МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

Отчет   
по лабораторной работе №1  
на тему: «Знакомство с платформой .NET. Реализация консольных приложений»  
по дисциплине «Проектирование и архитектура программных систем»

Выполнили: Марочкин М.А. Шифр: 170584   
 Шорин В.Д. Шифр: 171406  
 Щекотихин С.Е. Шифр: 170590  
ИПАИТ  
Направление: 09.03.04 «Программная инженерия»  
Группа: 71-ПГ  
Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Отметка о зачете:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата «\_\_\_\_» \_\_\_\_\_\_ 2019 г.

Орел, 2019 г.

## Задание на лабораторную работу

Спроектировать модульную архитектуру и реализовать на языке программирования c# программу для решения следующих задач:

1. Вывести на экран аргументы, переданные в программу при запуске в командной строке.
2. Распечатать лет с 1900 по 2000. Рядом с каждым годом вывести слово «високосный» если этот год високосный или «не високосный» – если нет.
3. Вывести последовательность чисел Фибоначи до заданного числа.
4. Вычислить факториал заданного числа.
5. Вывести все простые числа, не превышающие заданное. Для решения использовать алгоритм «решето Эратосфена».

Разработанная программа должна иметь единый консольный интерфейс. Каждая задача реализуется в отдельном модуле.

**Описание структуры программы**

Программа реализует модульную архитектуру, где каждая задача реализуется в отдельном своем соответствующем модуле.

В модуле «Program.cs» находится точка входа в программу, из которой вызывается следующий модуль «Menu.cs».

В модуле «Menu.cs» содержится консольный интерфейс и вызов каждого из пяти модулей в зависимости от введенного пользователем значения.

В модулях «Task\_1.cs» - «Task\_5.cs» реализованы соответствующие задачи из задания на лабораторную работу с соответствующим им консольным интерфейсом.

## Контрольные вопросы

1. **Что понимается под термином архитектура программной системы?**

**Архитектура программного обеспечения** — совокупность важнейших решений об организации программной системы. Архитектура включает:

* выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;
* соединение выбранных элементов структуры и поведения во всё более крупные системы;
* архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение

1. **Перечислите основные разновидности архитектуры программных систем.**

* ***Многоуровневый шаблон (Layered pattern).*** Система разбивается на уровни, которые на диаграмме изображаются один над другим. Каждый уровень может вызывать только уровень на 1 ниже него. Таким образом разработку каждого уровня можно вести относительно независимо, что повышает модифицируемость системы. Недостатками данного подхода являются усложнение системы и снижение производительности.
* ***Шаблон посредника (Broker pattern).*** Когда в системе присутствует большое количество модулей, их прямое взаимодействие друг с другом становится слишком сложным. Для решения проблемы вводится посредник (например, шина данных), по которой модули общаются друг с другом. Таким образом, повышается функциональная совместимость модулей системы. Все недостатки вытекают из наличия посредника: он понижает производительность, его недоступность может сделать недоступной всю систему, он может стать объектом атак и узким местом системы.
  + - Шаблон ***«Модель-Представление-Контроллер»*** (Model-View-Controller pattern). Т.к. требования к интерфейсу меняются чаще всего, то возникает потребность часто его модифицировать, при этом сохраняя корректное взаимодействие с данными (чтение, сохранение). Для этого в шаблоне Model-View-Controller (MVC) интерфейс отделён от данных. Это позволяет менять интерфейсы, равно как и создавать их разные варианты. В MVC система разделена на:
    - *Модель*, хранящую данные
    - *Представление*, отображающее часть данных и взаимодействующее с пользователем
    - *Контроллер*, являющийся посредником между видами и моделью
* ***Клиент-серверный шаблон*** (Client-Server pattern). Если есть ограниченное число ресурсов, к которым требуется ограниченный правами доступ большого числа потребителей, то удобно реализовать клиент-серверную архитектуру. Такой подход повышает масштабируемость и доступность системы. Но при этом сервер может стать узким местом системы, при его недоступности становится недоступна вся система.

1. **Какие требования предъявляются к проектированию модульной архитектуры программной системы?**

* **Эффективность системы.** В первую очередь программа, конечно же, должна решать поставленные задачи и хорошо выполнять свои функции, причем в различных условиях. Сюда можно отнести такие характеристики, как надежность, безопасность, производительность, способность справляться с увеличением нагрузки (масштабируемость) и т.п.
* **Гибкость системы**. Любое приложение приходится менять со временем — изменяются требования, добавляются новые. Чем быстрее и удобнее можно внести изменения в существующий функционал, чем меньше проблем и ошибок это вызовет — тем гибче и конкурентоспособнее система.
* **Расширяемость системы.** Возможность добавлять в систему новые сущности и функции, не нарушая ее основной структуры

1. **Перечислите основные особенности языка программирования C# и платформы .NET.**

C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java.

Основные черты платформы .Net:

* **Поддержка нескольких языков**. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) - своего рода ассемблер платформы .NET. Поэтому мы можем сделать отдельные модули одного приложения на отдельных языках.
* **Кроссплатформенность.** .NET является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент .NET Framework поддерживается на большинстве современных ОС Windows (Windows 10/8.1/8/7/Vista). А благодаря проекту Mono можно создавать приложения, которые будут работать и на других ОС семейства Linux, в том числе на мобильных платформах Android и iOS.
* **Мощная библиотека классов.** .NET представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# - текстовый редактор, чат или сложный веб-сайт - так или иначе мы задействуем библиотеку классов .NET.
* **Разнообразие технологий.** Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF. Для создания веб-сайтов - ASP.NET и т.д.

