### 1. ****На какие основные виды/типы делятся все коллекции .NET? Охарактеризуйте каждый из них.****

#### ****Необобщенные коллекции (****System.Collections****)****

* Не используют параметры типов (обобщения).
* Требуют явного приведения типов, что увеличивает вероятность ошибок и снижает производительность.
* Примеры: ArrayList, Hashtable.  
  **Плюсы:** Простота использования в небольших проектах.  
  **Минусы:** Меньшая безопасность типов и производительность.

#### ****Обобщенные коллекции (****System.Collections.Generic****)****

* Поддерживают параметризацию типами, что улучшает безопасность типов и производительность.
* Примеры: List<T>, Dictionary<TKey, TValue>, HashSet<T>.  
  **Плюсы:** Безопасность типов, отсутствие необходимости приведения.  
  **Минусы:** Нельзя использовать в .NET Framework 1.0.

#### ****Специальные коллекции с поразрядной организацией (****System.Collections.Specialized****)****

* Предназначены для хранения данных, требующих нестандартных структур.
* Примеры: BitArray, StringCollection, NameValueCollection.  
  **Плюсы:** Оптимизированы под узкие задачи.  
  **Минусы:** Ограниченная применимость.

#### ****Параллельные коллекции (****System.Collections.Concurrent****)****

* Поддерживают многопоточность без необходимости блокировок.
* Примеры: ConcurrentBag<T>, ConcurrentDictionary<TKey, TValue>.  
  **Плюсы:** Высокая производительность в многопоточных приложениях.  
  **Минусы:** Могут быть сложнее в использовании.

### 2. ****Что такое generic-коллекции? Назовите примеры известных вам generic-коллекций.****

**Generic-коллекции** — это коллекции, поддерживающие параметризацию типами. Это позволяет избежать ошибок приведения типов и повысить производительность.  
**Примеры:**

* List<T> — список с доступом по индексу.
* Dictionary<TKey, TValue> — коллекция пар "ключ-значение".
* Queue<T> — очередь (FIFO).
* Stack<T> — стек (LIFO).
* HashSet<T> — уникальное множество.

### 3. ****В чем разница между**** ArrayList ****и**** Array****?****

Array**:**

* + Статический массив фиксированного размера.
  + Не является коллекцией, а базовой структурой данных.
  + Требует указания типа при создании.

ArrayList**:**

* + Динамическая коллекция, которая может изменять размер.
  + Хранит элементы в виде объектов (object).
  + Требует приведения типов при работе с данными.

### 4. ****Охарактеризуйте коллекции, которые вы использовали в своем варианте.****

* List<T>**:** Используется для хранения упорядоченных элементов, предоставляет доступ по индексу.
* Dictionary<TKey, TValue>**:** Подходит для работы с ассоциативными массивами (ключ-значение).
* Queue<T>**:** Реализует очередь (FIFO), применяется для обработки задач в последовательности.
* Stack<T>**:** Реализует стек (LIFO), полезен для задач обратного порядка.

### 5. ****Чем отличаются коллекции, расположенные в пространстве имен**** System.Collections.Concurrent****?****

* Поддерживают многопоточную работу без необходимости использовать lock.
* Используют внутренние механизмы синхронизации, такие как блокировки на уровне сегментов.
* Высокая производительность в задачах с одновременным доступом.  
  **Примеры:** ConcurrentBag<T>, ConcurrentQueue<T>, ConcurrentDictionary<TKey, TValue>.

### 6. ****Какое пространство имен необходимо подключить в проект, чтобы иметь возможность использовать generic-коллекции?****

Необходимо подключить пространство имен:

using System.Collections.Generic;

### 7. ****Что такое наблюдаемая коллекция? Как ее можно использовать?****

**Наблюдаемая коллекция (**ObservableCollection<T>**)** — коллекция, которая уведомляет об изменениях (добавление, удаление, изменение элементов).

**Используется:**

* В приложениях WPF и MVVM для реализации двухстороннего привязки данных.
* Для уведомления UI об изменениях в данных.

**Пример:**

ObservableCollection<string> names = new ObservableCollection<string>();

names.CollectionChanged += (sender, e) => Console.WriteLine("Collection changed!");

names.Add("Alice");

### 8. ****Охарактеризуйте интерфейсы**** IEnumerator****,**** IEnumerable****. В чем отличие назначений?****

IEnumerable**:**

* + Определяет метод GetEnumerator(), возвращающий перечислитель (IEnumerator).
  + Предоставляет возможность перебирать элементы коллекции с помощью foreach.

