**|?| Опишете накратко платформата Microsoft .NET.**

**Кои са основните принципи, които залягат в нея?**

**Избройте четирите компонента, от които тя се състои.**

Съвкупност от технологии, които свързват хората с информацията -

навсякъде, по всяко време, от всяко устройство.

Четирите компонента са: .NET Framework (VS .NET), .NET Smart Clients,

.NET Building Block Services, .NET Enterprise Servers

**|?| Какво представляват .NET Enterprise сървърите? Избройте някои от** **тях.**

Сървърна инфраструктура за .NET платформата

Среда за изпълнение, управление и интеграция на XML Web услуги

Microsoft SQL Server

Microsoft BizTalk Server

Microsoft Host Integration Server

Microsoft Exchange

Microsoft SharePoint Portal Server

Microsoft Application Center

Microsoft Content Management Server

Microsoft Mobile Information Server

Microsoft Internet Security and Acceleration Server

Microsoft Commerce Server

**|?| Какво представлява .NET Framework? От какви компоненти се състои?**

Среда за разработка и изпълнение на приложения за .NET платформата.

Програмен модел, библиотеки от класове и единна инфраструктура за разработка, независима от езиците за програмиране

*Common Language Runtime (CLR)* - Среда за контролирано изпълнение на програмен код

*Framework Class Library (FCL*) - Стандартна библиотека от класове за разработка на .NET приложения

**|?| Какво е Visual Studio .NET?**

Интегрирана среда за .NET разработка

**|?| За какво служат .NET Building Block услугите?**

- Автентикация – .NET Passport (single sign-on)

- Доставка на съобщения

- Съхранение на лични потребителски данни – документи, контакти, електронна поща, календар, любими сайтове и т.н.

- Съхранение на настройките на приложенията

**Какво са .NET Smart клиентите? Какво е характерно за тях?**

Представляват smart клиенти (архитектурна концепция за изграждане на

клиентски приложения), базирани на .NET Framework

Работят върху различни smart устройства(компютри, мобилни телефони, handhelds, вградени устройства, ...)

Осигуряват достъп до информацията на потребителя – навсякъде и по всяко време

**|?| Какво представлява средата за контролирано изпълнение на програмен код Common Language Runtime (CLR)?**

Среда за контролирано изпълнение на управляван .NET програмен код

**|?| Какво е Framework Class Library (FCL)? Каква функционалност предлага тя?**

Стандартна библиотека от класове за разработка на .NET приложения.

Предоставя основната функционалност за разработка: ADO.NET, XML, ASP.NET, Web Services, Windows Forms

**|?| Какво е управляван код? Има ли причина да бъде използван вместо традиционния native код?**

Кодът, който CLR изпълнява. Представлява програмен код на езика от ниско ниво IL (Intermediate Language)

Защитен от неправилна работа с типове (type-safety), сигурен, надежден, Позволява интеграция между различни езици за програмиране и преносимост между различни платформи.

**Какво е характерно за езика MSIL?**

-Позволява ефективно компилиране до машинен код за различни платформи

-Винаги се компилира преди изпълнение

**|?| Какво представляват .NET асемблитата (assemblies)?**

Програмни компоненти – DLL и EXE файлове

Каква информация съдържат са метаданните в асемблитата?

**Какво са .NET приложенията? Какво е application domain?**

Конфигурируеми изпълними единици. Състоят се от едно или няколко асемблита.

Application Domain - .NET процеси в CLR

**|?| Какво е Common Language Specification (CLS)? Защо е необходима тя?**

Съвкупност от правила, които всички .NET езици са задължени да спазват.

Специфицира минималните изисквания на всички .NET езици и осигурява съвместимост между тях.

**|?| Какво представлява общата система от типове в .NET Framework** **(Common Type System)?**

Дефинира поддържаните от CLR типове данни и операции над тях.

Осигурява съвместимост на данните между различните .NET езици.

**|?| Избройте няколко от .NET езиците. Какво е общото между тях?**

C#, C++, J#, VB.NET, Jscript

Много езици, но с единна система от типове, програмен модел и библиотека от класове

**|?| Избройте основните пакети от Framework Class Library (FCL).**

System.Web

System.Windows.Forms

System.Drawing

System.Data

System.Xml

System

**|?| Формулирайте основните принципи на обектно-ориентираното програмиране.**

**Дефинирайте понятията клас, обект, атрибут, метод, капсулация на данните,**

**абстракция на данните и действията, наследяване, полиморфизъм.**

*Клас* – категория обекти с общи свойства и операции, които могат да се извършват върху тях (например всички сметки в дадена банка)

*Обект* - единичен обект от даден клас, инстанция на клас

*Метод* - операция, която всички обекти от даден клас могат да извършват

*Капсулация на данните* - събиране на данните за даден обект и операциите над тях в едно цяло (клас), като се ограничава директния достъп до тези данни и операции

