1. Концепции ООП (наследование, инкапсуляция, полиморфизм)  
     
   Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали   
   реализации от пользователя.  
     
   Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.  
     
   Наследование – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс – потомком, наследником или производным классом.
2. Модификаторы доступа (private, public, protected, internal)  
     
   Public - Неограниченный доступ.  
   Private - Доступ ограничен содержащим типом.  
   Protected - Доступ ограничен содержащим классом или типами, которые являются производными от содержащего класса.  
   Internal - Доступ ограничен текущей сборкой.  
   protected internal- Доступ ограничен текущей сборкой или типами, которые являются производными от содержащего класса.
3. Корень всех типов «System.Object» В конечном счете все типы происходят от типа System.Object, что позволяет гарантировать наличие у каждого типа минимального набора функциональных возможностей. Все типы получают «бесплатно» четыре открытых метода: bool Equals(), int GetHashCode(), Type GetType(), string ToString.
4. Различие между интерфейсом и абстрактным классом  
     
   Абстрактный класс — класс, который имеет хотя б 1 абстрактный метод; его обозначают abstract.  
   Интерфейс — такой же абстрактный класс,только в нем нету свойств и не определены тела у методов.  
     
   Таким образом, функционально любой интерфейс является абстрактным классом, но абстрактный класс не является интерфейсом.  
     
   Основные отличия интерфейса и абстрактного класса  
   • абстрактный класс может содержать свойства  
   • интерфейс не содержит свойств  
   • абстрактный класс может содержать тела методов  
   • интерфейс не содержит тел методов, только их объявление  
   • абстрактный класс наследуется (etxends)  
   • интерфейс реализуется (implements)  
   • наследник имеет только одного родителя (абстрактный класс)  
   • наследник может реализовывать методы нескольких интерфейсов  
   • наследник не обязан содержать реализацию методов абстрактного класса-родителя  
   • наследник обязан содержать реализацию всех методов реализуемого интерфейса
5. С# поддерживает множественное наследование в виде наследования от класса и нескольких интерфейсов, или просто от нескольких интерфейсов.  
   Но не поддерживает наследование от нескольких классов.
6. Понятие "ссылочный тип"  
     
   Переменные ссылок хранят ссылки на свои данные (объекты), в то время как переменные типа значения непосредственно содержат свои данные. Две переменные ссылочного типа могут ссылаться на один и тот же объект, поэтому операции над одной переменной могут затрагивать объект, на который ссылается другая переменная. Каждая переменная типа значения имеет собственную копию данных, и операции над одной переменной не могут затрагивать другую (за исключением переменных параметров out и ref)
7. Разница между структурой и классом  
     
   Структуры синтаксически очень похожи на классы, но существует принципиальное отличие, которое заключается в том, что класс – является ссылочным типом (reference type), а структуры – значимый класс (value type). А следовательно, классы всегда создаются в, так называемой, “куче” (heap), а структуры создаются в стеке (stack).Цитата из комментария: "Имхо, главное отличие структур и классов: структуры передаются по значению (то есть копируются), объекты классов - по ссылке. Именно это является существеннейшим различием в их поведении, а не то, где они хранятся. Примечание: структуру тоже можно передать по ссылке, используя модификаторы out и ref."
8. Делегаты (общие сведения)  
     
   Делегат представляет собой объект, который может ссылаться на метод. Следовательно, когда создается делегат, то в итоге получается объект, содержащий ссылку на метод. Более того, метод можно вызывать по этой ссылке. Иными словами, делегат позволяет вызывать метод, на который он ссылается.  
   По сути, делегат — это безопасный в отношении типов объект, указывающий на другой метод (или, возможно, список методов) приложения, который может быть вызван позднее.
9. Как работает сборщик мусора?  
     
   При попытке обнаружить недостижимый код объекты CLR-среды не проверяют буквально каждый находящийся в куче объект. Очевидно, что на это уходила бы масса времени, особенно в крупных проектах.  
   Для оптимизации процесса каждый объект в куче относится к определённому «поколению» Смысл в применении поколений выглядит довольно просто:  
   Чем дольше объект находится в куче, тем выше вероятность того, что он там будет оставаться.  
     