IEnumerator**:**

* + Используется для итерации по элементам коллекции.
  + Предоставляет методы MoveNext(), Reset() и свойство Current.

**Отличия:**  
IEnumerable — предоставляет доступ к перечислителю, а IEnumerator управляет самой итерацией.

### 9. ****Поясните принцип работы коллекций:****

#### ****a.**** LinkedList<T>

Двусвязный список. Каждый элемент хранит ссылку на предыдущий и следующий элемент.  
**Плюсы:** Быстрое добавление/удаление.  
**Минусы:** Медленный доступ по индексу.

#### ****b.**** HashSet<T>

Множество уникальных элементов, реализовано на основе хеш-таблицы.  
**Плюсы:** Быстрое добавление/поиск.  
**Минусы:** Не гарантирует порядок элементов.

#### ****c.**** Dictionary<TKey, TValue>

Коллекция пар "ключ-значение". Использует хеширование для быстрого доступа.  
**Плюсы:** Высокая производительность поиска.  
**Минусы:** Памятозатратен при больших объемах данных.

#### ****d.**** ConcurrentBag<T>

Параллельная коллекция для хранения объектов.  
**Плюсы:** Безопасность при многопоточном доступе.  
**Минусы:** Нет порядка добавления/извлечения.

#### ****e.**** Stack<T> ****и**** Queue<T>

* Stack<T> — реализует стек (LIFO). Пример: алгоритм обхода в глубину.
* Queue<T> — реализует очередь (FIFO). Пример: алгоритм обхода в ширину.

#### ****f.**** SortedList****,**** SortedDictionary

* SortedList — массив с сортировкой ключей.
* SortedDictionary — дерево с сортировкой ключей.  
  **Плюсы:** Быстрое извлечение упорядоченных данных.  
  **Минусы:** Меньшая производительность, чем у Dictionary.

### 1. ****Что такое LINQ?****

LINQ (Language Integrated Query) — это технология для работы с данными в .NET, которая позволяет писать запросы для обработки данных прямо на языке программирования, например, C#. LINQ предоставляет единый синтаксис для работы с различными источниками данных, такими как массивы, коллекции, базы данных, XML и другие.

### 2. ****В чем разница между отложенными операциями и неотложенными операциями LINQ to Object?****

**Отложенные операции (Deferred Execution):**  
Эти операции выполняются только тогда, когда вы фактически начинаете перебирать результаты (например, с помощью foreach или вызова методов, таких как ToList() или Count()). Примеры: Where, Select, Take.  
**Преимущество:** позволяет оптимизировать выполнение, так как запросы выполняются только при необходимости.

**Неотложенные операции (Immediate Execution):**  
Эти операции выполняются немедленно, и результат возвращается сразу. Примеры: Count, ToList, Sum.  
**Преимущество:** результат сразу готов к использованию, но может занимать больше памяти.

### 3. ****Что такое лямбда-выражения?****

Лямбда-выражение — это короткий способ записи анонимных функций. Они используются для передачи логики в методы LINQ.  
**Синтаксис:**

(parameter) => expression

**Пример:**

numbers.Where(x => x > 5);

Здесь x => x > 5 — лямбда-выражение, где x — параметр, а x > 5 — условие.

### 4. ****Какие есть группы операций в LINQ to Object?****

* **Фильтрация:** Where
* **Проекция:** Select, SelectMany
* **Сортировка:** OrderBy, OrderByDescending, ThenBy, ThenByDescending
* **Группировка:** GroupBy
* **Соединение:** Join, GroupJoin
* **Агрегация:** Count, Sum, Min, Max, Average
* **Элементы:** First, Last, ElementAt, Single
* **Объединение:** Union, Concat, Intersect, Except
* **Квантификаторы:** Any, All, Contains

### 5. ****Как используется операция**** Where ****в LINQ to Object?****

Операция Where используется для фильтрации элементов коллекции на основе условия.  
**Пример:**

var result = list.Where(x => x > 5);

Этот код возвращает все элементы list, которые больше 5.

### 6. ****Как используется операция**** Select****?****

Операция Select применяется для преобразования элементов коллекции.  
**Пример:**

var result = list.Select(x => x \* 2);

Этот код возвращает коллекцию, где каждый элемент умножен на 2.

