от конкретного типа автобуса (Маз, Мерседес, Ман и т.д.)
2. Стоимость билета зависит от типа автобуса и расстояния, т.е. билет

Гомель-Речица будет дешевле, чем Речица - Жлобин (ч/з Светлогорск)

1. Обработать ситуацию, если данный билет не может быть продан, т.к. нет мест.
2. Учесть, что пассажиры могут ехать по части маршрута и на освободившиеся места могут быть проданы билеты.
3. Вывести информацию, сколько осталось свободных билетов по каждому из возможных маршрутов (*10* маршрутов)

**Ход работы**

В проекте мы реализуем иерархию классов следующим путём: от класса *Bus* будут наследоваться такие классы как *MAZ, Mersedes* и *Man.* В первом классе будет хранится базовая информация о транспорте, а в остальных классах более подробная, к примеру, начальная стоимость поездки или количество мест на определённом виде автобуса.

На рисунке 1 изображен процесс создания записи для поездки по вводимому маршруту:

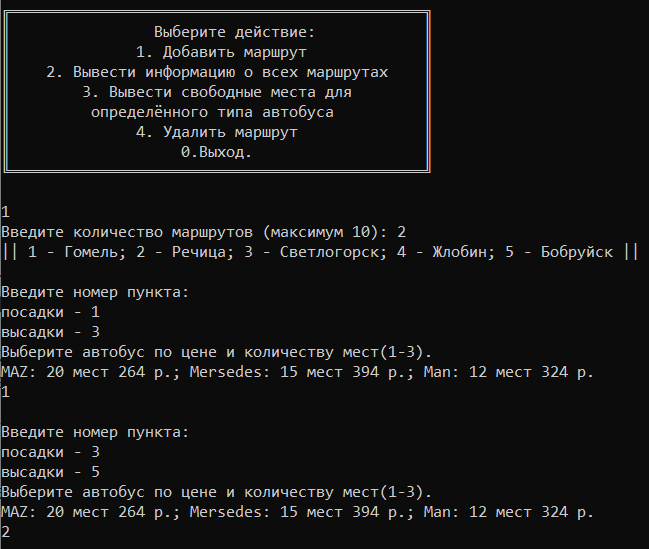


Рисунок 1 – Процесс создания записи.

Стоит отметить, что пункты назначения вводятся в виде цифр (1 - Гомель; 2 - Речица; 3 - Светлогорск; 4 - Жлобин; 5 - Бобруйск). Сделано это с целю упрощения вычисления «расстояния» маршрута. Путь находится путём вычитания от номера пункта высадки номера пункта посадки. Благодаря такому подходу можно с лёгкостью вычислить стоимость поездки только на основе данных о расстоянии. Найти данную стоимость можно по формуле (1):

|  |  |
| --- | --- |
|  | (1) |

где *x* – номер начального пункта;

*y* – номер конечного пункта;

*a−* произвольный коэффициент.

Полная же стоимость складывается из двух параметров: начальная стоимость поездки на определённом транспорте и нашей стоимости, найденной на основе данных о расстоянии. Нахождение полной стоимости представлено в формуле (2):

|  |  |
| --- | --- |
|  | (2) |

где *bus coast* – начальная стоимость поездки;

*coast* – стоимость поездки из расстояния.

Разницу в цене можно увидеть на рисунках 2 и 3:

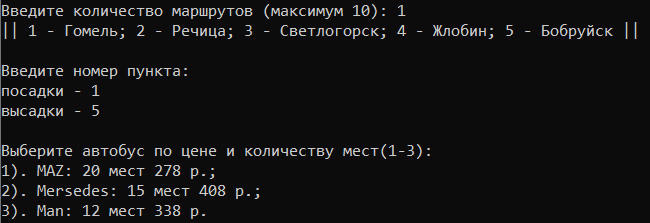


Рисунок 2 – Цены на автобусы на длинном маршруте.

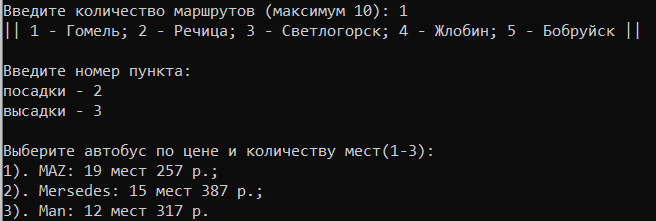


Рисунок 3 – Цены на автобусы на коротком маршруте.

