1. **Какое основное назначение технологии Windows Forms, ее особенности, преимущества и недостатки?**

Windows Forms - интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework.

Данный интерфейс упрощает доступ к элементам интерфейса [Microsoft Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows) за счет создания обёртки для существующего [Win32 API](https://ru.wikipedia.org/wiki/Windows_API) в [управляемом коде](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%8F%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4). Причём управляемый код — классы, реализующие [API](https://ru.wikipedia.org/wiki/API) для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, С++, так и на VB.Net, J# и др.

С одной стороны, Windows Forms рассматривается как замена более старой и сложной библиотеке [MFC](https://ru.wikipedia.org/wiki/Microsoft_Foundation_Classes), изначально написанной на языке [C++](https://ru.wikipedia.org/wiki/C%2B%2B). С другой стороны, WF не предлагает парадигму, сравнимую с [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller). Для исправления этой ситуации и реализации данной функциональности в WF существуют сторонние библиотеки. Одной из наиболее используемых подобных библиотек является [User Interface Process Application Block](http://www.microsoft.com/downloads/details.aspx?familyid=98C6CC9D-88E1-4490-8BD6-78092A0F084E&displaylang=en), выпущенная специальной группой Microsoft, занимающейся примерами и рекомендациями, для бесплатного скачивания. Эта библиотека также содержит исходный код и обучающие примеры для ускорения обучения.

Внутри .NET Framework, Windows Forms реализуется в рамках [пространства имён](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%B8%D0%BC%D1%91%D0%BD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) System.Windows.Forms.

Windows Forms предоставляет широкий спектр элементов, которые условно можно разделить на два типа: интерфейсные – те, которые видны пользователю и с которыми он может работать непосредственно (разного рода кнопки, панели, таблицы) и служебные – те, что выполняют определенные задачи и вызываются путем взаимодействия пользователя с интерфейсными элементами (диалоги, таймеры, адаптеры).

Одно из преимуществ Windows Forms - в том, что на нем можно писать кросс-платформенные приложения. Простые проекты, написанные на Windows Forms, можно довольно легко перенести на другую операционную систему, если на ней установлен .NetFramework нужной модели, на котором написан Ваш проект. Выглядит это действительно впечатляюще, но на практике очень часто проект не будет работать. Особенно при использовании посторонних контролов, написанных другой компанией.

1. **Зачем используется класс Form? Назовите основные методы, свойства и события данного класса.**

[Form](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.form?view=netcore-3.0) — это представление любого окна, отображаемого в приложении. Класс [Form](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.form?view=netcore-3.0) можно использовать для создания стандартных окон, инструментов, без границ и плавающих окна. Можно также использовать класс [Form](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.form?view=netcore-3.0) для создания модальных окон, таких как диалоговое окно.

Используя свойства, доступные в классе [Form](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.form?view=netcore-3.0), можно определить внешний вид, размер, цвет и функции управления окнами создаваемого окна или диалогового окна. Свойство [Text](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.text?view=netcore-3.0) позволяет указать заголовок окна в строке заголовка. Свойства [Size](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.form.size?view=netcore-3.0) и [DesktopLocation](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.form.desktoplocation?view=netcore-3.0) позволяют определить размер и расположение окна при его отображении. Свойство Цвет [ForeColor](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.forecolor?view=netcore-3.0) можно использовать для изменения цвета переднего плана по умолчанию для всех элементов управления, размещенных в форме. Свойства [FormBorderStyle](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.form.formborderstyle?view=netcore-3.0), [MinimizeBox](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.form.minimizebox?view=netcore-3.0)и [MaximizeBox](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.form.maximizebox?view=netcore-3.0) позволяют управлять возможностью сворачивания, разворачиванием или изменением размера формы во время выполнения.

Для взаимодействия с пользователем в Windows Forms используется механизм событий. События в Windows Forms представляют стандартные события на C#, только применяемые к визуальным компонентам и подчиняются тем же правилам, что события в C#. Но создание обработчиков событий в Windows Forms все же имеет некоторые особенности.

Прежде всего в WinForms есть некоторый стандартный набор событий, который по большей части имеется у всех визуальных компонентов. Отдельные элементы добавляют свои события, но принципы работы с ними будут похожие. Чтобы посмотреть все события элемента, нам надо выбрать этот элемент в поле графического дизайнера и перейти к вкладке событий на панели форм.

1. **Поясните структуру проекта и назначение всех файлов?**



**4. Зачем нужен атрибут STAThreadAttribute?**

STA-single-threaded apartment -означает, что все потоки в этой программе выполняются в рамках одного процесса, а управление программой осуществляется одним главным потоком.

