

**ПРОФЕСИОНАЛНА ГИМНАЗИЯ**

**ПО ЕЛЕКТРОТЕХНИКА И ЕЛЕКТРОНИКА “АПОСТОЛ АРНАУДОВ”**

гр. Русе, ул. “Потсдам” № 3; п.к. 7005, тел. 082/84-60-96; e-mail: [info-1806301@edu.mon.bg](mailto:info-1806301@edu.mon.bg)

Д И П Л О М Е Н П Р О Е К Т

**ЗА ПРИДОБИВАНЕ НА**

**ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ**

**по професия код 481020 „Системен програмист“**

**специалност код 4810201 „Системно програмиране“**

**ТЕМА:**

**SOLID принципите при разработка на софтуер в c#**

**Ученик**: Юлиян Велиславов Наумов

**Ръководител-консултант:** Милена Дамесова-Христова

гр. Русе

2024

**ГЛАВА ПЪРВА**

**Основи на езика C# и ООП концепцията**

1. **Основни възможности на програмния език C#**

C# (C Sharp, произнася се Си Шарп) е обектно ориентиран език за програмиране, разработен от Microsoft като част от софтуерната платформа .NET. Стремежът още при създаването на C# езика е бил да се създаде прост, модерен, обектно ориентиран език с общо предназначение. Основа за C# са C++, Java и донякъде езици като Delphi, VB.NET и C. Проектиран е да балансира мощност (C++) с възможност за бързо разработване (Visual Basic и Java). Те представляват съвкупност от дефиниции на класове, които съдържат в себе си методи, а в методите е разположена програмната логика – инструкциите, които компютърът изпълнява. Програмите на C# представляват един или няколко файла с разширение .cs, в които се съдържат дефиниции на класове и други типове. Тези файлове се компилират от компилатора на C# до изпълним код и в резултат се получават асемблита – файлове със същото име, но с различно разширение (.exe или .dll).

Той е известен със своята синтаксисна яснота и гъвкавост, което го прави предпочитан избор сред програмистите по целия свят.

Сред основните възможности на C# са:

* **Обектно-ориентиран подход:** C# поддържа основните концепции на обектно-ориентираното програмиране, като класове, обекти, наследяване, полиморфизъм и инкапсулация. Това позволява по-лесно управление на сложни проекти и по-ефективно използване на кода.
* Мултиплатформеност: C# е разработен да бъде използван на различни операционни системи и платформи, включително Windows, Linux и macOS. Това осигурява гъвкавост и възможност за създаване на приложения, които могат да работят на различни устройства.
* Интеграция с .NET Framework и .NET Core: C# е основен език за програмиране в рамките на .NET Framework и .NET Core, които предоставят обширен набор от библиотеки и инструменти за разработка на разнообразни приложения. Това включва уеб приложения, десктоп приложения, мобилни приложения и много други.
* Съвременни функции на програмния език: C# постоянно се развива и включва съвременни функции и концепции на програмиране, като асинхронно програмиране, LINQ (Language Integrated Query), патерн съпоставяне и др. Тези функции улесняват разработката на по-ефективен и чист код.

Богата екосистема от инструменти и ресурси: Заедно с C# и .NET идва богата екосистема от инструменти за разработка, IDE (Integrated Development Environment) като Visual Studio, както и голямо количество отворени библиотеки и ресурси за подпомагане на разработката.

A computer chip with many squares and wires

Description automatically generated with medium confidence

Фигура 1 / Nuget.org /

Тези основни възможности на C# правят езика предпочитан избор за множество програмисти при създаването на разнообразни приложения.

* 1. **Типове данни и колекции**

**1.1.1. Прости типове**

Простите типове, който притежава C#, имат някои общи характеристики. Първо, всички те са псевдоними на .NET системните типове. Второ, изразите с константи от прост тип се изчисляват само при компилирането, не и при стартирането. На края, простите типове могат да бъдат инициализирани с литерали.

