АКТ(ф) СПбГУТ

**ОТЧЕТ**

по практическим работам

Поддержка и тестирование программных модулей

Студент ИСПП-21 30.11.2024 С.А.Миклякова

Преподаватель 30.11.2024 Р.В.Садовский

Архангельск 2024

# Практическая работа №1

## Изучение отладочных возможностей сред программирования

1. **Цель работы**
   1. Познакомиться с основными инструментами отладки и профилирования в Visual Studio 2022;
   2. Научиться ставить точки останова (breakpoints), использовать пошаговое выполнение кода, инспектировать значения переменных и применять профилировщик для анализа производительности программы.
2. **Ответы на контрольные вопросы**
   1. Точки останова в программировании позволяют программисту остановить выполнение программы в определенном месте для анализа ее состояния. Они позволяют устанавливать точки останова на определенные строки кода и просматривать значения переменных в этой точке.
   2. "Step Over" позволяет выполнить текущую строку кода и перейти к следующей строке без захода внутрь функций. "Step Into" позволяет заходить внутрь вызываемых функций и выполнять код пошагово. "Step Out" позволяет завершить выполнение текущей функции и вернуться к вызывающей функции.
   3. Окно "Call Stack" позволяет отслеживать последовательность вызовов функций во время выполнения программы. Оно показывает стек вызовов, позволяя программисту понять, какие функции были вызваны перед текущей точкой выполнения программы.
   4. Профилировщик в Visual Studio используется для анализа производительности программы и выявления узких мест в коде. Он позволяет отслеживать время выполнения различных участков кода, использование памяти, CPU и другие характеристики. Профилировщик помогает оптимизировать код и улучшить производительность приложения.
3. **Вывод**
   1. В ходе проделанной работы были изучены основные инструменты отладки и профилирования в Visual Studio 2022;
   2. В ходе проделанной работы были получены навыки, как ставить точки останова (breakpoints), использовать пошаговое выполнение кода, инспектировать значения переменных и применять профилировщик для анализа производительности программы.

# Практическая работа №2

## Изучение процесса формирования набора тестовых данных

1. **Цель работы**
   1. Изучить основные методы и техники формирования тестовых данных для тестирования программного обеспечения;
   2. Научиться генерировать наборы тестовых данных для различных типов тестирования.
2. **Ход работы**
   1. Создали консольное приложение на C#, которое принимает число и выводит, является ли оно простым.

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите число:");

if (int.TryParse(Console.ReadLine(), out int number))

{

if (IsPrime(number))

{

Console.WriteLine($"{number} - простое число.");

}

else

{

Console.WriteLine($"{number} - не простое число.");

}

}

else

{

Console.WriteLine("Введены некорректные данные.");

}

Console.ReadKey();

}

static bool IsPrime(int number)

{

if (number <= 1) return false;

for (int i = 2; i \* i <= number; i++)

{

if (number % i == 0) return false;

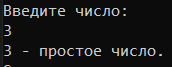
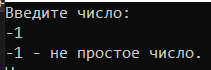
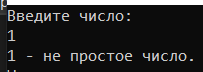
}

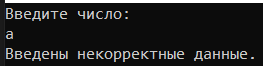
return true;

}

}

* 1. Сформировали набор тестовых данных для этого приложения с использованием граничных значений и эквивалентных классов.





|  |  |  |
| --- | --- | --- |
| **Тестовое значение** | **Ожидаемый результат** | **Примечание** |
| 1 | Не простое | Не простое число |
| 2 | Простое | Минимальное простое |
| 3 | Простое | Простое число |
| 4 | Не простое | Составное число |
| -1 | Не простое | Отрицательное число |
| 0 | Не простое | 0 не является простым |
| а | Не число | Нечисловые данные |
| 1000000 | Не простое | Большое значение |

* 1. Реализовали функцию, которая генерирует случайные числа в диапазоне (например, от 1 до 100) и передает их в тесты для проверки поведения программы.

class Program

{

static void Main(string[] args)

{

for (int i = 0; i < 100; i++)

{

int randomNum = GenerateRandomNumber(1, 100);

if (IsPrime(randomNum))

