1. Платформа Microsoft .NET предлагает управляемую среду выполнения приложения, в основании которой лежит виртуальный высокоуровневый процессор.
2. CLR (Common Language Runtime) — это среда выполнения приложений, в основе которой лежит виртуальный высокоуровневый процессор, и которая реализует работу с памятью, процессами, потоками и остальным окружением
3. CTS — это формальная спецификация, в которой описано то, как должны быть определены типы для того, чтобы они могли обслуживаться в CLR.

Ссылочные (reference types):

- Классы

- Интерфейсы

- Делегаты

Значимые (value types):

- Перечисление

- Структуры (также все стандартные типы данных: int, double, etc).

1. CLS — это подмножество спецификации CTS. CLS является набором правил, которых должны придерживаться создатели компиляторов при желании, чтобы их продукты могли без проблем функционировать в .NET. Главным в CLS является правило, гласящее, что правила CLS касаются только той части типов, которые делаются доступными за пределы сборки, в которой они определены. Как определили, что входит в CLS? Из возможности библиотеки BCL.
2. BCL (base class library) - стандартная библиотека базовых классов платформы .NET. Программы, написанные на любом из языков, поддерживающих платформу .NET, должны иметь возможность пользоваться всеми классами и методами BCL.
3. FCL (framework class library) - стандартная библиотека всех базовых классов и фреймворков платформы .NET. То есть библиотека BCL — это подмножество библиотек FCL. отличие от BCL, языки, поддерживающие платформу .NET, не обязаны предоставлять одинаково полный доступ ко всем классам и всем возможностям FCL.
4. Язык CIL (Common Intermediate Language). Старое название - MSIL или IL.

Плюсы .NET:

- межплатформенность;

- межязыковость;

- управляемость - управляемая среда. .NET позволяет писать managed & safe code - нет утечек памяти и нельзя “промахнуться” указателем - за это платим прослойкой, которое контролирует это.

- сборка мусора. Сборщик мусора - магия в С# и Java, которая самая занимается delete’ом.

1. Assebly – сборка это пакет из семантически близких модулей, состоящий из одного или нескольких файлов. Предполагает полную инкапсуляцию реализуемых функциональных возможностей.
2. Манифест сборки - содержит метаданные, описывающие требования сборки к версиям окружения и к безопасности, а также необходимые для определения области видимости сборки и разрешения ссылок на ресурсы и классы.
3. Модуль — это общий термин для обозначения действительного двоичного файла .NET. Технически сборка может состоять из нескольких модулей.
4. JIT (just in time compiler) - компилятор, который переводит машинные команды виртуального процессора (CIL) в машинные команды настоящего.
5. Обфускация - затруднения для понимания исходного кода, запутывание и устранение видимых логических связей в коде. Дотфускация - обфускация для платформы .NET.
6. Вывод типа данных. В С++ auto. В C# var. Стандартные типы не var’ят. Предоставляет возможность компилятору самому определить тип локальной переменной, исходя из значения, которым она инициализируется.
7. Массивы реализованы в виде объекта, это ссылочный тип данных, соответственно располагаются в динамической памяти. Могут быть массивы объектов, многомерные массивы, рваные массивы.
8. Класс Array – это абстрактный класс, который является предком всех создаваемых массивов.
9. String – это строковый тип данных, строки являются объектами. Интернирование строк - строковые литералы, определенные в коде, хранятся в единственном экземпляре. Строки в CLR неизменяемые (immutable). Любая попытка изменить строку приводит к созданию новой строки.
10. Сравнение строк (два способа):

- порядковое сравнение. Подходит в случае, когда строки используются для внутренних или системных целей.

- сравнение строк с учетом культурной среды - по умолчанию (та, что установлена уже). Следует использовать, если результат будет показан пользователю.