Теперь купим билет на маршрут 1 − 3 (Гомель − Светлогорск) на первый тип автобуса. Изначально свободных мест было *20*, но т.к. мы забронировали место, то кол-во свободных мест уменьшилось. На рисунке 4 представлен результат работы метода по выводу информации об оставшихся свободных местах в автобусе №1 (автобус Маз):

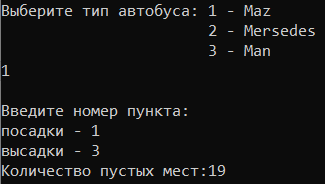


Рисунок 4 – Результат работы метода функции по выводу информации об оставшихся свободных местах.

На рисунке 5 изображена работа программы в случае, когда билетов на автобус по указанному маршруту не осталось, но попытка заказа всё равно была совершена:

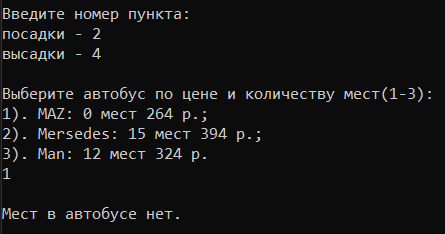


Рисунок 5 – Работа программы с исключительной ситуацией

В приложении А представлен код программы и код классов из библиотеки. В приложении Б находится *xml*-документация на три проекта данного решения.

На рисунке 6 уже изображён результат тестов, разработанных для проверки методов, созданных внутри класса для записей:

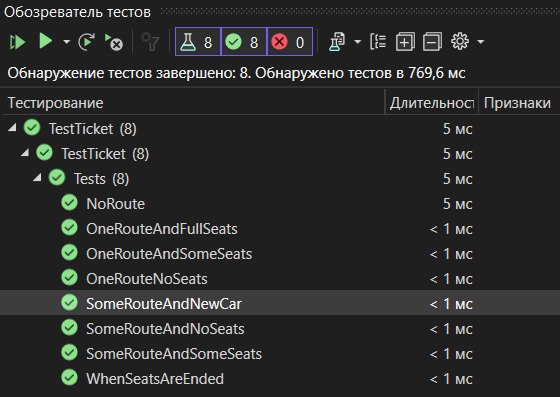


Рисунок 6 – 8пройденных модульных тестов.

На рисунке 7 показан журнал *Git*:

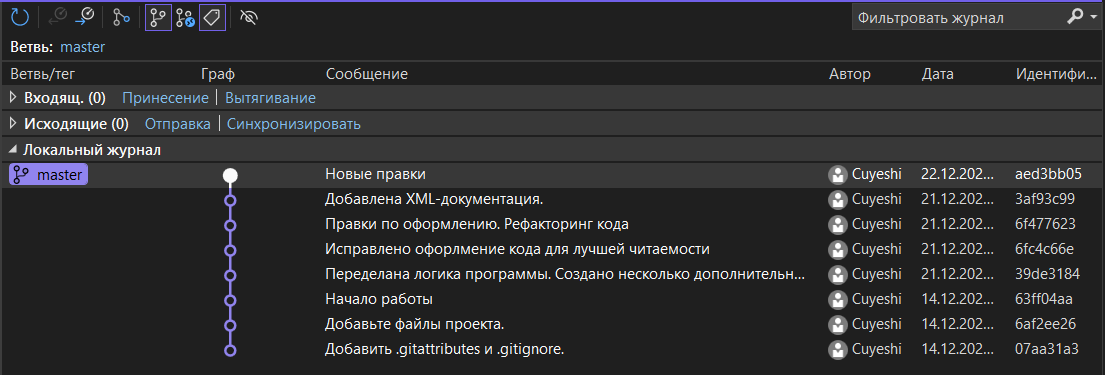


Рисунок 7 – Журнал *Git*.

Так как тесты все верны, то можно сделать вывод, что методы были разработаны правильно, а именно все действия, которые должны были выполнятся по условию.