*Абстракция на данните* - възможността да работим с данни без да се интересуваме от тяхното вътрешно представяне, а само от операциите над тях

*Абстракция на действията* - възможността да изпълняваме действия, за които не знаем точно как са реализирани

*Наследяване* - възможността един клас (наследник) да придобие свойства и действия от друг клас (родител)

*Полиморфизъм* - възможността да разглеждаме обекти от клас-наследник като обекти от базов клас, като класът наследник може да е предефинирал някои от действията на базовия клас

**|?| Избройте основните разлики между стойностните и референтните типове.**

**Кои от следните типове са стойностни и кои референтни?**

**int, char, string, float, изброени типове, класове, структури,**

**интерфейси, делегати, масиви, указатели, опаковани стойностни типове**

При присвояване на *стойностни* типове се копира самата им стойност, а при *референтни* типове – само референцията (адреса)

При създаване на променлива от *стойностен* тип тя се заделя в стека, а при *референтните* типове – в динамичната памет

*Стойностните* типове наследяват типа System.ValueType, а *референтните* наследяват директно System.Object

Променливи от *стойностен* тип могат да се съхраняват в променливи от *референтен* тип чрез т.нар. опаковане (boxing)

*стойностни типове* - int, char, string, float .

*референтни типове* - класове, структури, интерфейси, делегати, масиви, указатели

**|?| Обяснете какво представляват делегатите в .NET Framework.**

Делегатите представляват .NET типове, които описват сигнатурата на даден метод (броя, типа и последователността на параметрите му) и връщания от него тип.

Делегатите в .NET Framework са специални класове и могат да бъдат два вида:

*Единични (single-cast) делегати* - Съдържат референция към един единствен метод

*Множествени (multicast) делегати* - Съдържат свързан списък от референции към методи

**|?| Обяснете какво представляват събитията (events) в .NET Framework.**

В компонентно-ориентираното програмиране компонентите изпращат събития (events) към своя притежател за да го уведомят при настъпване на интересна за него ситуация. В компонентния модел на .NET Framework абонирането, изпращането и получаването на събитията се поддържа чрез делегати и събития

**|?| Какво се препоръчва от утвърдената конвенция за събитията в .NET Framework?**

Делегатите, които се използват за събития да:

имат имена образувани от глагол + EventHandler (SomeVerbEventHandler)

връщат void и приемат два параметъра – обект-изпращач от тип System.Object и обект, съдържащ данни за събитието от тип, наследник на System.EventArgs

**|?| Обяснете какво представляват атрибутите в .NET Framework.**

Атрибутите представляват специални декларативни тагове, които служат за прикрепяне на описателна информация (анотации) към декларациите в кода

При компилация се записват в метаданните на асемблито

**Как се прилагат атрибути? Как се прилагат атрибути с параметри?**

За да се приложи атрибут, името му се огражда в квадратни скоби и се поставя преди декларацията, за която се отнася

**Как се задава цел при прилагане на атрибут?**

Чрез [AttributeUsage]

**|?| Опишете накратко стандарта Unicode. Каква е основната му идея?**

Стандартът Unicode съпоставя уникален номер на всеки знак (символ) от азбуките и йероглифните системи на езиците по цял свят.

В .NET Framework се използва кодиращата схема UTF-16, която представя един знак като една или две 16-битови стойности

**Има ли връзка между Unicode и типа System.Char в .NET Framework?**

**???**

**|?| Опишете какво представляват символните низове в .NET Framework.**

Символните низове представляват последователности от Unicode знаци в кодиране UTF-16

**Какво е характерно за класа System.String?**

Съдържа неизменима последователност от символи (System.Char)

Използва Unicode за да поддържа много езици едновременно

Съхранява низовете в динамичната памет (защото System.String е референтен тип)

**|?| Избройте по-важните методи и свойства на класа System.String и обяснете за какво служат.**

*Length* – връща дължината на низа

*Equals(string)* – сравнява низа с друг низ

*Compare(string, string)* – сравнява два низа лексикографски един с друг (прави се разлика между малки и главни букви)

*оператори == и !=* – също сравняват низове

*оператори + и +=* – слепват низове

*this[int]* – индексатор, който връща символа на зададената позиция (броенето започва от 0)

*StartsWith(string)* – проверява дали низът започва с посочения низ

*EndsWith(string)* – проверява дали низът завършва с посочения низ

*Substring(int startIndex, int Length)* – извлича подниз по дадено начало и дължина

*IndexOf(string)* – връща позицията на първото срещане на посочения низ или -1.