1. Класс System.Object как универсальный тип данных. Все уже созданные и наши будущие классы являются потомками класса Object => есть возможность сделать массив объектов разных классов. Т.к. Object является базовым классом для всех остальных типов, а упаковка значений значимых типов происходит автоматически, то его можно использовать в качестве “универсального” типа данных. Однако, с C#2.0 появились подлинно обобщенные типы данных - generics (обобщения / шаблоны). Благодаря обобщениям отпала необходимость использовать Object как универсальный тип данных. Универсальный характер этого класса лучше теперь оставить для применения в особых случаях.
2. При компиляции const вставляется прямо по месту использования. Таким образом const могут быть только value - типы (значимые типы).
3. При работе программы readonly — это модификатор, запрещающий изменение значения поля после вызова конструктора (в самом конструкторе еще можно).
4. Статический конструктор используется для инициализации статических компонентов класса. Члены класса инициализируются статическим конструктором до создания каких-либо объектов этого класса. Статический конструктор вызывается автоматически, когда класс загружается впервые, причем до любого конструктора экземпляра. Более того, у статических конструкторов отсутствуют модификаторы доступа - они пользуются доступом по умолчанию, а, следовательно, их нельзя вызывать из программы напрямую.
5. Сцепление конструктора (Constructor chaining) (А в С++ это делегирование).
6. Статический класс служит, во-первых, для хранения совокупности связанных друг с другом статических методов, а во-вторых, статический класс требуется при создании методов расширения (которые, в основном, связаны с языком LINQ).
7. Для освобождения ресурсов, не контролируемых сборщиком мусора, в самом .NET Framework в классе object реализован виртуальный защищенный метод Finalize, который вызывается сборщиком мусора перед непосредственным уничтожением объекта. Явно переопределить метод Finalize нельзя - компилятор выдает ошибку.
8. **ref и out** – аналог ссылок в С++, для передачи аргументов и обьектов в по ссылке. Разница между ними в том, что ref должен быть проинициализирован до передачи в метод, а out конкретно в методе, который его принимает.
9. **in** - параметр передается по ссылке (как ref или out), но в методе рассматривается как константный и не может быть изменен. Никак не обозначается при вызове метода, поскольку важен только в самом методе.
10. Исключение - исключительные ситуация - разнообразные аварийные ситуации, которые могут возникать на этапе выполнения программы. В .NET есть структурированная обработка исключений SEH (Structured Exception Handling). SEH (Structured Exception Handling) - эта методика подразумевает, что исключения представляют собой объекты, в которых содержится читабельное описание проблемы, а также детальный снимок стека вызовов на момент возникновения исключения.
11. Исключения уровня системы (класс System.SystemException) — это исключения, которые генерируются самой платформой .NET.
12. Исключения уровня приложения (класс System.ApplicationException) — это собственные исключения разработчика приложения;
13. Блок finally будет выполняться каждый раз, когда происходит выход из блока try / catch, независимо от причин, которые к этому привели. То есть, если блок try завершается нормально, из-за исключения или из-за возврата из метода, то последним в любом случае выполняется код из блока finally.
14. Фильтры исключения позволяют обрабатывать исключения в зависимости от определенных условий.
15. Переполнение — это превышение диапазона представления чисел для числовых типов данных, которое может возникнуть в результате некоторых видов арифметических вычислений.
16. Конструкции checked и unchecked - специальные средства, связанные с генерированием исключений, возникающих при переполнении.
17. Пространство имен – создает дополнительную область видимости и предотвращает конфликты, связанных с повторением идентификаторов в разных модулях программы. Вложенные пространства имен можно вкладывать друг в друга для построения логической иерархии. При объявлении типа вне пространства имен, компилятор назначит ему по умолчанию глобальное пространство имен.
18. Перегрузка осуществляется статическими методами. Операции сокращенного присваивания тоже работают если перегружены обычные операции. Операции отношений, а также true и false перегружаются попарно. При перегрузке «! =» и «==» требуется переопределить методы Object.Equals() и Object.GetHashCode().
19. Лучше прибегать к явному преобразованию, так как не теряется информация, преобразование не приводи к исключительной ситуации.
20. Индексатор — это возможность проектировать классы и структуры, которые могут быть индексированы подобно стандартному массиву. В индексаторе не определено место в памяти под его хранение, поэтому его нельзя передавать через ref или out.
21. Свойства — это специальные методы доступа к полям классов и структур. Свойство не определяет место для хранения данных, и поэтому не может быть передано методу в качестве параметра. Компилятор сам создает так называемое поддерживающее поле и аксессоры для него, по сути разворачивается в get и set для которых можно установить разные модификаторы доступа.