**Вывод:** были получены навыки работы с механизмом наследования, изучена организация наследования классов, а также методы и механизмы наследования, разработана иерархия классов. Все условия задания выполнены.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Тексты кода программы и классов**

***Program.cs:***

using LibraryForClasses;

using System;

namespace Lab\_7

{

internal class Program

{

static void Main(string[] args)

{

MAZ MAZBus = new MAZ();

Mersedes MersedesBus = new Mersedes();

Man ManBus = new Man();

Ticket[] ticket = new Ticket[10];

for (int j = 0; j < 10; j++)

{

ticket[j] = new Ticket();

}

bool tryCatchExit = true, exit = false, validateValue1;

int freeSeats1, freeSeats2, freeSeats3, sale1, sale2, sale3, routesCount = 0, s = 0, n1 = 0, n2 = 0, n3 = 0, i;

while (tryCatchExit)

{

try

{

while (!exit)

{

Console.Clear();

Console.WriteLine("\n╔══════════════════════════════════════════════╗");

Console.WriteLine("║ Выберите действие: ║");

Console.WriteLine("║ 1. Добавить маршрут ║");

Console.WriteLine("║ 2. Вывести информацию о всех маршрутах ║");

Console.WriteLine("║ 3. Вывести свободные места для ║");

Console.WriteLine("║ определённого типа автобуса ║");

Console.WriteLine("║ 4. Удалить маршрут ║");

Console.WriteLine("║ 0.Выход. ║");

Console.WriteLine("╚══════════════════════════════════════════════╝\n");

int choice = 0;

string switchNumber = Console.ReadLine();

ExceptionTickets.ValidateInt(switchNumber, out choice);

switch (choice)

{

case 1:

Console.Write("Введите количество маршрутов (максимум 10): ");

string inputK = Console.ReadLine();

ExceptionTickets.ValidateIntForInput(inputK, out routesCount);

Console.WriteLine("|| 1 - Гомель; 2 - Речица; 3 - Светлогорск; 4 - Жлобин; 5 - Бобруйск ||");

for (i = 0; i < routesCount; i++)

{

validateValue1 = false;

Console.WriteLine("\nВведите номер пункта: ");

Console.Write("посадки - ");

string begin = Console.ReadLine();

Console.Write("высадки - ");

string end = Console.ReadLine();

ExceptionTickets.ValidateInt(begin, out ticket[s].begin);

ExceptionTickets.ValidateInt(end, out ticket[s].end);

if (ticket[s].end < ticket[s].begin || ticket[s].begin < 1 || ticket[s].begin > 5 || ticket[s].end < 1 || ticket[s].end > 5)

{

Console.WriteLine("\nТакого маршрута нет.");

}

else

{

while (!validateValue1)

{

int coast = ticket[s].CoastRoute();

freeSeats1 = SearchMin(MAZBus.freeSeats, ticket[s].begin, ticket[s].end);

freeSeats2 = SearchMin(MersedesBus.freeSeats, ticket[s].begin, ticket[s].end);

freeSeats3 = SearchMin(ManBus.freeSeats, ticket[s].begin, ticket[s].end);

sale1 = MAZBus.coastBus + coast;

sale2 = MersedesBus.coastBus + coast;

sale3 = ManBus.coastBus + coast;

Console.WriteLine("Выберите автобус по цене и количеству мест(1-3).");

Console.Write($"MAZ: {freeSeats1} мест {sale1} р.; Mersedes: {freeSeats2} мест {sale2} р.; " +

$"Man: {freeSeats3} мест {sale3} р.\n");

string switchValue = Console.ReadLine();

ExceptionTickets.ValidateInt(switchValue, out int value);

switch (value)

{

case 1:

if (freeSeats1 != 0)

{

ticket[s].type = "Maz";

MAZBus.tickets[n1] = ticket[s];

n1++;

MAZBus.FreeSeats(ticket[s].begin, ticket[s].end);

s++;

validateValue1 = true;

}

else

{

Console.WriteLine("\nМест в автобусе нет.");

}

break;

case 2:

if (freeSeats2 != 0)

{

ticket[s].type = "Mersedes";

MersedesBus.tickets[n2] = ticket[s];

n2++;

MersedesBus.FreeSeats(ticket[s].begin, ticket[s].end);

s++;

validateValue1 = true;

}

else

{

Console.WriteLine("\nМест в автобусе нет.");

}

break;

case 3:

if (freeSeats3 != 0)

{

ticket[s].type = "Man";

ManBus.tickets[n3] = ticket[s];

n3++;