### 7. ****Как используются операции**** Take ****и**** Skip****?****

* Take**:** берет указанное количество элементов с начала коллекции.
* Skip**:** пропускает указанное количество элементов и возвращает оставшиеся.  
  **Пример:**

var firstTwo = list.Take(2); // первые 2 элемента

var afterTwo = list.Skip(2); // все элементы после первых 2

### 8. ****Как используется операция**** Concat****?****

Операция Concat объединяет две коллекции.  
**Пример:**

var combined = list1.Concat(list2);

Этот код возвращает объединенную коллекцию list1 и list2.

### 9. ****Как используется операция**** OrderBy****?****

Операция OrderBy сортирует элементы коллекции в порядке возрастания.  
**Пример:**

var sorted = list.OrderBy(x => x);

Этот код сортирует list по значениям.

### 10. ****Как используется операция**** Join****?****

Операция Join соединяет две коллекции на основе общего ключа.  
**Пример:**

var result = people.Join(

departments,

person => person.DepartmentId,

department => department.Id,

(person, department) => new { person.Name, department.Name }

);

### 11. ****Как используются операции**** Distinct****,**** Union****,**** Except ****и**** Intersect****?****

* Distinct**:** удаляет дубликаты.
* Union**:** объединяет две коллекции, исключая дубликаты.
* Except**:** возвращает элементы первой коллекции, которых нет во второй.
* Intersect**:** возвращает общие элементы двух коллекций.

### 12. ****Как используются операции**** First****,**** Last****,**** Any****,**** All ****и**** Contains****?****

* First**:** возвращает первый элемент, соответствующий условию.
* Last**:** возвращает последний элемент, соответствующий условию.
* Any**:** проверяет, есть ли хотя бы один элемент, соответствующий условию.
* All**:** проверяет, соответствуют ли все элементы условию.
* Contains**:** проверяет, содержит ли коллекция заданный элемент.

### 13. ****Как используются операции**** Count****,**** Sum****,**** Min****,**** Max****,**** Average****?****

* Count**:** возвращает количество элементов.
* Sum**:** возвращает сумму элементов.
* Min**:** возвращает минимальное значение.
* Max**:** возвращает максимальное значение.
* Average**:** возвращает среднее значение элементов.

### 14. ****Что выведет на экран данный код?****

class Test

{

public static void Main()

{

List<int> list = new List<int>();

list.AddRange(new int[] { 3, 1, 4, 8, 10, 4 });

List<int> some = list.FindAll(i => (i >= 9));

foreach (int x in some)

Console.Write(x);

}

}

**Ответ:**  
На экран будет выведено:

10

Поскольку только элемент 10 из исходного списка удовлетворяет условию i >= 9.

### 1. ****Что такое рефлексия?****

**Рефлексия** — это механизм .NET, позволяющий программе исследовать свою структуру во время выполнения, а также изменять или вызывать ее элементы. С помощью рефлексии можно:

* Получать информацию о типах, методах, полях, свойствах и других элементах программы.
* Вызывать методы, создавать экземпляры объектов, читать и изменять значения полей или свойств.

Рефлексия используется для разработки гибких и динамичных приложений, таких как сериализация, тестирование, работа с плагинами и т.д.

### 2. ****Какие есть способы получения объекта типа**** Type****? Приведите три способа получения типа.****

**Через оператор** typeof**:**  
Используется для получения объекта Type для известного типа.

Type type = typeof(string); // Тип System.String

**Через метод** GetType**:**  
Возвращает объект Type для конкретного экземпляра объекта.

string str = "Hello";

Type type = str.GetType(); // Тип System.String

**Через метод** Assembly.GetType**:**  
Позволяет получить тип из сборки.

Type type = Assembly.GetExecutingAssembly().GetType("Namespace.ClassName");

### 3. ****Охарактеризуйте классы из пространства имен**** System.Reflection****.****

Пространство имен System.Reflection предоставляет классы для работы с рефлексией. Основные классы:

* Assembly**:** Для работы с сборками. Позволяет загружать сборки, получать список типов.
* Module**:** Представляет модули (файлы .dll или .exe) в сборке.
* Type**:** Используется для работы с типами (классами, структурами, интерфейсами).
* MethodInfo**:** Для работы с методами (получение информации, вызов).
* PropertyInfo**:** Для работы со свойствами (получение, установка значений).
* FieldInfo**:** Для работы с полями (получение, изменение значений).
* ConstructorInfo**:** Для работы с конструкторами (создание экземпляров классов).
* ParameterInfo**:** Для получения информации о параметрах методов.