   Например, класс, определённый в главном окне настольного приложения, будет оставаться в памяти вплоть до завершения программы. С другой стороны, объект, попавший в кучу совсем недавно (например те, что находятся в области видимости методов), вероятнее всего будут становиться недостижимыми достаточно быстро. Изводя из этих предположений, каждый объект в куче относится к:  
   Поколение 0. Идентифицируется новый только что размещённый объект, который ещё никогда не помечался как надлежащий удалению в процессе сборки мусора  
   Поколение 1. Идентифицирует объект, который уже «пережил» один процесс сборки мусора.  
   Поколение 2. Идентифицирует объект, который пережил более одного прогона сбора мусора  
   Поколения 0 и 1 называются эфемерными.
10. Finalize() наиболее близко соответствует концепции традиционного деструктора. Если объявить в классе метод Finalize() он будет автоматически вызываться при уничтожении экземпляра класса.  
    Finalize() является детерминированным, т.е. нет способа выяснить, когда будет вызван сборщик мусора и соответственно Finalize(). Следовательно, в метод Finalize() нельзя помещать код, который должен выполняться в какое-то определенное время. Запустить сборщик мусора можно с помощью метода System.GC.Collect().  
      
    интерфейс IDisposable, который имеет единственный метод по имени Dispose(): когда действительно реализуется поддержка интерфейса IDisposable, то предполагается, что после завершения работы с объектом метод Dispose() должен вручную вызываться пользователем этого объекта, прежде чем объектной ссылке будет позволено покинуть область действия. Благодаря этому объект может выполнять любую необходимую очистку неуправляемых ресурсов без попадания в очередь финализации и без ожидания того, когда сборщик мусора запустит содержащуюся в классе логику финализации.  
    Интерфейс IDisposable может быть реализован как в классах, так и в структурах (в отличие от метода Finalize(), который допускается переопределять только в классах), потому что метод Dispose() вызывается пользователем объекта (а не сборщиком мусора).
11. Значение Using непосредственно связано с интерфейсом IDisposable. Интерфейс IDisposable представляет нам возможность быстро освободить общие ресурсы, не полагаясь на автоматический сборщик мусора (garbage collector).  
    Конструкция Using позволяет вызывать метод Dispose автоматически, как только нужный объект выйдет за блок Using.
12. object Current { get; } bool MoveNext(); void Reset();Благодаря такой реализации мы можем перебирать объекты в цикле foreach
13. «А что нужно, чтобы конструкция **foreach** успешно компилировалась?». Интуитивным ответом на этот вопрос кажется что-то типа: «Реализация классом интерфейса **IEnumerable** или **IEnumerable<T>**.». Однако, это не так, ну, или не совсем так.

Полный ответ на этот вопрос такой: «Для того чтобы конструкция **foreach** успешно компилировалась необходимо, чтобы у объекта был метод **GetEnumerator()**, который вернет объект с методом **MoveNext()** и свойством **Current**, а если такого метода нет, то тогда будем искать интерфейсы **IEnumerable** и **IEnumerable<T>**».

1. Интерфейс IEnumerable находится в пространстве имен **System.Collections**. Объект IEnumerable представляет набор данных в памяти и может перемещаться по этим данным только вперед. Запрос, представленный объектом IEnumerable, выполняется немедленно и полностью, поэтому получение данных приложением происходит быстро.

При выполнении запроса IEnumerable загружает все данные, и если нам надо выполнить их фильтрацию, то сама фильтрация происходит на стороне клиента.

Интерфейс IQueryable располагается в пространстве имен **System.Linq**. Объект IQueryable предоставляет удаленный доступ к базе данных и позволяет перемещаться по данным как в прямом порядке от начала до конца, так и в обратном порядке. В процессе создания запроса, возвращаемым объектом которого является IQueryable, происходит оптимизация запроса. В итоге в процессе его выполнения тратится меньше памяти, меньше пропускной способности сети, но в то же время он может обрабатываться чуть медленнее, чем запрос, возвращающий объект IEnumerable.