Простите типове в C# са групирани по следният начин:

 Целочислени типове

 Тип bool

 Тип char (специален случай на целичислен тип)

 Типове с плаваща запетая

 Типът decimal

* + 1. **Целочислени типове**

Съществуват девет целочислени типа в C#: sbyte, byte, short, ushort, int, uint, long, ulong и char (обсъден в отделна секция). Те имат следните характеристики:

 Типът sbyte представлява 8 битови цели числа със знак, чиито стойности са между -128 и 127.

 Типът byte представлява 8 битови цели числа без знак, чиито стойности са между 0 и 255.

 Типът short представлява 16 битови цели числа със знак, чиито стойности са между -32,768 и 32,767.

 Типът ushort представлява 16 битови цели числа без знак, чиито стойности са между 0 и 65,535.

 Типът int представлява 32 битови цели числа със знак, чиито стойности са между -2,147,483,648 и 2,147,483,647.

 Типът uint представлява 32 битови цели числа без знак, чиито стойности са между 0 и 4,294,967,295.

 Типът long представлява 64 битови цели числа със знак, чиито стойности са между-9,223,372,036,854,775,808 и 9,233,372,036,854,755,807.

 Типът ulong представлява 64 битови цели числа без знак, чиито стойности са между 0 и 18,446,744,073,709,551,615

Може би програмистите на C и VB ще бъдат малко изненадани от новите обхвати на типовете int и long. За разлика от други програмни езици, в C# типът int вече не е зависим от размера на думата за съответната система, а типът long е установен на 64 бита.

**1.1.1.1 Типът bool**

Типът bool представлява булевите променливи true и false. На булева променлива можете да присвоите една от двете стойности (true или false), или път да присвоите израз, чийто резултат се свежда отново до една от тях:

bool bTest = (100 > 90);

За разлика от C и C++, в C# стойността true вече не се представя от която и да била нулева стойност. Не съществува възможност за конвертиране на целочислени типове в булев такъв, за да се нарушава тази конвенция.

**1.1.1.2 Типът char**

Типът char представлява един Unicode символ. Unicode символът е със 16 бита дължина и може да бъде използван за представянето на повече езици по света. Може да се присвои символ към променливата от тип char по следният начин:

char chSomeChar = 'A';

В допълнение към това, може да се присвои стойност на символна променлива посредством шестнадесетична стойност с префикс x или Unicode формат с префикс u:

char chSomeChar = 'x0065';

char chSomeChar = 'u0065';

Не съществува безусловно конвертиране от тип char в някакъв друг тип данни. Това означава, че не може да се третира променлива от този тип като някакъв друг целочислен тип в C#. Това е още една подробност, с която програмистите на C трябва да свикнат. Въпреки това, може да се извърши изрично преобразуване:

char chSomeChar = (char)65;

**1.1.1.3 Типове с плаваща запетая**

Два типа за данни спадат към категорията типове с плаваща запетая: float и double. Разликата между тях е в обхвата от сойности и броя цифри след десетичната запетая:

float: Обхватът от стойности е от 1.5 по 10 на -45 степен до 3.4 по 10 на 38 степен, с точност от 7 знака след десетичната точка.

double: Обхватът от стойности е от 5.0 по 10 на -324 степен до 1.7 по 10 на 308 степен с точност от 15-16 знака след десетичната точка.

При извършването на изчисления с които и да било от двата типа с плаваща запетая, могат да се получат следните стойности:

 Положителна или отрицателна нула

 Положителна или отрицателна безкрайност

 Не числена стойност (Not-a-Number - NaN)

 Крайно множество от не нулеви стойности

Друго правило при изчисленията на стойностите е, че ако една променлива в даден израз е от тип плаваща запетая, всички останали типове се конвертират в тип с плаваща запетая преди извършването на изчисленията.

**1.1.1.3 Типът double**

Типът decimal е много прецизен, 128 битов тип за данни, който е предназначен за употреба при финансови и парични изчисления. Може да се представят стойности, вариращи приблизително от 1.0 по 10 на -28 степен до 7.9 по 10 на 28 степен с 28 до 29 значими цифри. Важно е да се отбележи, че точността се определя от цифри, а не от десетични знаци. Операциите са точни, максимум до 28 десетични знака.