*LastIndexOf(string str)* – връща позицията на последното срещане на посочения низ

*Split(params char[])* – разделя низа на множество от низове спрямо зададена група разделящи символи

*Join(string, string[])* – слепва поредица от низове посредством зададен разделител

*Format(string, params object[])* – връща низ, получен чрез замяна на форматиращите идентификатори с посочените за тях стойности (като Console.WriteLine(…))

*PadLeft(int, char)* – подравнява отляво до посочената дължина със зададения символ

*PadRight(int, char)* – подравнява отдясно до посочената дължина със зададения символ

*Trim(params char[])* – изтрива всички зададени символи от началото и края на низа

*ToLower()* – заменя всички главни букви със съответните им малки, обратно на ToUpper()

*Insert(int index, string)* – връща нов низ, получен чрез вмъкване на посочения низ преди зададената позиция

*Remove(int index, int count)* – връща нов низ, получен чрез премахване на зададена поредица от символи

*Replace(string oldVal, string newVal)* – връща низ, получен чрез замяна на всички срещания на даден низ с даден друг низ

--Lecture 9--

**|?| Опишете накратко какво представляват регулярните изрази.**

Регулярен израз е символен низ, който описва някаква съвкупност от символни низове (регулярен език).

**Кои са основните елементи на езика на регулярните изрази? Какви метасимволи познавате?**

\t – табулация

\r – символ за връщане на каретката CR (0x0D)

\n – символ за нов ред LF (0x0А)

\xXX – символ с ASCII код XX (шестнайсетично)

\uXXXX – Unicode символ с номер XXXX (шестнайсетично)

\\ – символ \

\\* – символ \*

\+ – символ +

**|?| Опишете накратко средствата на .NET Framework за работа с регулярни изрази - основните класове и по-важните им методи.**

*Класът* ***Regex*** съдържа неизменим регулярен израз и методи за търсене, заместване и разделяне на низове чрез този израз.

*С* ***Regex*** *може да се работи по два начина:*

1. Инстанцира се класът и в конструктора му се подава регулярен израз, след което му се извикват методите за обработка на текст (IsMatch, Match, Matches, Replace, Split)

2. Използват се статичните методи на класа (IsMatch, Match, Matches, Replace, Split), на които се подава текста за обработка и регулярен израз, който да се използва

*По-важни методи и свойства на* ***Regex****:*

*IsMatch(text, pattern)* – проверява дали в даден текст се среща поне един подниз, който съответства на даден регулярен израз

*Match(text, pattern)* – търси зададения регулярен израз в зададения текст и връща първото съвпадение като Match обект

*Matches(text, pattern)* – търси зададения регулярен израз в зададения текст и връща MatchCollection от всички съвпадения

*Replace(text, pattern, replacement)* – замества всички срещания за даден регулярен израз в даден текст със заместващ текст, който може да съдържа части от намерените съвпадения (групи в рег. израз)

*Класът* ***Match*** съдържа описание на едно съвпадение (стойност, начална позиция и дължина), получено в резултат от търсене с регулярен израз

*По-важни свойства и методи на* ***Match***:

*Success* – връща дали съвпадението е

валидно – дали търсенето е намерило нещо

*Value* – връща стойността на съвпадението

*Index* – връща позицията на съвпадението в текста

*Length* – връща дължината на съвпадението

*NextMatch()* – предизвиква продължаване на търсенето от края на текущата съвпадение и връща следващото съвпадение (ако има)

*Groups* – връща групите, съдържащи се в съвпадението във вид на GroupCollection

*Captures* – връща CaptureCollection от Capture обектите, образуващи съвпадението

*Други*:

*Класът MatchCollection* съдържа списък от съвпадения (получени в резултат от търсене)

*Класът Group* представлява група от символи, съдържаща се в дадено съвпадение (Match). В едно съвпадение може да има няколко групи

*Класът GroupCollection* съдържа списък от групи, съдържащи се в дадено съвпадение

Делегатът *MatchEvaluator* се използва при заместване с регулярен израз за обработка на всяко едно съвпадение

Изброеният тип *RegexOptions* се използва за задаване на опции за търсенето с рег. изрази

**|?| Възможно ли е чрез регулярен израз да се провери дали скобите в даден числов израз са поставени правилно (дали за всяка отваряща скоба има съответстваща затваряща). Защо?**

Да???

--Lecture 10--

**|?| Какво знаете за автоматичното управление на паметта и ресурсите в .NET Framework?**

**Какви са предимствата и недостатъците на автоматичното управление на паметта?**

**Как работи т. нар. garbage collector?**

Динамично-заделените обекти се разполагат в т. нар. Managed Heap

Неизползваните обекти се почистват автоматично от т. нар. Garbage Collector

При "препълване" на динамичната памет се включва системата за почистване на паметта. Тя анализира динамичната памет и освобождава тези обекти от нея, които не се използват от програмата.