1. ***Словарь (dictionary)*** представляет собой сложную структуру данных, позволяющую обеспечить доступ к элементам по ключу. Главное свойство словарей — быстрый поиск на основе ключей. Можно также свободно добавлять и удалять элементы, подобно тому, как это делается в List<T>, но без накладных расходов производительности, связанных с необходимостью смещения последующих элементов в памяти.

Тип, используемый в качестве ключа словаря, должен переопределять метод GetHashCode() класса Object. Всякий раз, когда класс словаря должен найти местоположение элемента, он вызывает метод GetHashCode().

Целое число, возвращаемое этим методом, используется словарем для вычисления индекса, куда помещен элемент.

1. При использовании блока **try...catch..finally** вначале выполняются все инструкции в блоке **try**. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок **finally**. И затем конструкция try..catch..finally завершает свою работу.Если же в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда CLR начинает искать блок **catch**, который может обработать данное исключение. Если нужный блок catch найден, то он выполняется, и после его завершения выполняется блок finally.Если нужный блок catch не найден, то при возникновении исключения программа аварийно завершает свое выполнение.
2. Строка иммьютабельна (её невозможно изменить после того, как она созадана). Если вы что-то прибавляете к строке, то в действительности вы не получите старую строку + добавку, вы получите новую страку, составленную из старой и добавки. Отсюда все проблемы с производительностью в приложениях, которые последовательно формируют большие строки.  
     
   StringBuilder же будет последовательно выделять память и "по-настоящему" дозаписываться данными
3. В первом приближении управляемым кодом называется код, выполнение которого управляется средой выполнения. В этом случае соответствующая среда выполнения называется **общеязыковой средой выполнения** или средой CLR, независимо от реализации ([Mono](https://www.mono-project.com/), .NET Framework или .NET Core).Среда CLR отвечает за использование управляемого кода, его компиляцию в машинный код и последующее выполнение. Кроме того, среда выполнения предоставляет несколько важных служб, таких как автоматическое управление памятью, границы безопасности, безопасность типа и т. д.
4. –
5. delegate — это класс, содержащий в себе «шаблон» метода, то есть, сигнатуру метода. Переменная делегатного типа — объект типа [MulticastDelegate](http://msdn.microsoft.com/en-us/library/system.multicastdelegate.aspx) (точнее, производного от него), который может содержать один или несколько объектов, представляющих собой методы с совместимой с «шаблоном» сигнатурой (контр- и ковариантность немного усложняет картину). То есть это как бы переменная, которая может содержать функции. Для таких переменных определена операция +, которая комбинирует слагаемые-функции в одну новую функцию, и симметричная операция -. Эти операции автоматически порождают производные операции +=и -=.

event же — это просто пара методов в классе, обозначаемых как add и remove, и имеющих произвольную семантику, выбранную программистом. (Аналог — геттер и сеттер свойства.) В имплементации по умолчанию для event'а заводится скрытое поле делегатного типа, а add/remove добавляют или убирают из него методы (под lock'ом). (Чтобы немного запутать картину, это скрытое поле доступно по тому же имени, что и event.) Функции add/remove, составляющие event, вызываются соответственно как += и -=. Никаких операций +/-, разумеется, нету.

1. Суть Eager Loading заключается в том, чтобы использовать для подгрузки связанных по внешнему ключу данных метод **Include**.

Еще один способ представляет так называемая "ленивая загрузка" или **lazy loading**. При таком способе подгрузки при первом обращении к объекту, если связанные данные не нужны, то они не подгружаются. Однако при первом же обращении к навигационному свойству эти данные автоматически подгружаются из бд.

При использовании ленивой загрузки надо иметь в виду некоторые моменты при объявлении классов. Так, классы, использующие ленивую загрузку должны быть публичными, а их свойства должны иметь модификаторы **public** и **virtual**.