Както може да се забележи, обхватът от стойности на типа char е по-малък от този на типа double, но с по-голяма прецизност. Поради тази причина безусловното конвертиране между типовете decimal и double не е възможно - в едната посока може да възникне препълване, а в другата може да се загуби точността. Затова е необходимо изрично да се заяви конвертирането чрез използване на образец.

Навсякъде да е в неопределено лице: Не да направите, а да се направи...

Когато се дефинира променлива и се присвоява стойност на нея, се използва суфикса m, за да се обозначи, че стойността е от тип decimal.

decimal decMyValue = 1.0m;

**1.1.1.4 Типът int**

Типът данни int представлява цели числа в C#. Можете да присвоите стойности на променлива от тип int, като използвате цели числа без десетична запетая. При дефиниране на променлива от тип int, можете да използвате оператора за присвояване (=), за да й зададете стойност.

int age = 25;

int numberOfStudents = 50;

Целочислените типове данни в C# могат да представят както положителни, така и отрицателни числа, в рамките на определен обхват. В зависимост от конкретната имлементация, обхватът на типа int обикновено е от -2,147,483,648 до 2,147,483,647.

При използването на целочислени операции, трябва да се внимава за възможността за препълване или загуба на точност при извършването на операции, особено при деление или умножение на големи числа.

* 1. **Оператори**

A screenshot of a computer

Description automatically generated

Фигура 2 / Оператори /

**1.3 Управляващи инструкции**

**1.3.1 if-else case**

Условните конструкции if и if-else предоставят условен тип контрол, чрез който програмата може да изпълнява различни функции в зависимост от някакво условие, което се проверява по време на изпълнение на конструкцията.

public static void IfElse

{

if (булев израз)

{

тяло тип (1) на условната конструкция;

}

else if (нов булев израз)

{

тяло тип (2) на условната конструкция;

}

else

{

тяло тип (3) на else-конструкция;

}

}

Форматът включва: if клауза, булев израз и тяло на условната конструкция, if-else конструкцията се състои от: запазена дума if, булев израз, тяло на условната конструкция, запазена дума else, тяло на else конструкция. Тялото на else конструкцията може да се състои от един или няколко оператора, заградени в къдрави скоби, също както тялото на условната конструкция.

**1.3.2 switch case**

Конструкцията switch-case избира измежду части от програмен код на базата на стойност на зададения селектор, който представлява променлива или израз (най-често целочислен). Форматът на конструкцията за избор на вариант е следният:

public static void SwitchCase

{

switch (селектор)

{

case стойност на селектора (1): конструкция; break;

case стойност на селектора (2): конструкция; break;

default: конструкция; break;

}

}

Операторът switch сравнява резултата от селектора с всяка една стойност от изброените в тялото на switch конструкцията в case етикетите. Ако се открие съвпадение с някой case етикет, се изпълнява съответната конструкция. Ако не се открие съвпадение, се изпълнява default конструкцията.

**1.3.3 while loop**

While цикълът в C# е основна конструкция за повтаряне на определен блок код. Използва се, за да повтаря определени действия, докато е изпълнено определено условие. Когато програмата достигне while цикъла, тя първо проверява условието. Ако това условие е верно, блокът от код в рамките на while цикъла се изпълнява. След като блокът приключи изпълнението си, програмата отново проверява условието. Ако то остане верно, цикълът се изпълнява отново. Този процес продължава, докато условието стане невярно, като по този начин while цикълът завършва.

public static void while

{

// Докато условието е вярно цикъла ще се изпълнява.

while (условие)

{

}

}

**1.3.4 for loop**

За разлика от while цикъла, for цикълът в C# изисква да бъде предварително определен брой итерации. Когато програмата достигне for цикъла, тя изпълнява инициализацията, след това проверява условието. След изпълнението на блока от код, програмата изпълнява стъпката за промяна и отново проверява условието. Този процес продължава, докато условието стане невярно, като по този начин завършва for цикълът.

public static void for

{

//Инициализация, следвана от проверка на условието, изпълнение на цикъла.