Также еще следует отметить такую особенность языка C# и фреймворка .NET, как автоматическая сборка мусора. А это значит, что нам в большинстве случаев не придется, в отличие от С++, заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.

1. **Как происходит компиляция и выполнение программы на языке C#?**

* **Управляемый и неуправляемый код.** Нередко приложение, созданное на C#, называют управляемым кодом (managed code). Что это значит? А это значит, что данное приложение создано на основе платформы .NET и поэтому управляется общеязыковой средой CLR, которая загружает приложение и при необходимости очищает память. Но есть также приложения, например, созданные на языке С++, которые компилируются не в общий язык CIL, как C# или VB.NET, а в обычный машинный код. В этом случае .NET не управляет приложением.

В то же время платформа .NET предоставляет возможности для взаимодействия с неуправляемым кодом. Мы наряду со стандартными классами библиотеки .NET можем также использовать сборки COM.

* **JIT-компиляция.** Как выше писалось, код на C# компилируется в приложения или сборки с расширениями exe или dll на языке CIL. Далее при запуске на выполнение подобного приложения происходит JIT-компиляция (Just-In-Time) в машинный код, который затем выполняется. При этом, поскольку наше приложение может быть большим и содержать кучу инструкций, в текущий момент времени будет компилироваться лишь та часть приложения, к которой непосредственно идет обращение. Если мы обратимся к другой части кода, то она будет скомпилирована из CIL в машинный код. При том уже скомпилированная часть приложения сохраняется до завершения работы программы. В итоге это повышает производительность.

**Листинг программы**

**Program.cs**

namespace Lab1{

internal class Program {

public static void Main(string[] args) { Menu.PrintMenu(args); }

}

}

**Menu.cs**

using System;

namespace Lab1{

public static class Menu {

public static void PrintMenu(String[] args) {

while (true) {

Console.WriteLine("1 - Task 1"); Console.WriteLine("2 - Task 2");

Console.WriteLine("3 - Task 3"); Console.WriteLine("4 - Task 4");

Console.WriteLine("5 - Task 5"); Console.WriteLine("0 - Exit");

Console.Write("Write command: "); int command = Convert.ToInt32(Console.ReadLine());

switch (command) {

case 1: {

Task\_1.PrintArgs(args); Console.ReadKey(); Console.Clear(); break;

}

case 2: {

Task\_2.IsYears(); Console.ReadKey(); Console.Clear(); break;

}

case 3: {

Task\_3.PrintFibDigits(); Console.ReadKey(); Console.Clear(); break;

}

case 4: {

Task\_4.PrintFactorial(); Console.ReadKey(); Console.Clear(); break;

}

case 5: {

Task\_5.PrintSimpleNumbers(); Console.ReadKey(); Console.Clear(); break;

}

case 0: {

Environment.Exit(0); break;

}

default: {

Console.WriteLine("Write correct command"); break;

}

}

}

}

}

}

**Task\_1.cs**

using System;

namespace Lab1{

public static class Task\_1 {

public static void PrintArgs(String[] args) {

if (args.Length == 0) { Console.WriteLine("No args"); return; }

foreach (string var in args) { Console.WriteLine(var); }

}

}

}

**Task\_2.cs**

using System;

namespace Lab1{

public static class Task\_2 {

public static void IsYears() {

for (int year = 1900; year <= 2000; year++) {

Console.Write(year + " ");

if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0) { Console.WriteLine("YES"); }

else { Console.WriteLine("NO"); }

}

}

}

}

**Task\_3.cs**

using System;

namespace Lab1{

public static class Task\_3 {

public static void PrintFibDigits() {

int fib1 = 0; int fib2 = 1; int res = 0;

Console.Write("Write max fibonacci: ");

int maxFib = Convert.ToInt32(Console.ReadLine());

Console.WriteLine(fib1 + "\n" + fib2);

while (res <= maxFib) {

res = fib2 + fib1;

if (res > maxFib) return;

Console.WriteLine(res);

fib1 = fib2;

fib2 = res;

}

}

}

}

**Task\_4.cs**

using System;

namespace Lab1{

public static class Task\_4 {

public static void PrintFactorial() {

Console.Write("Write factorial number: ");

int maxFact = Convert.ToInt32(Console.ReadLine());

int fact = 1;

for (int var = 1; var <= maxFact; var++) { fact \*= var; }

Console.WriteLine(fact);

}

}

}

**Task\_5.cs**

using System;

namespace Lab1 {

public static class Task\_5 {

public static void PrintSimpleNumbers() {

Console.Write("Write max simple number: ");

int maxNumber = Convert.ToInt32(Console.ReadLine());

int[] numbers = new int[maxNumber + 1];

for (int i = 0; i < maxNumber + 1; i++) { numbers[i] = i; }

int x = 2;

for (int i = 2; i < maxNumber + 1; i++) {

if (numbers[i] != 0) {

Console.WriteLine(numbers[i]);

for (int j = i \* i; j < maxNumber + 1; j += i) { numbers[j] = 0; }

}

}

}

}

}