22. В C#/CLR реализовано только одиночное наследование, и оно только public. Объект производного класса в то же время является и объектом базового класса. Сокрытие имен — это обращение из производного к базовому классу.
23. Ключевое слово «base» определяет какой конструктор базового класса следует вызвать.
24. Цепочки виртуальных методов иерархии классов — это про то, когда мы в подклассах пишем override к virtual методу абстрактного класса, но потом в одном из подклассах приписываем к этому методу virtual вместо override, таким образом мы начинаем новую цепочку переопределения virtual метода.
25. Интерфейс – это публичный контракт. Реализация интерфейса описывается так же, как и наследование. При явной реализации члена интерфейса, при множественном наследовании интерфейсов, указывается имя метода, с именем интерфейса, и вызвать такой метод можно через интерфейсную ссылку.
26. Интерфейсные ссылки используют операции «is» - bool и «as» как dynamic\_cast.
27. Структуры - значимые типы. Конструктор по умолчанию в структурах реализуется CLR и инициализирует поля значениями по умолчанию для их типов. Если объявить структуру без new поля будут заполнены с мусором, а с new поля будут инициализированы дефолтными значениями. Структуры не поддерживают наследование и соответственно полиморфизм.
28. Перечисление – это пользовательский целочисленный тип. Специфицирует набор допустимых значений, которые могут принимать экземпляры перечисления. Преобразование не определены, поэтому нужно явное приведение типов.
29. Nullable - типы аналогия nullptr (который доступен только ссылочным типам) для значимых типов. Это официальное значение «в переменной содержится ничего».
30. null -объединения «??» предназначено для установки значение по умолчанию для значимых типов и ссылочных, которые допускают значение null.
31. Делегат — это тип, который несет идею указателя на функцию для реализации call back функции. Это абстракция указателя на функцию (т.к. всё есть класс, то указателя на метод). Это класс, объект которого может ссылаться на метод (методы) некого класса (классов) и вызывать его (их) по ходу работы программы. Делегат — это ссылочный тип данных, но внешне не выглядит как класс.
32. Реализация — это сгенерированный компилятором запечатанный класс, унаследованный от System.MulticastDelegate, который унаследован от System.Delegate (который потомок System.Object), но от этих классов напрямую наследовать нельзя (иначе будет ошибка компилятора).
33. Групповое преобразование делегируемых методов — это возможность присвоить имя метода делегату, не прибегая к оператору new или явному вызову конструктора делегата.
34. Групповая адресация — это возможность вызвать несколько методов через интерфейс одного делегата (multicasting) посредством создания коллекции делегатов и организации цепочки вызова их метода.
35. Ковариантность (Covariance) — это возможность присвоить делегату метод, возвращаемым типом которого служит класс, производный от класса, указываемого в возвращаемом типе делегата.
36. Контравариантность (Contravariance) - возможность присвоить делегату метод, типом параметра которого служит класс, являющийся базовым для класса, указываемого в объявлении делегата.
37. Обобщенные делегаты — это заготовленные типы делегатов для передаваемого в качестве параметра метода без явного объявления пользовательского делегата для него. Action<T> инкапсулирует методы без возвращаемого значения: тип возвращаемого значения void ; Func<T> - тип возвращаемого значения указывается последним в списке параметров в < >. В первую очередь, это удобно при передаче в метод в качестве параметра лямбда-выражения.
38. Анонимные функции — это безымянные блок кода, полностью определенные внутри другого метода и передаваемые конструктору делегата. Анонимная функция имеет тип делегата; реализуют идею замыкания (!).
39. Анонимный метод — это безымянный блок кода после ключевого слова delegate.
40. Замыкание — это анонимная функция, находящаяся целиком в теле другого метода и захватывающая локальные переменные этого внешнего метода в свой контекст.
41. Лямбда-выражения — это еще один способ реализовать замыкание. Виды лямбда-выражения. - одиночное; - блочное.
42. События — это механизм автоматического уведомления о том, что в зоне ответственности класс, которому принадлежит это событие, произошло некоторое изменения. Событие - это публичный элемент элемент класса представляющий из себя синтаксическую обертку вокруг поля-делегата и двух методов для более простого добавления и удаления подписчиков (методов-подписок) в этот делегат; - по сути, это синтаксический сахар (для поля-делегата и служебных методов), который предназначены для реализации паттерна Observer.
43. Аксессоры события. Для самостоятельного управления списком обработчиков событий служит расширенная форма оператора event с аксессорами событий add и remove.
44. Методы расширения - возможность добавить метод классу на лету. Реализация паттерна Decorator. Механизм методов расширения позволяет как бы добавить к уже созданному объекту нашего класса методы, которых в нашем классе нет. Расширять методами расширения можно не только класс, но и интерфейсы.
45. Обобщения (Generics) - параметризованный тип; сущности CLR.
46. Ограничения параметров типа — это возможность наложить определенные ограничения на параметры типов в обобщении (оператор where). Виды ограничений