for (i=0; i<10; i++)

{

}

}

**1.3.5 foreach loop**

Foreach цикълът в C# се използва за итериране през елементите на колекция (като масив, списък и др.) без необходимост от указване на начало, край и стъпка за промяна. Вместо това, foreach цикълът автоматично преминава през всеки елемент от колекцията и изпълнява определен блок код за всяка итерация. Този вид цикъл е особено удобен, когато искаме да извършим действие върху всеки елемент на колекцията без да се налага да се грижим за индекси или броячи.

public static void foreach

{

//Изпълнение на цикъла за всеки елемент от масив или списък.

foreach (елемент in колекция)

{

}

}

**Изчисти си белите полета навсякъде!!!**

1. **Етапи при разработката на софтуер**

Писането на софтуер може да бъде сложна задача, която отнема много време на цял екип от софтуерни инженери и други специалисти. Затова с времето са се обособили различни методики и практики, които улесняват работата на програмистите. Общото между всички тях е, че разработката на всеки софтуерен продукт преминава през няколко етапа, а именно:

* 1. **Етапи при разработката на софтуер**

От решаващо значение е групата софтуерни експерти да събере бизнес нуждите в тази първоначална стъпка, преди да могат да измислят широк дизайн за която и да е програма. Основната цел на потребителите и ръководителите на проекти в този момент е да съставят списък на специфичните функции, изисквани от всяка програма, която се разглежда. На този етап трябва да бъдат зададени няколко въпроса, включително:

Кой се очаква да използва тази програма?; Когато програмата приключи, как ще се използва?; Какъв вид информация трябва да бъде включена в софтуера?; Какви данни трябва да генерира тази програма?

След отговорите на тези фундаментални въпроси се изготвя общ план, върху който софтуерните инженери да се концентрират. След това се проверява истинността на данните, както и възможността за включването им. Накрая се създава документ за спецификация на изискванията, който ще служи като ръководство за следващия етап от процеса на разработка на софтуер.

* 1. **Дизайн**

Разработването на софтуер сега преминава към следващия етап. Според изискванията от етап 1, на този етап се създава проект на програмата. Системните проекти помагат при спецификацията както на хардуера, така и на системните изисквания. Също така помага при дефинирането на цялостната система на софтуерния архитект.

Изискванията за проектиране на системата се използват като вход в следващата стъпка от методологията. Тестовите стратегии се разработват от тестери по време на тази фаза, която включва списък с елементи за тестване и как да ги проверите.

* 1. **Кодиране**

След получаване на проектната документация за изгражданата програма, работата се разделя равномерно на множество части и модули. Тук започва истинският код. Производството на отлични кодове от програмистите е основният акцент на тази фаза. Това е най-отнемащата време стъпка от целия процес.

* 1. **Тестване**

Този етап е наистина важен за разработчиците. Ако нещо се обърка по време на стъпката на тестване или ако бъдат открити грешки в кодовете, процесът на кодиране ще трябва да се повтори и цикълът ще продължи, докато проектът приключи. Всички видове функционални тестове, включително фаза на тестване, тестване на единици, автоматизация на тестове, валидиране на входа и нефункционално тестване, са завършени на този етап.

* 1. **Внедряване**

Етапът на внедряване започва, след като всички грешки в кода бъдат елиминирани по време на етапа на тестване. Попълненият код впоследствие се интегрира в програмата и се разпространява или предоставя на потребителите за използване.

Тъй като решението се разпространява сред бъдещи клиенти, първата стъпка е да се проведе бета тестване, за да се потвърди, че функционира правилно в широк мащаб. Ако някакви корекции са осъществими или ако по време на внедряването бъдат открити някакви неизправности, те незабавно се уведомяват до персонала, който гласят грешките, за да функционира правилно в реално време. Окончателното разпръскване или внедряване започва след като всички модификации са били приложени и всички проблеми са били адресирани.