{

Console.WriteLine($"{randomNum} - простое число.");

}

else

{

Console.WriteLine($"{randomNum} - не простое число.");

}

}

Console.ReadKey();

}

static bool IsPrime(int number)

{

if (number <= 1) return false;

for (int i = 2; i \* i <= number; i++)

{

if (number % i == 0) return false;

}

return true;

}

static int GenerateRandomNumber(int min, int max)

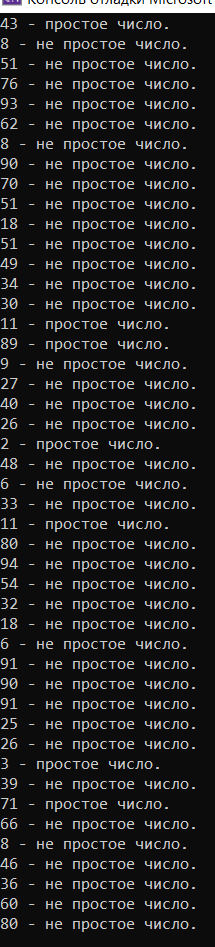
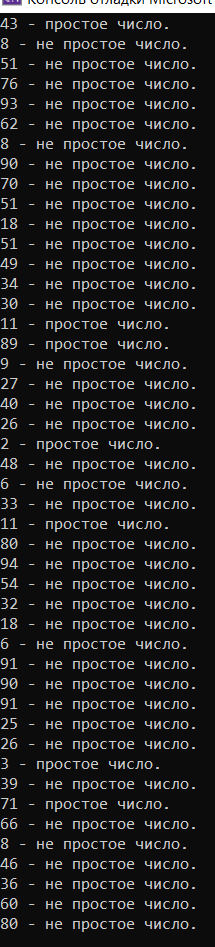
{

Random random = new Random();

return random.Next(min, max + 1);

}

}



* 1. Модифицировали программу из задания 1 так, чтобы она могла обрабатывать массив чисел (например, массив из 1000 чисел).

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите количество чисел в массиве:");

if (int.TryParse(Console.ReadLine(), out int arraySize))

{

int[] numbers = new int[arraySize];

Console.WriteLine("Введите числа, разделяя их пробелом:");

string[] input = Console.ReadLine().Split(' ');

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

{

if (int.TryParse(input[i], out numbers[i]))

{

if (IsPrime(numbers[i]))

{

Console.WriteLine($"{numbers[i]} - простое число.");

}

else

{

Console.WriteLine($"{numbers[i]} - не простое число.");

}

}

else

{

Console.WriteLine($"Некорректные данные в позиции {i + 1}.");

}

}

}

else

{

Console.WriteLine("Введены некорректные данные для размера массива.");

}

Console.ReadKey();

}

static bool IsPrime(int number)

{

if (number <= 1) return false;

for (int i = 2; i \* i <= number; i++)

{

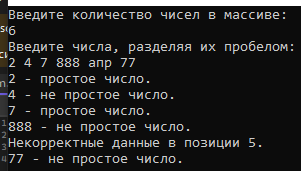
if (number % i == 0) return false;

}

return true;

}

}



* 1. Сгенерировали большой набор данных и провели нагрузочное тестирование программы.

class Program

{

static void Main(string[] args)

{

int[] numbers = GenerateRandomArray(10000, 1, 1000000000);

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

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

{

if (IsPrime(numbers[i]))

{

Console.WriteLine($"{numbers[i]} - простое число.");

}

else

{

Console.WriteLine($"{numbers[i]} - не простое число.");

}

}

stopwatch.Stop();

Console.WriteLine($"Время выполнения: {stopwatch.ElapsedMilliseconds} мс");

Console.ReadKey();

}

static bool IsPrime(int number)

{

if (number <= 1) return false;

for (int i = 2; i \* i <= number; i++)

{

if (number % i == 0) return false;

}

return true;

}

static int[] GenerateRandomArray(int size, int min, int max)

{

int[] array = new int[size];

Random random = new Random();