1. –
2. Концепция паттерна (шаблона) MVC (model - view - controller) предполагает разделение приложения на три компонента:

**Контроллер** (controller) представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления.

**Представление** (view) - это собственно визуальная часть или пользовательский интерфейс приложения. Как правило, html-страница, которую пользователь видит, зайдя на сайт.

**Модель** (model) представляет класс, описывающий логику используемых данных.

1. Поля для чтения можно инициализировать при их объявлении либо на уровне класса, либо инициилизировать и изменять в конструкторе. Инициализировать или изменять их значение в других местах нельзя, можно только считывать их значение.
2. -
3. Статические конструкторы имеют следующие отличительные черты:

Статические конструкторы не должны иметь модификатор доступа и не принимают параметров

Как и в статических методах, в статических конструкторах нельзя использовать ключевое слово this для ссылки на текущий объект класса и можно обращаться только к статическим членам класса

Статические конструкторы нельзя вызвать в программе вручную. Они выполняются автоматически при самом первом создании объекта данного класса или при первом обращении к его статическим членам (если таковые имеются)

Статические конструкторы обычно используются для инициализации статических данных, либо же выполняют действия, которые требуется выполнить только один раз

1. EF Core является ORM-инструментом (object-relational mapping - отображения данных на реальные объекты). То есть EF Core позволяет работать базами данных, но представляет собой более высокий уровень абстракции: EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.
2. Entity Framework предполагает три возможных способа взаимодействия с базой данных:

**Database first**: Entity Framework создает набор классов, которые отражают модель конкретной базы данных

**Model first**: сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере.

**Code first**: разработчик создает класс модели данных, которые будут храниться в бд, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы

1. (INNER)JOIN - внутреннее соединение. В результирующем наборе присутствуют только записи, значения связанных полей в которых совпадают.

LEFT JOIN - левое внешнее соединение. В результирующем наборе присутствуют все записи из Table1 и соответствующие им записи из Table2. Если соответствия нет, поля из Table2 будут пустыми.

RIGHT JOIN - правое внешнее соединение. В результирующем наборе присутствуют все записи из Table2 и соответствующие им записи из Table1. Если соответствия нет, поля из Table1 будут пустыми.

FULL JOIN - полное внешнее соединение. Комбинация двух предыдущих. В результирующем наборе присутствуют все записи из Table1 и соответствующие им записи из Table2. Если соответствия нет - поля из Table2 будут пустыми. Записи из Table2, которым не нашлось пары в Table1, тоже будут присутствовать в результирующем наборе. В этом случае поля из Table1 будут пустыми.

CROSS JOIN - Cartesian product. Результирующий набор содержит все варианты комбинации строк из Table1 и Table2. Условие соединения при этом не указывается.

1. Отправляя из браузера запрос к приложению, нередко мы практически моментально получаем нужный контент. Однако в реальности обработка запроса проходит кучу различных этапов. И в этой статье мы рассмотрим основные этапы жизненного цикла приложения на ASP.NET MVC 5.  
     
   1. После получения IIS запроса на обработку, при первом обращении к ресурсу, работающему под управлением CLR, создается объект класса ApplicationManager, который представляет домен приложения, внутри которого обрабатывается запрос. Домены приложений изолируют выполняющиеся приложения друг от друга. Уже внутри домена приложения создается объект классаHostingEnvironment, который предоставляет доступ к информации о приложении, в частности, он сообщает имя и каталог приложения.  
     
   2. После создания домена приложения также создаются и инициализируются такие объекты, как HttpContext, HttpRequest иHttpResponse, которые инкапсулируют всю информацию, связанную с текущим запросом к приложению.  
     
   3. На следующей стадии уже непосредственно запускается приложение, которое представляет экземпляр класса HttpApplication. Если в приложении определен файл Global.asax, то среда ASP.NET в качестве приложения создает объект класса Global.asax, который в свою очередь наследуется от класса HttpApplication. На этой же стадии происходит начальная инициализация приложения в методе Application\_Start, который находится в файле Global.asax.cs.  
     