* 1. **Поддръжка**

Когато потребителите започнат да използват добре разработен софтуер, с времето започват да се появяват истински трудности. Това не означава, че програмата ще бъде подправена. Въпреки това може да се наложи притесненията да се появяват отново от време на време. Тази процедура е известна като поддръжка на завършен продукт или програма.

* 1. **Поддръжка**

Парите и усилията, изразходвани за проектиране на софтуер, варират в зависимост от нуждите. В такива случаи е за предпочитане да получите помощ от реномирана фирма за разработка на софтуер, която знае какво прави. Това намалява количеството време, енергия и пари, изразходвани за проекта. Наемането на неопитен разработчик за завършване на проект може да доведе до непредвидени проблеми дори след като проектът е завършен. Това не означава, че неопитните кодери не са способни. Ако имате ограничен бюджет и не бихте искали да рискувате обаче, придържането към опитни разработчици е правилният начин.

**ГЛАВА ВТОРА**

**ООП парадигмата и SOLID принципите при разработка на софтуер.**

* 1. **SOLID Принципи**
     1. **Single Responsibility Principle**

Single Responsibility Principle (SRP) / Принцип за единствена отговорност – Един клас трябва да има една единствена причина да бъде променен. Този принцип може да определи дали един клас изпълнява твърде много задачи. Класът трябва да прави едно нещо и само едно нещо, и да го прави добре. И ако един клас прави едно нещо, много по-лесно е кодът да се разбере, когато се чете от човек. Вероятността този код в бъдеще да се промени е минимална, но дори и това да се наложи, най-вероятно няма да възникнат непредвидени странични ефекти от тази промяна

* + 1. **Open-Closed Principle**

Open-Closed Principle (OCP) / Принцип отворен/затворен – Един клас трябва да бъде отворен за разширение, но затворен за модификации. Този принцип се отнася до добавянето на нова функционалност в една програма без да се променя вече написан код или дори асембли. Причината за това е, че всеки път, когато се променя даден код, се появява и риска от възникването на бъгове в съществуваща функционалност.

* + 1. **Liskov Substitution Principle**

Liskov Substitution Principle (LSP) / Принцип на заместване на Лисков – Всеки един клас трябва да може да бъде заменен с подклас без извикващият го код да знае за промяната. Всеки наследник (подтип) трябва лесно да заменя всичките си базови типове. Подтипът не трябва да премахва нито една от функционалностите на базовия клас, а при нужда само да ги екстендва, ако по някаква причина даден клас не поддържа някоя от функционалностите на своя базов клас (родител), трябва да се помисли, дали се използва правилен вид наследяване.

* + 1. **Interface Segregation Principle**

Interface Segregation Principle (ISP) / Принцип за разделяне на интерфейсите – Повече специфични интерфейси са по-добре, отколкото един общ интерфейс. Този принцип гласи, че всеки интерфейс трябва да бъде разбит на много на брой малки интерфейси. Тези интерфейси трябва да отговарят за едно-единствено нещо. Нито един клас не трябва да бъде принуждаван да имплементира методи, които няма да ползва никога.

„Дебел“, голям, пълен с различни методи интерфейс води до:

* Класовете да имплементират методи, които не са им нужни.
* Увеличена свързаност(coupling) между класовете.
* Намалена гъвкавост.
* Поддръжката става по-трудна.
  + 1. **Dependency Inversion Principle**

Dependency Inversion Principle (DIP) / Принцип на обръщане на зависимостите – Кодът (класът) трябва да зависи от абстракции, а не от конкретни имплементации. Принципът на обръщане на зависимостите е специфичен начин за отвързване (отделяне; decoupling) на софтуерните модули. Когато следваме този принцип, модулите на по-високо ниво не зависят от тези на по-ниско ниво, като и двата трябва да зависят само и единствено от абстракции. В същото време абстракциите не трябва да зависят от детайлите, а детайлите трябва да зависят от абстракциите. Този принцип е въведен от Роберт Мартин. Зависимост наричаме даден клас или модул, от който зависи друг клас или модул.

**ГЛАВА ТРЕТА**

**ПРИЛОЖЕНИЕ ЗА УПРАВЛЕНИЕ НА ХОТЕЛ**