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

{

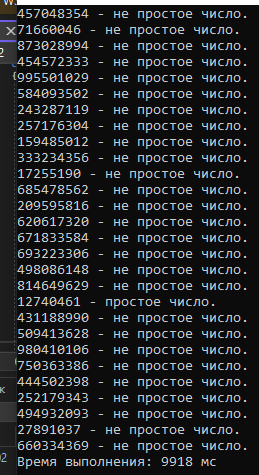
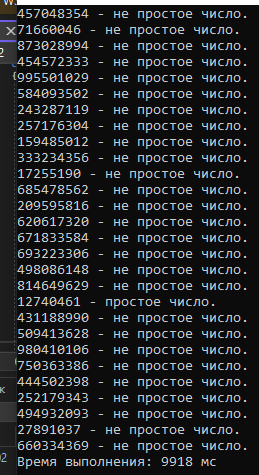
array[i] = random.Next(min, max + 1);

}

return array;

}

}



* 1. Добавили обработку исключений, если пользователь вводит некорректные данные.

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите число или 'массив' для обработки массива чисел:");

string input = Console.ReadLine();

try

{

if (input.ToLower() == "массив")

{

ProcessArray();

}

else

{

int number = int.Parse(input);

if (IsPrime(number))

{

Console.WriteLine($"{number} - простое число.");

}

else

{

Console.WriteLine($"{number} - не простое число.");

}

}

}

catch (FormatException)

{

Console.WriteLine("Введены некорректные данные. Введите целое число или 'массив'.");

}

catch (OverflowException)

{

Console.WriteLine("Введенное число слишком большое.");

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка: {ex.Message}");

}

Console.ReadKey();

}

static bool IsPrime(int number)

{

if (number <= 1) return false;

for (int i = 2; i \* i <= number; i++)

{

if (number % i == 0) return false;

}

return true;

}

static void ProcessArray()

{

Console.WriteLine("Введите количество чисел в массиве:");

if (int.TryParse(Console.ReadLine(), out int arraySize))

{

int[] numbers = new int[arraySize];

Console.WriteLine("Введите числа, разделяя их пробелом:");

string[] input = Console.ReadLine().Split(' ');

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

{

if (int.TryParse(input[i], out numbers[i]))

{

if (IsPrime(numbers[i]))

{

Console.WriteLine($"{numbers[i]} - простое число.");

}

else

{

Console.WriteLine($"{numbers[i]} - не простое число.");

}

}

else

{

Console.WriteLine($"Некорректные данные в позиции {i + 1}.");

}

}

}

else

{

Console.WriteLine("Введены некорректные данные для размера массива.");

}

}

static int[] GenerateRandomArray(int size, int min, int max)

{

int[] array = new int[size];

Random random = new Random();

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

{

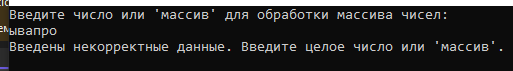
array[i] = random.Next(min, max + 1);

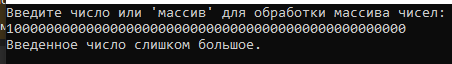
}

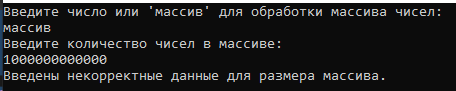
return array;

}

}







1. **Ответы на контрольные вопросы**
   1. Набор тестовых данных — это набор входных данных и ожидаемых результатов, которые используются для проверки правильности работы программы или функции.
   2. Граничные значения — это значения, которые находятся на границе допустимых входных данных.
   3. Классы эквивалентности — это группы входных данных, которые, как ожидается, будут обрабатываться программой одинаково.
2. **Вывод**
   1. В ходе практической работы были изучены основные методы и техники формирования тестовых данных для тестирования программного обеспечения;
   2. В ходе практической работы научились генерировать наборы тестовых данных для различных типов тестирования.

# Практическая работа №3

## Деление классов тестов по видам, типам и областям

1. **Цель работы**
   1. Научиться разделять тесты на различные виды и категории, а также понимать их назначение и особенности.
2. **Ход работы**
   1. Составили таблицу с классификацией тестов на различные виды и категории.