   4. После этого запрос начинает обрабатываться в конвейере класса HttpApplication - начинается собственно обработка запроса приложением. И первым шагом здесь является установка маршрута. Запрос перехватывается специальным HTTP-модулем под названием UrlRoutingModule. Этот модуль выбирает маршрут, который соответствует входящему запросу.  
   Весь набор маршрутов определяется в файле RouteConfig.cs:  
   А сама регистрация маршрутов происходит в методе Application\_Start строкой RouteConfig.RegisterRoutes(RouteTable.Routes);. Для каждого приложения имеется только один объект коллекции маршрутов RouteTable.  
     
   5. Если модулю UrlRoutingModule удастся сопоставить запрос с одним из маршрутов в коллекции RouteTable, то затем UrlRoutingModule выбирает обработчик маршрутов сопоставленного маршрута - объект IRouteHandler. По умолчанию объект IRouteHandler представляет экземпляр класса MvcRouteHandler.  
     
   6. Затем у объекта IRouteHandler вызывается метод GetHandler, который возвращает объект интерфейса IHttpHandler, используемый для обработки запроса. По умолчанию в качестве IHttpHandlera используется объект класса MvcHandler.  
     
   7. У обработчика IHttpHandler вызывается метод ProcessRequest для обработки А сама регистрация маршрутов происходит в методе Application\_Start строкой RouteConfig.RegisterRoutes(RouteTable.Routes);. Для каждого приложения имеется только один объект коллекции маршрутов RouteTable.  
     
   8. Если модулю UrlRoutingModule удастся сопоставить запрос с одним из маршрутов в коллекции RouteTable, то затем UrlRoutingModule выбирает обработчик маршрутов сопоставленного маршрута - объект IRouteHandler. По умолчанию объект IRouteHandler представляет экземпляр класса MvcRouteHandler.  
     
   9. Далее привязчик модели (по умолчанию класс System.Web.Mvc.DefaultModelBinder) извлекает данные из запроса, производит из преобразование, форматирование, валидацию и связывает их с определенными параметрами вызываемого метода.  
     
   10. При вызове метода в ASP.NET MVC 5 запускается фильтр аутентификации, представляющий объект интерфейса IAuthenticationFilter. Он аутентифицирует пользователя  
     
   11. После фильтра аутентификации в ASP.NET MVC 5 запускается фильтр авторизации, представляющий реализацию интерфейса IAuthorizationFilter. До MVC 5 фильтры аутентификации и авторизации объединялись в один фильтр и срабатывали вместе. Фильтр авторизации управляет доступом пользователя к определенным ресурсам на основе его учетных данных.  
     
   12. Перед непосредственным выполнением метода контроллера запускается метод OnActionExecuting фильтра действий. Фильтр действий представляет объект интерфейса IActionFilter. Кроме того, также после выполнения метода контроллера запускается другой метод фильтра действий - метод OnActionExecuted  
     
   13. Собственно выполнение метода контроллера. Он выполняет определенную логику и на выходе генерирует результат обработки в виде объекта ActionResult.  
     
   14. При обработке результата срабатывает другой фильтр - фильтр результатов - объект интерфейса IResultFilter. Его метод OnResultExecuting срабатывает до обработки результата, а метод OnResultExecutedпосле.  
     
   15. Генерация результата представляет создание объекта одного из классов результатов действий - ViewResult, ContentResult, FileResult, RedirectResult и др.  
   На заключительном этапе у каждого объекта ActionResult вызывается метод ExecuteResult, который обрабатывает результат действия. Для объектов ViewResult и PartialViewResult это выражается в поиске необхожимого представления и его рендеринге движком представлений (как правило, движком Razor), который представляет объект интерфейса IViewEngine. Результат обработки в виде html-страницы посылается пользователю.  
   Для других объектов ActionResult (ContentResult, RedirectRefult и др.) происходит простая отправка результата в выходной поток.  
     
   16. И в конце пользователь получает результат обработки своего запроса.