### 4. ****Как можно использовать класс**** System.Type****? Перечислите его свойства и методы.****

**Класс** System.Type используется для получения информации о типах.

**Основные свойства:**

* Name — имя типа.
* Namespace — пространство имен.
* FullName — полное имя типа с пространством имен.
* BaseType — базовый тип.
* IsClass, IsValueType, IsEnum — проверка типа (класс, структура, перечисление).

**Основные методы:**

* GetMethods(), GetMethod(string name) — получение методов.
* GetProperties() — получение свойств.
* GetFields() — получение полей.
* GetConstructors() — получение конструкторов.
* GetInterfaces() — получение интерфейсов, реализованных типом.
* IsAssignableFrom(Type) — проверяет, можно ли присвоить объект указанного типа.

**Пример использования:**

Type type = typeof(string);

Console.WriteLine(type.Name); // "String"

Console.WriteLine(type.Namespace); // "System"

### 5. ****Что такое позднее и раннее связывание?****

**Раннее связывание (Early Binding):**  
Связывание методов, свойств и других членов класса происходит на этапе компиляции. Пример: вызов метода через явный объект.

MyClass obj = new MyClass();

obj.Method(); // Раннее связывание

**Позднее связывание (Late Binding):**  
Связывание методов, свойств и других членов класса происходит на этапе выполнения с помощью рефлексии. Пример: вызов метода через MethodInfo.

Type type = typeof(MyClass);

MethodInfo method = type.GetMethod("Method");

method.Invoke(obj, null); // Позднее связывание

### 6. ****Как динамически загрузить сборку в приложение?****

Для загрузки сборки используется класс Assembly и метод Load или LoadFrom.

**Пример:**

Assembly assembly = Assembly.LoadFrom("MyAssembly.dll");

Type type = assembly.GetType("Namespace.MyClass");

object obj = Activator.CreateInstance(type);

### 7. ****Что такое позднее (раннее) связывание?****

Ответ аналогичен пункту 5.

### 8. ****Для чего предназначены**** BindingFlags****? Какую комбинацию флагов необходимо использовать, чтобы иметь возможность получать приватные члены класса?****

BindingFlags — это перечисление, используемое для настройки поиска членов типа (методов, полей, свойств и т.д.) при использовании рефлексии.

**Основные флаги:**

* Public, NonPublic — для поиска публичных и непубличных (приватных) членов.
* Static, Instance — для поиска статических и нестатических членов.
* DeclaredOnly — для поиска только тех членов, которые объявлены непосредственно в данном типе.

**Для доступа к приватным членам:**  
Необходимо комбинировать флаги NonPublic и Instance (или Static для статических членов).

**Пример:**

Type type = typeof(MyClass);

FieldInfo privateField = type.GetField("privateField", BindingFlags.NonPublic | BindingFlags.Instance);

### 1. ****Какие классы содержатся в**** System.IO****?****

В пространстве имен System.IO содержатся классы для работы с файлами, каталогами, потоками, сжатием данных и файловыми путями. Основные классы:

**Работа с файлами и каталогами:**

* + File
  + FileInfo
  + Directory
  + DirectoryInfo
  + Path

**Работа с потоками:**

* + Stream
  + FileStream
  + MemoryStream
  + StreamReader
  + StreamWriter
  + BinaryReader
  + BinaryWriter

**Сжатие данных:**

* + GZipStream
  + DeflateStream

**Другие:**

* + DriveInfo — для получения информации о дисках.
  + FileSystemWatcher — для мониторинга изменений в файловой системе.

### 2. ****Для чего используются классы**** Directory ****и**** DirectoryInfo****? В чем отличие?****

Directory — предоставляет статические методы для работы с каталогами (создание, удаление, проверка существования). Пример:

Directory.CreateDirectory("MyDirectory");

bool exists = Directory.Exists("MyDirectory");

Directory.Delete("MyDirectory");

DirectoryInfo — предоставляет аналогичный функционал, но с помощью экземплярных методов. Пример:

DirectoryInfo dir = new DirectoryInfo("MyDirectory");

dir.Create();

bool exists = dir.Exists;

dir.Delete();

**Основное отличие**:  
Directory работает только с путями, переданными как строки, а DirectoryInfo позволяет работать с объектами, представляющими каталоги.