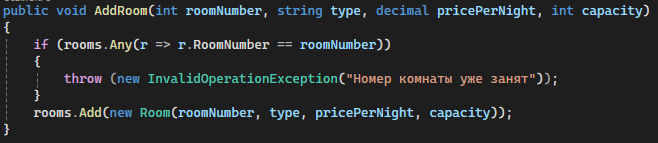
|  |  |
| --- | --- |
| **Вид теста и его описание** | **Пример** |
| По уровню тестирования | |
| Модульное тестирование | Позволяет проверить компоненты на уровне модулей. Тестировщик проверяет каждый исходный код и сопоставляет его с ожидаемым результатом. |
| Интеграционное тестирование | Предназначено для проверки связи между модулями. Этот уровень помогает выявить ошибки, которые мешают взаимодействовать интегрированным компонентам. Для его проведения используют несколько подходов, таких как «сверху вниз», «снизу вверх» и «сэндвич». Например, разработанная игра может поддерживать социальную сеть Facebook, чтобы можно было играть с друзьями. В этом случае, необходимо протестировать корректно ли работает вход в социальную сеть через игру, можем ли мы видеть список друзей и т.д. |
| Системное тестирование | Проверяет корректность работы всей системы. Это тестирование даёт полный отчёт о производительности и соответствии системы заданным бизнес-требованиям. |
| По видам тестирования | |
| Функциональное тестирование | Процесс проверки различных функций программного обеспечения для выявления ошибок и убеждения в том, что они работают согласно спецификациям.  Примеры:   1. Проверка форм на сайте: валидация данных, обязательные поля, правильность сообщений об ошибках. 2. Тестирование навигации: переход между страницами, гиперссылки, кнопки, меню и другие элементы управления. 3. Тестирование функционала поиска: алгоритмы фильтрации, сортировки, пагинация. |
| Нефункциональное тестирование | Относится к аспектам программного обеспечения, которые не связаны с конкретными функциями.  Примеры:   1. Тестирование производительности: измерение времени отклика сервера, обработки запросов и времени загрузки страниц. 2. Тестирование безопасности: проверка на уязвимости, атаки на отказ в обслуживании (DDoS), SQL-инъекции, перехват данных. |
| По областям | |
| Тестирование совместимости | Направлено на проверку совместимости продукта с операционными системами, браузерами, сетевыми окружениями, аппаратными конфигурациями, и т.п. Приложение должно работать во всех предусмотренных в его документации окружениях.  Например, Windows-приложение должно быть совместимым со всеми распространенными версиями ОС Windows. Если это веб-приложение, оно должно без проблем открываться во всех распространенных браузерах. |
| Тестирование доступности | Проверка доступности, или легкости пользования.  Например, приложение должно быть проверено на удобство для слабослышащих и слабовидящих людей, и при необходимости откорректировано. |
| Нагрузочное тестирование | На систему подается нагрузка в виде запросов/одновременных «пользователей», которая позволяет оценить, какое количество нагрузки система способна обработать до того как начнет ухудшать свою производительность. |
| Тестирование черного ящика | Тип тестирования программного обеспечения, при котором тестировщик не интересуется внутренними знаниями или деталями реализации программного обеспечения, а сосредотачивается на проверке функциональности на основе предоставленных спецификаций или требований. |
| Тестирование белого ящика | Тестировщик имеет доступ к исходному коду и использует эти знания для разработки тестовых сценариев, которые могут проверить корректность программного обеспечения на уровне кода. |
| Тестирование серого ящика | метод тестирования программного обеспечения, представляющий собой комбинацию метода тестирования черного ящика и метода тестирования белого ящика . |
| Тестирование локализации | Проводится, чтобы проверить качество перевода продукта с одного языка на другой. Также проверяется совместимость приложения со стандартами различных регионов и правильность оформления документации и вспомогательных файлов. |
| Тестирование надежности | Проверка выносливости приложения, время работы без единого отказа. Например, нужно не допустить ситуации, когда важная личная информация пользователей хранится в базе данных под управлением приложения, и через время удаляется, из-за каких-то ошибок в коде. |
| Тестирование масштабируемости | Определяется как способность сети, системы, приложения, продукта или процесса корректно выполнять функции при изменении размера или объёма системы для удовлетворения растущих потребностей. Это гарантирует, что программный продукт сможет справиться с запланированным увеличением пользовательского трафика, объёма данных, частоты транзакций и другим. |
| Тестирование стабильности | Фокусируется на максимальной нагрузке на программный компонент, проверяет качество и поведение программного обеспечения при различных параметрах окружающей среды. Оно определяется как способность продукта продолжать функционировать в течение длительного времени без сбоев. |