ManBus.FreeSeats(ticket[s].begin, ticket[s].end);

s++;

validateValue1 = true;

}

else

{

Console.WriteLine("\nМест в автобусе нет.");

}

break;

default:

Console.WriteLine("\nТакого автобуса нет!");

break;

}

}

}

}

break;

case 2:

Console.WriteLine("Список купленных билетов:");

for (i = 0; i < routesCount; i++)

{

Console.WriteLine($"{i + 1}: автобус {ticket[i].type}, маршрут {ticket[i].begin} - {ticket[i].end}.");

}

Console.ReadKey();

break;

case 3:

Console.WriteLine("Выберите тип автобуса: 1 - Maz ");

Console.WriteLine(" 2 - Mersedes");

Console.WriteLine(" 3 - Man ");

string switchChoice1 = Console.ReadLine();

ExceptionTickets.ValidateInt(switchChoice1, out int busChoice1);

switch (busChoice1)

{

case 1:

RemainingPlaces(MAZBus);

Console.ReadKey();

break;

case 2:

RemainingPlaces(MersedesBus);

Console.ReadKey();

break;

case 3:

RemainingPlaces(ManBus);

Console.ReadKey();

break;

default:

Console.WriteLine("Некорректный выбор. Выберите существующую опцию.");

Console.ReadKey();

break;

}

break;

case 4:

Console.Write("Введите номер маршрута для его удаления: ");

string inputN = Console.ReadLine();

ExceptionTickets.ValidateInt(inputN, out int outputN);

if (outputN > 0 && outputN <= 10)

ticket[outputN - 1] = new Ticket();

else

Console.WriteLine("Вы ввели неверный номер маршрута!");

Console.ReadKey();

break;

case 0:

exit = true;

break;

default:

Console.WriteLine("Некорректный выбор. Выберите существующую опцию.");

Console.ReadKey();

break;

}

}

tryCatchExit = false;

}

catch (ExceptionTickets ex)

{

Console.WriteLine($"Ошибка ввода: {ex.Message}\n");

Console.ReadKey();

}

}

}

/// <summary>

/// Метод поиска минимального внутри массива.

/// </summary>

/// <param name="tickets"></param>

/// <param name="begin"></param>

/// <param name="end"></param>

/// <returns></returns>

static int SearchMin(int[] tickets, int begin, int end)

{

int min = tickets[begin - 1];

for (int i = begin - 1; i < end; i++)

{

if (tickets[i] < min)

{

min = tickets[i];

}

}

return min;

}

/// <summary>

/// Метод для вывода информации о свободных местах в указываемом транспорте.

/// </summary>

/// <param name="bus"></param>

static void RemainingPlaces(Bus bus)

{

try

{

int begin = 0, end = 0;

Console.WriteLine("\nВведите номер пункта: ");

Console.Write("посадки - ");

string input1 = Console.ReadLine();

Console.Write("высадки - ");

string input2 = Console.ReadLine();

ExceptionTickets.ValidateInt(input1, out begin);

ExceptionTickets.ValidateInt(input2, out end);

//Console.Write("Начало (1-5): ");

//Console.Write("Конец (1-5): ");

if (end > begin && begin >= 1 && begin <= 5 && end >= 1 && end <= 5)

{

int min = Program.SearchMin(bus.freeSeats, begin, end);

Console.Write("Количество пустых мест:");

Console.WriteLine(min);

}

else

{

Console.WriteLine("Такого маршрута нет.");

}

}

catch (ExceptionTickets ex)

{

Console.WriteLine($"Ошибка ввода: {ex.Message}\n");

}

}

}

}

***Ticket.cs:***

namespace LibraryForClasses

{

/// <summary>

/// Класс билет для хранения информации о билете.

/// </summary>

public class Ticket

{

/// <summary>

/// Поле, хранящее значение номера начала вводимого маршрута.

/// </summary>

public int begin;

/// <summary>

/// Поле, хранящее значение номера конца вводимого маршрута.

/// </summary>

public int end;

/// <summary>

/// Поле, хранящее значение типа автобуса для билета.

/// </summary>

public string type;

/// <summary>

/// Конструктор для задания параметров объекта класса Ticket.

/// </summary>

public Ticket()

{

begin = 0;

end = 0;

type = "нет";

}

/// <summary>

/// Метод подсчёта стоимости билета.

