1. **Что такое производный и базовый классы?**

В языке C# класс, который наследуется, называется ***базовым***, а класс, который наследует, — ***производным***. Следовательно, производный класс представляет собой специализированный вариант базового класса. Он наследует все переменные, методы, свойства и индексаторы, определяемые в базовом классе, добавляя к ним свои собственные элементы.

1. **В чем заключена основная задача наследования?**

Базовая идея, лежащая в основе классического наследования, заключается в том, что новые классы могут создаваться с использованием существующих классов в качестве отправной точки

1. **Пусть базовый класс содержит метод basefunc(), а производный класс не имеет метода с таким именем. Может ли объект производного класса иметь доступ к методу basefunc()? Если да, то при каких условиях?**

Если у basefunc() модификатор доступа **не** private

1. **Напишите объявление конструктора без аргументов для производного класса B, который будет вызывать конструктор без аргументов базового класса A.**

Public B():base()

{

}

1. **Что такое полиморфизм? Приведите пример.**

**полиморфизм** – это различная реализация однотипных действий. Классическая фраза, которая коротко объясняет полиморфизм – «Один интерфейс, множество реализаций».

1. **Определите назначение виртуальных функций.**

Те методы и свойства, которые мы хотим сделать доступными для переопределения, в базовом классе помечается модификатором **virtual**. Такие методы и свойства называют виртуальными.

1. **Кому доступны переменные с модификатором protected?**

Переменные с таким модификатором доступны внутри класса, а также его наследником, в том числе и наследникам в других сборках.

1. **Наследуются ли переменные с модификатором private?**

Да

1. **As, is – что это, как применяется?**

С помощью этих операторов происходит проверка возможности приведения объекта к другому типу. Is возвращает значение bool. As, если возможно преобразование, возвращает преобразованный объект, если преобразование невозможно – возвращает null.

1. **Поддерживает ли C# множественное наследование?**

Множественное наследование классов не поддерживается, но имеется возможность реализовать в классе несколько интерфейсов.

1. **Можно ли запретить наследование от класса?**

Да, с помощью модификатора sealed.

1. **Можно ли разрешить наследование класса, но запретить перекрытие метода?**

Да, с помощью модификатора sealed

1. **Что такое абстрактный класс?**

Базовый класс, который не предполагает создания экземпляров

1. **В каком случае вы обязаны объявить класс абстрактным?**
2. В том случае, если класс является наследником абстрактного класса, но не все методы базового класса перекрыты и имеют реализацию.

2. В том случае, если хотя бы один метод класса является абстрактным.

1. **В чем разница между абстрактными и виртуальными классами? Между виртуальными и абстрактными методами?**

Абстрактный класс это класс, содержащий хотя бы один метод (abstract)..  
Виртуальный метод имеет реализацию и м.б. переопределен в производном классе. Абстрактный метод не имеет реализацию, только описание метода, который д.б. реализован в производных классах.

1. **Какие компоненты класса могут быть виртуальными?**

Виртуальными могут быть:

* Методы
* Свойства
* Индексаторы
* События

1. **Что такое интерфейс?**

Интерфейсы позволяют определить некоторый функционал, не имеющий конкретной реализации. Затем этот функционал реализуют классы, применяющие данные интерфейсы.

1. **Как работать с объектом через унаследованный интерфейс?**

Можно создать экземпляр класса с типом интерфейса. В этом случае такому объекту будут доступны только те методы класса, которые были объявлены в интерфейсе.

1. **Приведите пример явной реализации интерфейса.**

Interface IDno

{

Void WriteCode();

}

class Programmer : IDno

{

Public void WriteCode()

{

Console.Writeline(“Пишу код так скозать”)}}

1. **Почему нельзя указать модификатор видимости для методов интерфейса?**

Еще один момент в объявлении интерфейса: все его члены - методы и свойства не имеют модификаторов доступа, но фактически по умолчанию доступ **public**, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации.

1. **Можно ли наследовать от нескольких интерфейсов?**

Да

1. **Назовите отличия между интерфейсом и абстрактным классом.**

В интерфейсе все методы (свойства и т.д.) абстрактны и не имеют реализации. В абстрактном классе некоторые методы могут быть реализованы. В интерфейсе члены не могут иметь модификатора видимости (все они являются public по умолчанию), а в абстрактном классе члены могут иметь модификатор видимости.