1. **Вывод**
   1. В ходе лабораторной работы было выполнено разделение тестов на различные виды и категории, а также понимание их назначения и особенностей.

# Практическая работа №4

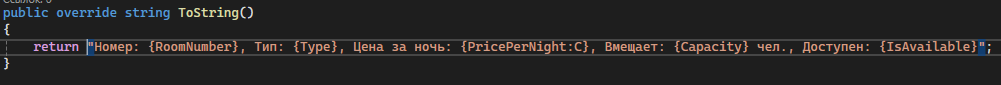
## Инспектирование кода программы.

1. **Цель работы**
   1. Изучить методы инспектирования кода программы и научиться применять их на практике для повышения качества программного обеспечения.
2. **Ход работы**
   1. В класс Hotel в метод AddRoom добавить обработку исключений для того, чтобы клиент не мог забронировать тот номер комнаты, который уже забронирован.

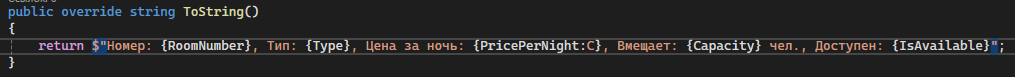


* 1. В классе Hotel в методе ToString была использована интерполяция, но с неправильным синтаксисом.

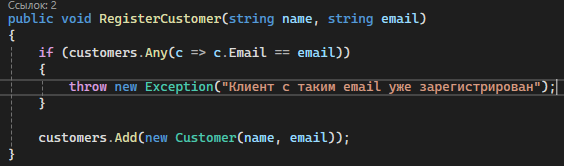
Было



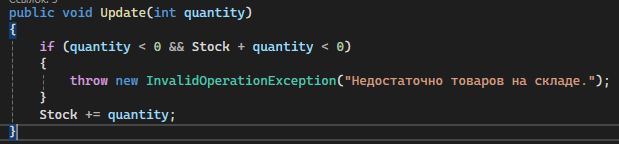
Стало



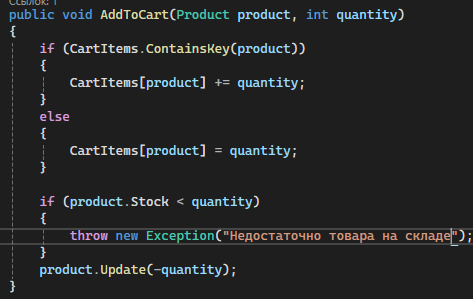
* 1. В классы OnlineStore и HotelManagementSystem в метод RegisterCustomer добавить обработку исключения на уникальность email.



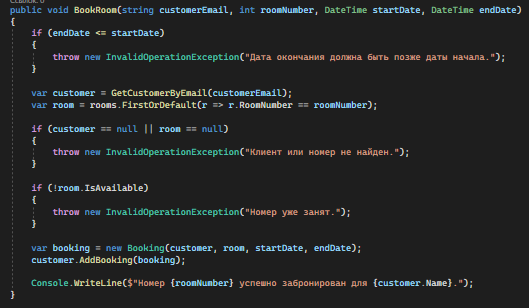
* 1. В класс Product в метод Update добавить проверку на нулевой или отрицательный запас на складе.



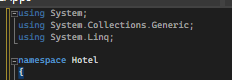
* 1. В класс Cart в метод AddToCart добавить обработку исключения, для проверки, есть ли товар на складе перед добавлением продуктов в корзину.