Инициализирует новый экземпляр класса System.STAThreadAttribute.

1. **Как в вашем проекте используются события и делегаты?**

Дело в том, что среда .NET содержит огромное количество встроенных событий, доступ к которым осуществляется по их названиям. Более того, среда сама привязывает обработчика для события Click (нажатие на кнопку) и нужный метод, используя встроенный делегат EventHandler:

this.button1.Click += new System.EventHandler(this.button1\_Click);

Обработчики событий не могут возвращать ничего, кроме void. В них отсутствует точка, которая могла бы служить для возврата значения. Обработчики должны принимать два параметра. Первый параметр является ссылкой на объект, который сгенерировал событие. Второй параметр должен быть ссылкой либо на базовый класс .NET System.EventArgs, либо на производный класс. КлассEventArgs представляет собой общий базовый класс для всех уведомлений о произошедших событиях.

1. **Объясните схему работы цепочек делегатов.**

Делегат может ссылаться на группу методов, которые связаны между собой в цепочку.**Групповая адресация** – это способ (возможность) создания списка методов, которые вызываются автоматически при обращении к делегату.

Делегаты, включающие в себя более одного метода, называются мультикаст-делегатами. При вызове они выполняют каждый метод в заданном порядке, позволяя таким образом связывать несколько методов в цепочку.

Для работы мультикаст-делегатов те не должны возвращать какой-либо результат. В противном случае обработается результат последнего метода цепочки.

При вызове мультикаст-делегатов каждый экземпляр в цепочке вызывается последовательно. Однако это может обернуться проблемой, если в одном из экземпляров сработало исключение или он вернул результат не void-типа.

Групповая адресация позволяет сформировать списки (цепочки) вызовов. Это есть эффективным, поскольку:

* можно гибко формировать списки методов, которые должны вызываться по одному вызову делегата;
* удобно формировать списки методов, которые выполняют разные виды работ над некоторым общим объектом;
* улучшается структура программного кода;
* удобно обрабатывать события, которые генерируются системой.

##### 6. Как работает групповая адресация, если делегат возвращает значение?

* Если метод возвращает значение, тогда будет вызван **последний метод** в списке (цепочке) методов.

##### 7. Как работает групповая адресация, если делегат возвращает тип void?

* Если делегат возвращает тип void, то при вызове делегата будут **вызываться все методы**, которые сформированы в список (цепочку) с помощью групповой адресации.
* Последовательность вызова методов соответствует последовательности добавления методов к списку. Первый добавленный метод будет вызван первым, второй добавленный метод будет вызван вторым и т.д.

**7. Объясните механизм подписки и отмены подписки на события.**

Добавление событий

1) двойной щелчок на ЭУ – происходит обращение к обработчику события, используемого по умолчанию

2) использование списка событий в окне Properties (Свойства) – дважды щелкнуть на этом событии в списке событий - генерация кода подписки ЭУ на событие и сигнатура метода обработки этого события.

3) Добавление кода подписки на событие вручную - подписка на событие (т.е. связывания с ним) и создание соответствующей сигнатуры метода обработчика

**Отмена подписки на событие**

* Чтобы отменить подписку на событие, воспользуйтесь оператором присваивания вычитания (-=).

**8. Как создать вторую форму и передать туда данные? Есть ли другие способы?**

Чтобы добавить еще одну форму в проект, нажмем на имя проекта в окне Solution Explorer (Обозреватель решений) правой кнопкой мыши и выберем Add(Добавить)->Windows Form...

9. **Как во время выполнения приложения добавить/удалить контрол?**

Распространенные задачи при разработке приложений — это добавление элементов управления и удаление элементов управления из любого контейнера в формах (например [Panel](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.panel) , или [GroupBox](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.groupbox) , или даже самой формы). Во время разработки элементы управления можно перетаскивать непосредственно на панель или в группу. Во время выполнения эти элементы управления поддерживают коллекцию Controls, которая отслеживает размещенные в них элементы управления.

**Программное добавление элемента управления в коллекцию**

1. Создайте экземпляр элемента управления, подлежащий добавлению.
2. Задайте свойства нового элемента управления.
3. Добавьте этот элемент управления в коллекцию Controls родительского элемента управления.

**Программное удаление элементов управления из коллекции**

1. Удалите обработчик событий из события. В Visual Basic используйте ключевое слово оператора RemoveHandler . в C# используйте оператор -=.
2. Используйте метод Remove для удаления требуемого элемента управления из коллекции Controls панели.
3. Вызовите Dispose метод, чтобы освободить все ресурсы, используемые элементом управления.