### 3. ****Для чего используются классы**** File ****и**** FileInfo****? Какие методы они содержат?****

File — предоставляет статические методы для работы с файлами (создание, удаление, копирование, чтение, запись). Пример методов:

* + File.Create()
  + File.Delete()
  + File.Copy()
  + File.ReadAllText()
  + File.WriteAllText()

FileInfo — предоставляет аналогичный функционал, но с помощью экземплярных методов. Пример методов:

* + Create()
  + Delete()
  + CopyTo()
  + MoveTo()

**Различие**:  
File работает только со строковыми путями, а FileInfo — с объектами, которые представляют конкретный файл.

### 4. ****Для чего используются классы**** StreamReader ****и**** StreamWriter****?****

StreamReader — используется для чтения текстовых данных из потока. Пример:

using (StreamReader reader = new StreamReader("file.txt"))

{

string content = reader.ReadToEnd();

}

StreamWriter — используется для записи текстовых данных в поток. Пример:

using (StreamWriter writer = new StreamWriter("file.txt"))

{

writer.WriteLine("Hello, World!");

}

### 5. ****Для чего используются классы**** BinaryWriter ****и**** BinaryReader****?****

BinaryWriter — записывает данные в поток в двоичном формате. Пример:

using (BinaryWriter writer = new BinaryWriter(File.Open("file.bin", FileMode.Create)))

{

writer.Write(42); // Запись числа

writer.Write("Hello"); // Запись строки

}

BinaryReader — читает данные из потока в двоичном формате. Пример:

using (BinaryReader reader = new BinaryReader(File.Open("file.bin", FileMode.Open)))

{

int number = reader.ReadInt32();

string text = reader.ReadString();

}

### 6. ****Как можно сжимать и восстанавливать файлы?****

Для сжатия и восстановления файлов используются классы из пространства имен System.IO.Compression, например:

* GZipStream
* DeflateStream
* ZipFile

Пример сжатия:

using (FileStream sourceFile = File.OpenRead("source.txt"))

using (FileStream compressedFile = File.Create("source.gz"))

using (GZipStream compressionStream = new GZipStream(compressedFile, CompressionMode.Compress))

{

sourceFile.CopyTo(compressionStream);

}

Пример восстановления:

using (FileStream compressedFile = File.OpenRead("source.gz"))

using (FileStream decompressedFile = File.Create("decompressed.txt"))

using (GZipStream decompressionStream = new GZipStream(compressedFile, CompressionMode.Decompress))

{

decompressionStream.CopyTo(decompressedFile);

}

### 7. ****Алгоритм сжатия GZip****

GZip основан на алгоритме сжатия DEFLATE, который объединяет:

* **Алгоритм LZ77**: поиск повторяющихся блоков данных.
* **Кодирование Хаффмана**: уменьшение размера данных с помощью кодов переменной длины.

### 8. ****Для чего служит класс**** Path****?****

Класс Path предоставляет методы для работы с путями файлов и каталогов, такие как:

* Path.GetExtension() — получить расширение файла.
* Path.Combine() — объединить части пути.
* Path.GetFileName() — получить имя файла.
* Path.GetDirectoryName() — получить имя каталога.

### 9. ****Текстовый и бинарный файл. Как выполнить чтение и запись?****

**Текстовый файл**: хранит данные в виде текста. Чтение/запись:

File.WriteAllText("file.txt", "Hello, World!");

string content = File.ReadAllText("file.txt");

**Бинарный файл**: хранит данные в двоичном формате. Чтение/запись:

using (BinaryWriter writer = new BinaryWriter(File.Open("file.bin", FileMode.Create)))

{

writer.Write(42);

}

using (BinaryReader reader = new BinaryReader(File.Open("file.bin", FileMode.Open)))

{

int value = reader.ReadInt32();

}

### 10. ****Произвольный доступ к файлу. Пример.****

Произвольный доступ позволяет читать и записывать данные в произвольное место файла. Это возможно с помощью класса FileStream:

Пример:

using (FileStream fs = new FileStream("file.bin", FileMode.OpenOrCreate))

{

fs.Seek(10, SeekOrigin.Begin); // Перемещаемся на 10-й байт

fs.WriteByte(42); // Записываем один байт

}

### 11. ****Как применяется конструкция**** using ****при работе с файловыми потоками?****

using автоматически закрывает поток после завершения работы, даже если произошла ошибка. Это гарантирует освобождение ресурсов.

Пример:

using (StreamWriter writer = new StreamWriter("file.txt"))

{

writer.WriteLine("Hello, World!");

}

**Преимущества**:

* Безопасное управление ресурсами.
* Уменьшение утечек памяти.