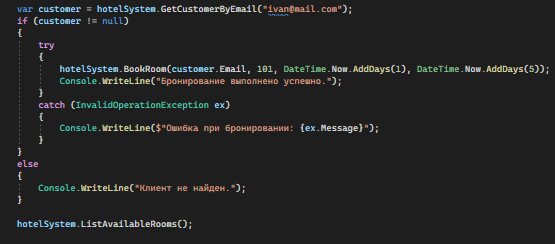
* 1. В класс Hotel в метод Booking добавить обработку исключения, для проверки, что дата окончания бронирования позже даты начала, и проверку на то, что номер не занят.



* 1. В класс Hotel добавить подключение библиотек.



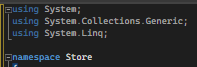
* 1. В класс Program в метод StartHotel перед вызовом BookRoom добавить проверку, доступен ли номер, чтобы избежать ошибки во время выполнения.



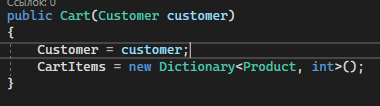
* 1. В класс Programв метод StartOnlineStore добавить проверку на наличие заказчика и наличие продукта перед добавлением его в корзину.



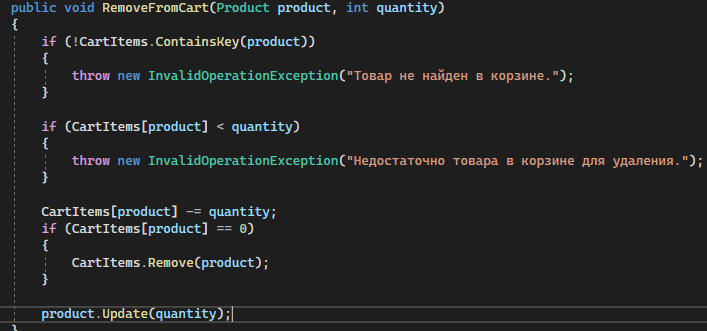
* 1. В класс Store добавить подключение библиотек.



* 1. В класс Cart добавить инициализацию Castomer.



* 1. В класс Store в метод RemoveFromCart добавить проверку на наличие товаров в корзине.



1. **Ответы на контрольные вопросы**
   1. Основные техники инспектирования кода:

- парное программирование,

- код-ревью,

- статический анализ кода,

- модульное тестирование,

- формальные инспекции.

* 1. Основные критерии при инспектировании кода:

- корректность,

- читаемость,

- производительность,

- безопасность,

- поддерживаемость,

- тестируемость.

* 1. Ошибки, которые можно предотвратить при помощи инспектирования кода:

- логические ошибки,

- ошибки синтаксиса,

- проблемы с производительностью,

- уязвимость безопасности,

- нарушения стандартов кодирования,

- ошибки в обработке исключений.

1. **Вывод**
   1. В ходе практической работы изучены методы инспектирования кода программы и приобретены навыки применения их на практике для повышения качества программного обеспечения.

# Практическая работа №5

## Описание результатов тестирования ПО.

1. **Цель работы**
   1. Изучить методы описания результатов тестирования и разработки ПО.
2. **Ход работы**
   1. Подготовили презентацию результатов разработки ПО по заданной теме.
   2. Подготовили выступление с презентацией ПО.
   3. Презентовали ПО преподавателю.

1. **Ответы на контрольные вопросы**
   1. Основные пункты, которые должна содержать презентация ПО:
2. введение: краткое описание темы презентации,
3. определение проблемы: главная проблема или потребность, которую решает ПО,
4. представление решения: функции и преимущества,
5. аргументация выгод и преимуществ,
6. призыв к действию: тестирование ПО, покупка или обратная связь.
   1. Ориентиры при подготовке презентации ПО:
   2. аудитория: нужно учитывать уровень знаний и интересы аудитории,
   3. структура и логика: нужно разбить материал на логические блоки с плавными переходами между ними,
   4. визуализация: нужно использовать графики, диаграммы и изображения для иллюстрации идей,
7. **Вывод**
   1. В ходе практической работы были изучены методы описания результатов тестирования и разработки ПО.