/// </summary>

/// <returns></returns>

public int CoastRoute()

{

int coast = (end - begin) \* 7;

return coast;

}

}

}

***Bus.cs:***

namespace LibraryForClasses

{

/// <summary>

/// Абстрактный класс для наследования к типам автобуса.

/// </summary>

abstract public class Bus

{

/// <summary>

/// Поле, хранящее значение количества мест в абстрактном классе.

/// </summary>

public int seats = 0;

/// <summary>

/// Массив билетов.

/// </summary>

public Ticket[] tickets = new Ticket[10];

/// <summary>

/// Массив переменных, хранящих значения о количестве свободных мест в автобусе.

/// </summary>

public int[] freeSeats = new int[5];

/// <summary>

/// Конструктор для абстракторного класса.

/// </summary>

public Bus()

{

for (int i = 0; i < 5; i++)

{

freeSeats[i] = seats;

}

}

/// <summary>

/// Метод подсчёта свободных мест по маршруту.

/// </summary>

/// <param name="begin"></param>

/// <param name="end"></param>

/// <returns></returns>

public int[] FreeSeats(int begin, int end)

{

int count = 0;

for (int i = begin - 1; i < end; i++)

{

if (freeSeats[i] == 0)

{

count++;

}

}

if (count == 0)

{

for (int i = begin - 1; i < end - 1; i++)

{

freeSeats[i]--;

}

}

return freeSeats;

}

}

}

***Maz.cs:***

namespace LibraryForClasses

{

/// <summary>

/// Класс потомок для автобусов типа МАЗ.

/// </summary>

public class MAZ : Bus

{

/// <summary>

/// Поле, хранящее значение базовой стоимости проезда на автобусе типа MAZ.

/// </summary>

public int coastBus = 250;

/// <summary>

/// Поле, хранящее значение количества мест в автобусе типа MAZ.

/// </summary>

public int carSeats = 20;

/// <summary>

/// Конструктор для задания параметров объекта класса-потомка MAZ.

/// </summary>

public MAZ() : base()

{

for (int i = 0; i < 5; i++)

{

freeSeats[i] = carSeats;

}

}

}

}

***Mersedes.cs:***

namespace LibraryForClasses

{

/// <summary>

/// Класс потомок для автобусов типа Мерседес.

/// </summary>

public class Mersedes : Bus

{

/// <summary>

/// Поле, хранящее значение базовой стоимости проезда на автобусе типа Mersedes.

/// </summary>

public int coastBus = 380;

/// <summary>

/// Поле, хранящее значение количества мест в автобусе типа Mersedes.

/// </summary>

public int carSeats = 15;

/// <summary>

/// Конструктор для задания параметров объекта класса-потомка Mersedes.

/// </summary>

public Mersedes() : base()

{

for (int i = 0; i < 5; i++)

{

freeSeats[i] = carSeats;

}

}

}

}

***Man.cs:***

namespace LibraryForClasses

{

/// <summary>

/// Класс потомок для автобусов типа Мэн.

/// </summary>

public class Man : Bus

{

/// <summary>

/// Поле, хранящее значение базовой стоимости проезда на автобусе типа Man.

/// </summary>

public int coastBus = 310;

/// <summary>

/// Поле, хранящее значение количества мест в автобусе типа Man.

/// </summary>

public int carSeats = 12;

/// <summary>

/// Конструктор для задания параметров объекта класса-потомка Man.

/// </summary>

public Man() : base()

{

for (int i = 0; i < 5; i++)

{

freeSeats[i] = carSeats;

}

}

}

}

***ExceptionTickets.cs:***

using System;

namespace LibraryForClasses

{

/// <summary>

/// Класс исключений для задачи.

/// </summary>

public class ExceptionTickets : Exception

{

/// <summary>

/// Конструктор для создания собственных исключений.

/// </summary>

/// <param name="message"></param>

public ExceptionTickets(string message) : base(message) { }

/// <summary>

/// Метод исключения для переменных типа int.