1. **Для чего используются стандартные интерфейсы ICloneable, IComparable, IComparer, lEnumerable?**

**Сравнение объектов (интерфейс IComparable)**

Интерфейс IComparable содержит всего один метод **CompareTo(),**возвращающий результат сравнения двух объектов – текущего и переданного ему в качеству параметра:

interface IComparable

{

int CompareTo(object obj)

}

Метод должен возвращать:

0, если текущий объект и параметр равны

отрицательное число, если текущий объект меньше параметра

положительное число, если текущий объект больше параметра

**Сортировка объектов по различным критериям (интерфейс IComparer)**

Данный интерфейс определен в пространстве имен System.Collections. Он также содержит один метод **Compare(),**возвращающий результат сравнения двух объектов, переданных ему в качестве параметров.

interface IComparer

{

int Compare(object obj1,object obj2)

}

Принцип применения этого интерфейса состоит в том, что для каждого критерия сортировки объектов описывается небольшой вспомогательный класс, реализующий этот интерфейс. Объект этого класса передается в стандартный метод сортировки массива в качестве второго аргумента.

Модернизируем предыдущий пример и отсортируем массив экземпляров класса Monster по двум критериям – по именам (свойство Имя, класс SortByИмя) и умениям (свойство Умение, класс SortByУмение). Классы критериев сортировки объявлены вложенными, так как требуются только экземплярам класса Monster.

**Клонирование объектов (интерфейс ICloneable)**

**Клонирование**-создание копии объекта. Копия объекта называется **клоном.** При присваивании одного экземпляра другому копируется ссылка, а не сам объект. Если необходимо скопировать в другую область памяти поля объекта, можно воспользоваться методом **MemberwiseClone(),** который любой объект наследует от класса object. При этом объекты, на которые указывают поля объекта, в свою очередь являющиеся ссылками, не копируются. Это называется ***поверхностным клонированием***. Для создания полностью независимых объектов необходимо ***глубокое копирование****,* когда в памяти создается дубликат всего дерева объектов, то есть объектов, на которые ссылаются поля объекта, поля полей, и т.д. Алгоритм глубокого копирования сложен, требует рекурсивного обхода всех ссылок объекта и отслеживания циклических зависимостей.

Объект, имеющий собственные алгоритмы клонирования, должен объявляться как производный интерфейса ICloneable и переопределять его единственный метод Clone().

**Перебор элементов (интерфейс IEnumerable) и итераторы**

Оператор foreach является удобным средством для перебора эелемнтов объекта. Массивы и все коллекции позволяют выполнять такой перебор благодаря тому, что в них реализованы интерфейсы IEnumerable и IEnumerator. Для применения оператора foreach к собсвенному типу данных требуется реализовать в нем эти итераторы.

Интерфейс IEnumerable (перечислимый) определяет единственный метод GetEnumerator(), возвращающий объект типа IEnumerator (перечислитель), который можно использовать для просмотра элементов объекта.

Интерфейс IEnumerator задает три элемента:

* Свойство Current, возвращающее текущий элемент объекта
* Метод MoveNext() продвигающий перечислитель на следующий элемент объекта
* Метод Reset() устанавливающий перечислитель в начало просмотра.

Цикл foreach использует эти методы для перебора элементов, из которых состоит объект.

Таким образом, для того, чтобы применить цикл foreach необходимо реализовать четыре метода GetEnumerator(), Current, MoveNext() и Reset().

Например, если элементы класса организованы как массив, необходимо описать закрытое поле класса, хранящее текущий индекс в массиве, в методе MoveNext() задать изменение этого индекса на 1 и проверку выхода за границу массива, а в свойстве Current – возврат элемента массива по текущему индексу.

В языке С# имеются стандартные средства, обеспечивающие аналогичные действия, которые выполняют перебор элементов в объекте – итераторы.

1. **В какой строке приведенного ниже фрагмента листинга не содержится ошибки?**

2,5

1, т.к. невозможно создать тело у абстрактного метода

3, т.к. модификатор new повторяется

4, т.к. статический метод не может быть помечен как virtual

1. **Что будет выведено на консоль в результате выполнения следующего фрагмента?**

3 4

1. **Что будет выведено на консоль в результате выполнения следующего фрагмента кода?**

A

B

1. **Чем может быть M4 если дано следующее определение?**

Только интерфейсом

1. **Выберите верное присваивание для объектов, определенных в листинге.**

Варианты ответа:

1) b = a; //не удаётся неявно преобразовать

2) a = b;

3) c = a;

4) d = a;

5) с = b;

1. **Что будет выведено на консоль в результате выполнения следующего фрагмента, если раскомментировать строчку 1?**

AB

1. **В какой строке приведенного ниже фрагмента листинга содержится ошибка?**

2, т.к. член помеченный как override не может быть помечен как new или virtual

1. **Почему приведенный ниже фрагмент листинга содержит ошибку?**

Нельзя создать экземпляр абстрактного класса.

1. **В какой строке может быть ошибка компиляции?**

Ошибки нет

1. **Что будет выведено на консоль в результате выполнения следующего фрагмента листинга?**

New G