*Предимства:*

Не трябва ръчно да се освобождава паметта като в C++

Не се получава "изтичане на памет" (memory leaks)

Не е възможно четене и писане по освободена памет или повторно освобождаване

Ресурсите винаги се освобождават

Паметта се заделя много бързо

*Недостатъци*:

Някои ресурси трябва да се управляват ръчно

Няма гаранция кога се изпълнява garbage collector-ът и колко време отнема

*Garbage collector-*ът се активира при създаване на нов обект, когато има недостиг на памет. Изчакват се всички нишки да достигнат безопасно състояние и се приспиват. Намират се използваните от всички нишки обекти и всички обекти, достижими от тях. Идентифицират се ненужните обекти и тези, които изискват финализация се преместват специална опашка.

**|?| Какво знаете за финализацията и интерфейса IDisposable в .NET Framework?**

**Кога се използват? Как се реализират?**

Финализацията осигурява правилното освобождаване на ресурсите, използвани от обектите. При почистване на паметта обектите, които имат метод за финализация, не се освобождават веднага – прехвърлят се от Finalization Queue във Freachable Queue.

Финализицията се извършва от отделна нишка за всеки обект от Freachable Queue.

Интерфейсът IDisposable позволява изрично (ръчно) освобождаване на ресурси.

Препоръчва се винаги IDisposable и Finalize() да се имплементират заедно.

- Всеки наследник на System.Object може да дефинира метод за финализация Finalize() (или деструктор в C#) - извиква се преди даден обект да бъде унищожен от системата за почистване на паметта.

- Методът IDisposable.Dispose() трябва да освободи използваните ресурси и да извика GC.SuppressFinalize()

--Lecture 11--

**|?| Какво представляват потоците в .NET Framework? Кои са основните операции с тях?**

**Кои са основните класове за работа с потоци?**

Абстракцията "поток" е основният начин за осъществяване на входно-изходна активност в съвременните обектно-ориентирани езици.

В .NET Framework потоците са два вида:

*Базови потоци (base stream*) - четат и пишат данни от и към външен механизъм за съхранение на данни

Преходни потоци (pass-through streams) - четат и пишат в други потоци, като добавят допълнителна функционалност (напр. буфериране, кодиране и компресиране)

*Основните операции с потоци са:*

*Конструиране (създаване)* - потокът се свързва с механизма за пренос/съхранение на данни или с друг поток и като параметър в конструктора на класа се подава информация за този механизъм

*Четене* - извличат се данни от потока и в зависимост от типа на потока тези данни се извличат по различен начин

*Писане* - изпращат се данни в потока

*Позициониране* - премества текущата позиция на потока спрямо текущата позиция, началото или края на потока

*Затваряне* - завършва се работата с потока и се освобождават използваните ресурси

Основни класове:

Базов клас за всички потоци е абстрактният клас System.IO.Stream - В него са дефинирани методи за извършване на основните операции

- System.IO.Stream

- System.IO.BufferedStream

- FileStream

- TextReader и TextWriter

- File и FileInfo

- Directory и DirectoryInfo

- Path

- FileSystemWatcher

--Lecture 12--

**|?| Какво представлява езикът XML? За какво служи? Кога се използва?**

Универсален език (нотация) за описание на структурирани данни - данните се съхраняват заедно с мета-информация за тях. С него се описват други езици (формати) за представяне на данни.

*Използване:*

- За обмяна на информация между различни системи

- За съхранение на структурирани данни

- За създаване на собствени езици за описание на информация

**|?| Какво представляват пространствата от имена в XML документите?**

**За какво служат? Кога се използват?**

Пространствата от имена (namespaces) в XML документите позволяват дефиниране и използване на тагове с еднакви имена

Кога се използват?

**???**

**|?| Какво представляват XML схемите? По какво си приличат и по какво се различават DTD, XSD и XDR схемите?**

Описват структурата на XML документите - допустимите тагове, допустимите атрибути на таговете, допустимите стойности за атрибутите и на елементите, реда на поставяне на таговете, стойности по подразбиране.

Има няколко стандарта за XML схеми:

DTD – Document Type Definition

XSD – XML Schema Definition Language

XDR – XML-Data Reduced

**???** *XSD* схемите имат по-голяма изразителна мощ от DTD

*XDR* - XML базиран език, подобен на XSD. По-мощен от DTD. По-слабо изразителен от XSD

**|?| Защо не можем да използваме XSLT за преобразуване на XML документ към произволен друг текстов формат, а само към друг XML?**

**???**

Защото XSLT разчита на XPath (език за адресиране на части от XML документи) за извличане на части от входния документ, които се използват в изходния документ.

В частност XSLT може да се използва и запреобразуване на XML документи в XHTML