/// </summary>

/// <param name="input"></param>

/// <param name="result"></param>

/// <exception cref="ExceptionTickets"></exception>

public static void ValidateInt(string input, out int result)

{

if (string.IsNullOrWhiteSpace(input))

{

throw new ExceptionTickets("Строка пуста!");

}

if (!Int32.TryParse(input, out result))

{

throw new ExceptionTickets("Строка содержит вещественное число или текст!");

}

if (result < 0)

{

throw new ExceptionTickets("Введённое число меньше нуля!");

}

}

/// <summary>

/// Метод исключения для переменных типа int для случая ввода количества маршрутов.

/// </summary>

/// <param name="input"></param>

/// <param name="result"></param>

/// <exception cref="ExceptionTickets"></exception>

public static void ValidateIntForInput(string input, out int result)

{

if (string.IsNullOrWhiteSpace(input))

{

throw new ExceptionTickets("Строка пуста!");

}

if (!Int32.TryParse(input, out result))

{

throw new ExceptionTickets("Строка содержит вещественное число или текст!");

}

if (result < 0 || result > 10)

{

throw new ExceptionTickets("Введённое число меньше нуля или больше 10!");

}

}

}

}

***Test.cs:***

using Microsoft.VisualStudio.TestTools.UnitTesting;

using LibraryForClasses;

namespace TestTicket

{

/// <summary>

/// Класс для тестирования программы.

/// </summary>

[TestClass]

public class Tests

{

/// <summary>

/// Метод сравнения массивов.

/// </summary>

/// <param name="a"></param>

/// <param name="b"></param>

/// <returns></returns>

public bool CompareArray(int[] a, int[] b)

{

if (a.Length != b.Length)

{

return false;

}

for (int i = 0; i < a.Length; i++)

{

if (a[i] != b[i])

{

return false;

}

}

return true;

}

/// <summary>

/// Тест при одном маршруте и свободном автобусе.

/// </summary>

[TestMethod]

public void OneRouteAndFullSeats()

{

Man bus = new Man();

int begin = 1, end = 2;

int[] result = bus.FreeSeats(begin, end);

int[] array = new int[] { 12, 13, 13, 13, 13 };

Assert.IsTrue(CompareArray(result, array));

}

/// <summary>

/// Тест для одного маршрута и автобуса с несколькими занятами местами.

/// </summary>

[TestMethod]

public void OneRouteAndSomeSeats()

{

Man bus = new Man();

for (int i = 0; i < 5; i++)

{

bus.freeSeats[i] = 1;

}

int begin = 1, end = 4;

int[] result = bus.FreeSeats(begin, end);

int[] array = new int[] { 0, 0, 0, 1, 1 };

Assert.IsTrue(CompareArray(result, array));

}

/// <summary>

/// Тест для одного маршрута и заполненного автобуса.

/// </summary>

[TestMethod]

public void OneRouteNoSeats()

{

Man bus = new Man();

for (int i = 0; i < 5; i++)

{

bus.freeSeats[i] = 0;

}

int begin = 1, end = 5;

int[] result = bus.FreeSeats(begin, end);

int[] array = new int[] { -1, -1, -1, -1, 0 };

Assert.IsFalse(CompareArray(result, array));

}

/// <summary>

/// Тест для нескольких маршрутов и автобуса с несколькими свободными местами.

/// </summary>

[TestMethod]

public void SomeRouteAndSomeSeats()

{

Man bus = new Man();

for (int i = 0; i < 5; i++)

{

bus.freeSeats[i] = 3;

}

int begin1 = 1, end1 = 4, begin2 = 3, end2 = 5;

bus.FreeSeats(begin1, end1);

int[] result = bus.FreeSeats(begin2, end2);

int[] array = new int[] { 2, 2, 1, 2, 3 };

Assert.IsTrue(CompareArray(result, array));

}

/// <summary>

/// Тест для нескольких маршрутов, но в автобусе другого типа.

/// </summary>

[TestMethod]

public void SomeRouteAndNewCar()

{

MAZ bus = new MAZ();

int begin1 = 1, end1 = 3, begin2 = 1, end2 = 5;

bus.FreeSeats(begin1, end1);

int[] result = bus.FreeSeats(begin2, end2);

int[] array = new int[] { 18, 18, 19, 19, 20 };

Assert.IsTrue(CompareArray(result, array));

}

/// <summary>

/// Тест при нескольких маршрутов и заполненном автобусе.

