**Вопросы**

**2. Пусть дан фрагмент листинга. В какой строчке содержится ошибка?**

**6. Какое ограничение на тип задано в следующем фрагменте листинга?** class A { } class В : A { } class С { } class Test where T : A { }

**7. Какое ограничение на тип задано в следующем фрагменте листинга?** interface A { } class Test where T : class { }

**8. Какое ограничение на тип задано в следующем фрагменте листинга?** interface A { } class Test where T : struct { }

**1. Что такое обобщение (generic)?**

Термин обобщение означает параметризированный тип.

**3. Как можно наложить определенное ограничение на параметр?**

С помощью ключевого слова where T : (class, struct, name\_class, name\_class<T>, new() – это что-то, что имеет конструктор без параметров).

**4. Как можно наложить несколько ограничений на параметр?**

Через ,

class Folf<T> where T: class, new()

**5. Перечислите все существующие ограничения на типы данных обобщения?**

class, struct, name\_class, name\_class<T>, new() – это что-то, что имеет конструктор без параметров

**9. Приведите примеры, когда обобщенный класс может действовать как базовый или производный класс.**

Class Dog<T, V>: Folf<T> (Folf<T> - базовый)

T – параметр передаваемый базовому классу

V – параметр характерный только для производного класса0

Class Folf<T> : Animal (Folf<T> - производный)

**10. В каких случаях в обобщениях может использоваться оператор default?**

Когда в обобщении нужно присвоить переменным некоторое начальное значение, то на прямую это сделать нельзя, поэтому используется ключевое слово default(T)

Он присваивает ссылочным типам значение null, а значимым типам 0.

**11. Поясните как использовать статические переменные в обобщенных классах.**

В обобщённых классах определённым образом создаётся набор статических членов и его можно инициализировать без создания объекта

Folf<T>

{

Public static T speed;

}

Folf<string>.speed = “fast”;

Folf<int>.speed = 50;

**12. Приведите пример обобщенного интерфейса.**

Interface IGen<T>

{

Void Show();

Void Add();

}

**13. В чем отличие обобщенных классов от обобщенных структур?**

Они похожи но структуры нельзя наследовать.

**14. Какие классы для работы с файлами вы знаете? Приведите пример**

File, fileInfo

Вопросы

Делегату без параметров соответствуют все методы, у которых также нету параметров при вызове. Если делегат имеет возвращающие значение void, то указывать он тоже будет на методы у которых стоит void.

События сигнализируют системе что произошло некоторое действие. Обработчик событий должен соответствовать делегату, который представляет событие. При генерации события можем передать во внешний код некоторые значения, как правило эти значения оформляются в один объект. Для представления такого объекта определяют новый класс.

Использование событий:

Имя\_объекта.имя\_события += (обработчик\_событий) можем добавить непосредственно делегат, в конструктор которого нужно передать ссылку на метод. new имя\_делегата(ссылка\_на\_метод);

Или можем сократить: Имя\_объекта.имя\_события += ссылка\_на\_метод;

**1. Что такое делегат? Как определить делегат?**

Делегат — это объект(тип), предназначенный для хранения ссылок на методы; объект – с определенным списком параметров и типом возвращаемого значения;

Делегаты не содержат реализации. Определяется с помощью ключевого слова delegate.

**2. Назначение делегатов**

Делегаты представляют такие объекты, которые указывают на методы (указатели на методы). Служат для позднего связывания и групповой адресации методов(цепочки). Используются для поддержки событий.

1) возможности определять вызываемый метод не при компиляции, а динамически во время выполнения программы;

2) обеспечения связи между объектами по типу «источник — наблюдатель»;

3) создания универсальных методов, в которые можно передавать другие методы;

4) поддержки механизма обратных вызовов.

**3. Какие есть способы присваивания делегату адреса метода?**

Способы передачи параметра делегату:

· Message mess = new Message(Display); // 1 способ – объявляем переменную делегата и передаём значения делегату через его конструктор

· mess = Display; //2 способ передачи значения переменной делегата

· mess = new Message(Display); //3 способ передачи значения в переменную через конструктор делегата

**4. Поясните назначение метода Invoke.**

· Делегат можно вызывать через метод delegate\_name.Invoke();

· Делегат можно вызывать как обычный метод delegate\_name();

**5. Что такое групповая адресация делегата?**

Создание списка или цепочки вызовов, для методов, которые вызываются автоматически при обращении к делегату.

Delegate void OperWithArr(ref int[] arr);

public class ArrayOperation

{

public static void WriteArray(ref int[] arr) {...}

public static void IncSort(ref int[] arr){...} //Сортировка массива

public static void NegatArr(ref int[] arr){...} //Заменяем отрицательные числа на ноль

}

Int [] somearr = new int[] {1, 5, 8, 13, 15, 21};

OperationWithArray DelegAll; // Групповая адресация

DelegAll = ArrayOperation.WriteArray;

DelegAll += ArrayOperation.IncSort;

DelegAll += ArrayOperation.WriteArray;

DelegAll += ArrayOperation.NegatArr;

DelegAll += ArrayOperation.WriteArray;

**6. Как создать событие?**

Для определения событий используется ключевое слово event.

Public event имя\_делегата(который представляет данное событие) имя\_события

**7. Как события связаны с делегатами? Опишите и поясните схему взаимодействия.**

События построены на основе делегатов: с помощью делегатов вызываются методы-обработчики событий. Поэтому создание события в классе состоит из следующих частей:

· описание делегата, задающего сигнатуру обработчиков событий;

· описание события;

· описание метода (методов), инициирующих событие.

**8. Что такое лямбда-выражения? Приведите пример лямбда-выражения с несколькими параметрами.**

Упрощенная запись анонимных методов

параметр => выражение

(список\_параметров) => выражение

(x, y) => x + y;

**9. Что такое ковариантность и контравариантность делегатов? Что это дает?**

Ковариантность позволяет присвоить делегату метод, возвращаемым типом которого служит класс, производный от класса, указываемого в возвращаемом типе делегата

delegate Person PersonFactory(string name);

PersonFactory personDel;

personDel = BuildClient; // ковариантность

Контравариантность позволяет присвоить делегату метод, типом параметра которого служит класс, являющийся базовым для класса, указываемого в объявлении делегата.

delegate void ClientInfo(Client client);

ClientInfo clientInfo = GetPersonInfo; // контравариантность

Как и в случае с обобщенными интерфейсами параметр ковариантного типа применяется только к типу значения, которые возвращается делегатом. А параметр контравариантностного типа применяется только к входным аргументам делегата.

**10. Поясните разницу между встроенными делегатами Action и Func.**

Делегат Action является обобщенным, принимает параметры и возвращает значение void:

от Action<in T1> Как правило, этот делегат передается в качестве параметра метода и предусматривает вызов определенных действий в ответ на произошедшие действия.

Func (Имеет тип возврата) - он возвращает результат действия и может принимать параметры, а также имеет различные формы: от Func<out T>(), где T - тип возвращаемого значения, до Func<in T1, in T2,...in T16, out TResult>(), то есть может принимать до 16 параметров.