* Ограничение на базовый класс (только этот класс или его потомки).
* Ограничения на интерфейс (объект класса, который реализует интерфейс).
* Ограничения на конструктора.
* Ограничение ссылочного типа.
* Ограничение типа значения.

1. Иерархия обобщенных классов. Главное отличие между иерархиями обобщенных и не обобщенных классов заключается в том, что в первом случае аргументы типа, необходимые обобщенному базовому классу, должны передаваться всеми производными классами вверх по иерархии аналогично передачи иерархии конструктора.
2. Обобщенные методы — это определенный внутри не обобщенного класса метод, в объявлении которого присутствуют обобщенный тип.
3. Обобщенные делегаты — это класс, а значит он тоже может быть обобщенным.
4. Обобщенные интерфейсы и структуры. Ковариантность в обобщенном интерфейсе. В обобщенном интерфейсе ковариантность разрешает методу возвращать тип, производный от класса, указанного в параметре типа. Параметр ковариантного типа объявляется с помощью ключевого слова out, которое предваряет имя этого параметра.
5. Контравариантность в обобщенном интерфейсе. В обобщенном интерфейсе контравариантность разрешает методу использовать аргумент, тип которого относится к базовому классу, указанному в соответствующем параметре типа. Параметр контравариантного типа объявляется с помощью ключевого слова in , которое предваряет имя этого параметра. Контравариантность оказывается пригодной только для ссылочных типов, а параметр контравариантного типа можно применять только к аргументам методов. Следовательно, ключевое слово in нельзя указывать в параметре типа, используемом в качестве возвращаемого типа.
6. Гипотеза поколений (generation ). Большинство размещений в памяти “умирают молодыми”, т.е., становятся мусором через очень короткое время после размещения. Предположения, лежащие в основе реализации сборщика мусора:

- если объект уже долго живет, вероятно, он и дальше будет жить долго;

- если объект недавно был создан, вероятно, он скоро умрет;

- очищать небольшой блок в куче проще, чем по отдельности.

Поколения в управляемой куче - их три: 0 (256кб), 1 (512кб), 2 (1мб).

Важно: стандарт CLI не требует, чтобы в сборщике мусора было реализовано именно 3 поколения. В .NET от Microsoft их действительно 3, но например, в Mono их 2.

1. Работа сборщика мусора. Запуск сборщика мусора происходит тогда, когда необходимо создавать новый объект, а поколение оказывается заполненным.

- все объекты в поколении помечаются как достижимые (есть “живые ссылки” на них) или недостижимые (на ни больше никто не ссылается).

- достижимые перемещаются в поколение выше (или просто остаются на месте, если они уже в самом высоком поколении).

- недостижимые либо сразу уничтожаются (есть не имеют переопределенного финализатора), либо перемещаются в очередь финализации (если имеют переопределенный финализатор).

1. Очередь финализации - очередь, в которую попадают объекты из поколения, подлежащие удаление.
2. Деструктор (который на самом деле финализатор) - унаследованный от Object метод Finalize. Его нельзя переопределить.
3. Все коллекции разработаны на основе набора четко определенных интерфейсов

- два основных интерфейса, реализуемых всеми контейнерами:

- ICollection / ICollection<T>;

- IEnumerable / IEnumerable<T>.

Основополагающим для всех коллекций является так называемый перечислитель (enumerator).

Реализует паттерн Iterator и позволяет последовательно перебрать коллекцию.

1. Перечислитель (enumerator) — это стандартный способ поочередного доступа к элементам коллекции.
2. Для коллекции пар “ключ-значение” используется перечислитель, задаваемый интерфейсом IDictionaryEnumerator.
3. Для получения перечислителя во всех коллекциях реализован метод GetEnumerator(). На самом деле все коллекции реализуют интерфейс IEnumerable / IEnumerable<T>, который и требует реализации этого метода. Также реализует IEnumerator / IEnumerator<T>.
4. Цикл foreach — это синтаксический сахар. Как сделать, чтобы наш класс можно было использовать с foreach. Считается, что для этого необходимо реализовать IEnumerable / IEnumerable<T>. А тот в свою очередь требует IEnumerator / IEnumerator<T>.
5. Итераторы — это метод, операция или аксессор, возвращающий по очереди элементы коллекции от ее начала и до конца.
6. Ключевое слово yield имеет специальное назначение только в теле итератора. Вне этого тела оно может быть использовано аналогично любому другому идентификатору. Используется в двух конструкциях:

- return yield - возвращает следующий элемент коллекции;

- yield break - уведомляет о том, что достигнут конец коллекции, т.е., используется для досрочного прерывания итератора.

Итератор может быть

- обычным;

- именованным

1. Две проблемы, которые решает итератор:

- память. Нужно получить лишь несколько элементов из коллекции.

- многоуровневые операции (например, получить миллион элементов, потом их преобразовать, еще что-то сделать. При этом каждый раз будет создаваться коллекция).

1. Именованный итератор - позволяет передавать аргументы итератору, управляя процессом получения конкретных элементов коллекции.
2. Виды коллекций

- необобщенные;

- обобщенные (шаблонные, дженерики);

- специальный;

- поразрядные;

- параллельные.