/// </summary>

[TestMethod]

public void SomeRouteAndNoSeats()

{

Man bus = new Man();

for (int i = 0; i < 5; i++)

{

bus.freeSeats[i] = 0;

}

int begin1 = 1, end1 = 4, begin2 = 3, end2 = 5, begin3 = 2, end3 = 4;

bus.FreeSeats(begin1, end1);

bus.FreeSeats(begin2, end2);

int[] result = bus.FreeSeats(begin3, end3);

int[] array = new int[] { 0, 0, 0, 0, 0 };

Assert.IsTrue(CompareArray(result, array));

}

/// <summary>

/// Тест без маршрутов.

/// </summary>

[TestMethod]

public void NoRoute()

{

Man bus = new Man();

int begin = 1, end = 1;

int[] result = bus.FreeSeats(begin, end);

int[] array = new int[] {13, 13, 13, 13, 13 };

Assert.IsTrue(CompareArray(result, array));

}

/// <summary>

/// Тест, когда сидения заканчиваются не сразу.

/// </summary>

[TestMethod]

public void WhenSeatsAreEnded()

{

Man bus = new Man();

for (int i = 0; i < 5; i++)

{

bus.freeSeats[i] = 2;

}

int begin1 = 1, end1 = 4, begin2 = 3, end2 = 5, begin3 = 1, end3 = 5;

bus.FreeSeats(begin1, end1);

bus.FreeSeats(begin2, end2);

int[] result = bus.FreeSeats(begin3, end3);

int[] array = new int[] { 0, 0, -1, 0, 2 };

Assert.IsFalse(CompareArray(result, array));

}

}

}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

***XML-*документация**

***Lab-7.xml:***

<?xml version="1.0"?>

<doc>

<assembly>

<name>Lab-7</name>

</assembly>

<members>

<member name="M:Lab\_7.Program.SearchMin(System.Int32[],System.Int32,System.Int32)">

<summary>

Метод поиска минимального внутри массива.

</summary>

<param name="tickets"></param>

<param name="begin"></param>

<param name="end"></param>

<returns></returns>

</member>

<member name="M:Lab\_7.Program.RemainingPlaces(LibraryForClasses.Bus,System.Int32,System.Int32)">

<summary>

Метод для вывода информации о свободных местах в указываемом транспорте.

</summary>

<param name="bus"></param>

<param name="begin"></param>

<param name="end"></param>

</member>

</members>

</doc>

***LibraryForClasses.xml:***

<?xml version="1.0"?>

<doc>

<assembly>

<name>LibraryForClasses</name>

</assembly>

<members>

<member name="T:LibraryForClasses.Ticket">

<summary>

Класс билет для хранения информации о билете.

</summary>

</member>

<member name="F:LibraryForClasses.Ticket.Begin">

<summary>

Поле, хранящее значение номера начала вводимого маршрута.

</summary>

</member>

<member name="F:LibraryForClasses.Ticket.End">

<summary>

Поле, хранящее значение номера конца вводимого маршрута.

</summary>

</member>

<member name="M:LibraryForClasses.Ticket.#ctor">

<summary>

Конструктор для задания параметров объекта класса Ticket.

</summary>

</member>

<member name="M:LibraryForClasses.Ticket.CoastRoute">

<summary>

Метод подсчёта стоимости билета.

</summary>

<returns></returns>

</member>

<member name="T:LibraryForClasses.Mersedes">

<summary>

Класс потомок для автобусов типа Мерседес.

</summary>

</member>

<member name="F:LibraryForClasses.Mersedes.coastBus">

<summary>

Поле, хранящее значение базовой стоимости проезда на автобусе типа Mersedes.

</summary>

</member>

<member name="F:LibraryForClasses.Mersedes.carSeats">

<summary>

Поле, хранящее значение количества мест в автобусе типа Mersedes.

</summary>

</member>

<member name="M:LibraryForClasses.Mersedes.#ctor">

<summary>

Конструктор для задания параметров объекта класса-потомка Mersedes.

</summary>

</member>

<member name="T:LibraryForClasses.MAZ">

<summary>

Класс потомок для автобусов типа МАЗ.

</summary>

</member>

<member name="F:LibraryForClasses.MAZ.coastBus">

<summary>

Поле, хранящее значение базовой стоимости проезда на автобусе типа MAZ.

</summary>

</member>

<member name="F:LibraryForClasses.MAZ.carSeats">

<summary>

Поле, хранящее значение количества мест в автобусе типа MAZ.

</summary>

</member>

<member name="M:LibraryForClasses.MAZ.#ctor">

<summary>

Конструктор для задания параметров объекта класса-потомка MAZ.

</summary>

</member>

<member name="T:LibraryForClasses.Man">

<summary>

Класс потомок для автобусов типа Мэн.

</summary>

</member>

<member name="F:LibraryForClasses.Man.coastBus">

<summary>

Поле, хранящее значение базовой стоимости проезда на автобусе типа Man.

</summary>

</member>

<member name="F:LibraryForClasses.Man.carSeats">

<summary>

Поле, хранящее значение количества мест в автобусе типа Man.

</summary>

</member>

<member name="M:LibraryForClasses.Man.#ctor">

<summary>

Конструктор для задания параметров объекта класса-потомка Man.

</summary>

</member>

<member name="T:LibraryForClasses.Bus">

<summary>

Абстрактный класс для наследования к типам автобуса.

</summary>

</member>

<member name="F:LibraryForClasses.Bus.seats">

<summary>

Поле, хранящее значение количества мест в абстрактном классе.

</summary>

</member>

<member name="F:LibraryForClasses.Bus.tickets">

<summary>

Массив билетов.

</summary>

</member>

<member name="F:LibraryForClasses.Bus.freeSeats">

<summary>

Массив переменных, хранящих значения о количестве свободных мест в автобусе.

</summary>

</member>

<member name="M:LibraryForClasses.Bus.#ctor">

<summary>

Конструктор для абстракторного класса.

</summary>

</member>

<member name="M:LibraryForClasses.Bus.FreeSeats(System.Int32,System.Int32)">

<summary>

Метод подсчёта свободных мест по маршруту.

</summary>

<param name="begin"></param>

<param name="end"></param>

<returns></returns>

</member>

<member name="T:LibraryForClasses.ExceptionTickets">

<summary>

Класс исключений для задачи.

</summary>

</member>

<member name="M:LibraryForClasses.ExceptionTickets.#ctor(System.String)">

<summary>

Конструктор для создания собственных исключений.

</summary>

<param name="message"></param>

</member>

<member name="M:LibraryForClasses.ExceptionTickets.ValidateInt(System.String,System.Int32@)">

<summary>

Метод исключения для переменных типа int.

</summary>

<param name="input"></param>

<param name="result"></param>

<exception cref="T:LibraryForClasses.ExceptionTickets"></exception>

</member>

</members>

</doc>

***TestTicket.xml:***

<?xml version="1.0"?>

<doc>

<assembly>

<name>TestTicket</name>

</assembly>

<members>

<member name="T:TestTicket.Tests">

<summary>

Класс для тестирования программы.

</summary>

</member>

<member name="M:TestTicket.Tests.CompareArray(System.Int32[],System.Int32[])">

<summary>

Метод сравнения массивов.

</summary>

<param name="a"></param>

<param name="b"></param>

<returns></returns>

</member>

<member name="M:TestTicket.Tests.OneRouteAndFullSeats">

<summary>

Тест при одном маршруте и свободном автобусе.

</summary>

</member>

<member name="M:TestTicket.Tests.OneRouteAndSomeSeats">

<summary>

Тест для одного маршрута и автобуса с несколькими занятами местами.

</summary>

</member>

<member name="M:TestTicket.Tests.OneRouteNoSeats">

<summary>

Тест для одного маршрута и заполненного автобуса.

</summary>

</member>

<member name="M:TestTicket.Tests.SomeRouteAndSomeSeats">

<summary>

Тест для нескольких маршрутов и автобуса с несколькими свободными местами.

</summary>

</member>

<member name="M:TestTicket.Tests.SomeRouteAndNewCar">

<summary>

Тест для нескольких маршрутов, но в автобусе другого типа.

</summary>

</member>

<member name="M:TestTicket.Tests.SomeRouteAndNoSeats">

<summary>

Тест при нескольких маршрутов и заполненном автобусе.

</summary>

</member>

<member name="M:TestTicket.Tests.NoRoute">

<summary>

Тест без маршрутов.

</summary>

</member>

<member name="M:TestTicket.Tests.WhenSeatsAreEnded">

<summary>

Тест, когда сидения заканчиваются не сразу.

</summary>

</member>

</members>

</doc>