1. Необобщенные коллекции — это динамические структуры данных общего назначения, оперирующие ссылками типа Object.
2. Упаковка (boxing) — это процесс, при котором экземпляр значимого типа из стека попадает в управляемую кучу. По сути, значимый типа (структура или перечисление) становится ссылочным типов (классом).
3. Распаковка (unboxing) — это процесс, обратный упаковке: объект ссылочного типа преобразуется в значимый экземпляр, который будет помещен в стек. Аналогично упаковке, созданный при распаковке экземпляр значимого типа никаким образом не будет связан с ссылочным объектом.
4. Обобщенные коллекции - пространстве имен System.Collections.Generic — это динамические структуры данных общего назначения, оперирующие объектами указанного типа.
5. Специальные коллекции — это коллекции, оптимизированные для работы с данными конкретного типа или для их обработки особым образом.
6. Наблюдаемые (observable) коллекции. Класс ObservableCollection<T>. Позволяет создать коллекцию, которая умеет уведомлять о добавлении или удалении элементов коллекции.
7. Коллекции с поразрядной организацией. Класс BitArray - совокупность битов. Физически хранится в виде динамического массива целых чисел. Поддерживает поразрядные операции (например, and, or, xor) и существенно отличается своими возможностями от остальных типов коллекции. System.Collections.
8. Параллельные коллекции — это обобщенные коллекции, поддерживающие многопоточный доступ к своему содержимом.
9. Собственные коллекции - в System.Collections.ObjectModel находится ряд классов, поддерживающих создание разработчиками собственных обобщенных коллекций.
10. Сравнение объектов. В Collection API предусмотрены два способа сравнения объектов:

- в самом классе;

- при помощи внешних компараторов.

Интерфейс IComparable / IComparable<T>, Интерфейс IComparer / IComparer<T>.

1. Регулярные выражения - механизм поиска подстроки в строке по шаблону (поиск любой подстроки, подходящей под маску).
2. Метасимвол — это специальный символ, задающий команду или управляющую последовательность. Предваряется знаком обратного слеша ‘\’ и имеет специальное значение.
3. Жадность квантификаторов:

Жадные

- пытаются найти самое длинное вхождение заданного шаблона, даже если внутри него оказываются более короткие вхождения этого шаблона.

Ленивые (отложенные)

- пытаются найти самое короткое вхождение заданного шаблона;

- чтобы превратить жадный в ленивый достаточно в нему добавить символ “?”.

1. Рефлексия (reflection) - рефлексия означает самоосознание — это механизм программного получения метаданных любого типа в виде объектной модели во время выполнения.
2. XML (eXtensible Markup Language) - это универсальный язык разметки - имеет древовидную структуру.
3. Валидация XML-документа — это проверка XML-документа на соответствие требованиям грамматики XML.
4. XML-парсер (XML-анализатор) — это API, при помощи которого можно прочитать XML-документа и извлечь из него данные, не разбирая синтаксис XML вручную. Определяет API, позволяющие абстрагироваться от физической природы XML-документа.

Виды:

- DOM (Document Object Model) - это построенное в памяти полное дерево элементов,

отображающие структуру XML-документа. Дает удобные возможности анализа содержимого

XML-документа , но требователен к памяти, т.к считывает сразу весь документ.

Стандартизирован W3C.

- SAX (Simple Api for Xml) - это API на основе событий.

1. Атрибуты — это класс, унаследованный от базового класса Attribute, используемый для создания описания типа и его членов;

- обеспечивают эффективный способ связывания метаданных или декларативной информации с кодом и могут быть получены при помощи рефлексии;

- являю

1. Сериализация — это процесс преобразования объекта в последовательность байтов, которую можно хранить и передаваться элементом парадигмы аспектно-ориентированного программирования.
2. Десериализация - это процесс обратного преобразования созданной последовательности байтов в объект.
3. Графы объектов. Идея в том, что сериализовать надо не только сам объект, но и все вложенные в него объекты. Граф объектов — это набор взаимосвязанных объектов, участвующих в сериализации. Представляет простой способ документирования набора отношений между объектами.
4. Ключевое слово dynamic - позволяет опустить проверку типов во время компиляции и дает dynamic -объекту возможность в течении работы программы менять свой тип;

- может быть применено не только к переменным, но и к свойствам и методам;

- можно использовать в качестве типа возвращаемого значения метода.

1. Класс ExpandoObject позволяет создавать динамические объекты в стиле js. Динамическому объекту класса ExpandoObject можно на этапе выполнения добавить любые свойства и (с помощью делегатов) методы.