#### Questions for this assignment

What are static classes in C# and when would you use them?

Explain real world scenarios where static classes are useful

What are the restrictions on using static classes in C#?

What is the difference between normal class and static class in C#

What are partial classes in C# and when would you use them?

Explain real world scenarios where partial classes are useful

What are partial methods in C# and when would you use them?

Explain real world scenarios where partial methods are useful

Can partial methods have implementation in more than one part of a partial class? If not, why?

What are static classes in C# and when would you use them?

Static classes in C# are classes that cannot be instantiated and can only contain static members, such as static fields, static properties, and static methods. They are commonly used when you want to provide utility functions or shared resources that can be accessed without creating an instance of the class.

Static classes are initialized only once when the type is accessed for the first time, and the instance remains in memory for the lifetime of the application domain (as long as the application runs). Static classes are useful in scenarios where you need to provide global access to common functions or resources, such as logging, configuration management, or mathematical calculations.

Explain real world scenarios where static classes are useful

Static classes in C# are useful in various real-world scenarios where you want to provide utility methods, shared functionality, or global access points without needing to instantiate objects of that class. Here are some examples:

**Utility methods:** Static classes can be used to define utility methods that perform common operations, such as mathematical calculations, string manipulations, or date/time formatting. For example, a MathUtility static class could contain methods like CalculateSquareRoot(), CalculateAverage(), or ConvertToRadians() that can be used across different parts of the codebase without needing to create instances of a class.

**Helper classes:** Static classes can be used to provide helper methods or functions that assist in performing specific tasks. For example, a FileUtility static class could provide methods like ReadFromFile(), WriteToFile(), or DeleteFile() that can be used to perform file operations in a consistent and reusable manner.

**Global access points:** Static classes can be used as global access points for resources or configurations that need to be shared across different parts of an application. For example, a Configuration static class could store global settings like database connection strings, API endpoints, or application settings that can be accessed from anywhere in the codebase without needing to pass them around as parameters or create instances of a class.

**State management:** Static classes can be used to manage shared state or data that needs to be accessed or modified by different parts of an application. For example, a SessionManager static class could store session data for a web application, or a CacheManager static class could provide methods to store and retrieve data from a shared cache. It is mainly useful in desktop applications but not on Web applications (because web applications run on request / response pattern).

**Performance optimizations:** Static classes can be used in performance-critical scenarios where the overhead of creating and managing instances of objects is not desirable. For example, a Logger static class could provide logging functionality in a high-traffic application, where creating a new object for every log entry may result in unnecessary memory overhead.

**Extension methods:** Static classes can be used to define extension methods that provide additional functionality to existing types without modifying their source code. For example, C# provides static classes like System.Linq.Enumerable and System.String that define extension methods to perform LINQ operations or string manipulations respectively.

In summary, static classes in C# are useful in real-world scenarios where you need to provide utility methods, helper classes, global access points, state management, performance optimizations, or extension methods. They provide a convenient way to encapsulate functionality that can be shared across different parts of an application without needing to create instances of a class.

What are the restrictions on using static classes in C#?

Static classes in C# have several restrictions:

1. Static classes cannot be instantiated, and no instance members, such as instance fields or instance methods, can be defined in them.
2. Static classes can only contain static members, such as static fields, static properties, and static methods.
3. Static members in a static class cannot be overridden or declared as abstract or virtual.
4. Static classes cannot be used as a base class or inherited from.
5. Static classes are sealed by default, meaning they cannot be inherited.
6. Static classes cannot have an instance constructor, as they cannot be instantiated.

What is the difference between normal class and static class in C#

In C#, there are several differences between a normal class and a static class:

* **Instantiation:** A normal class can be instantiated, meaning you can create objects of that class using the new keyword, and each object can have its own state (i.e., data members). On the other hand, a static class cannot be instantiated, and you cannot create objects of a static class using the new keyword. Instead, a static class can only contain static members, such as static fields, static properties, and static methods, that are shared across all instances of the class.
* **Inheritance:** A normal class can be inherited by other classes, and it can be used as a base class for deriving new classes. On the other hand, a static class cannot be inherited by other classes, and it cannot be used as a base class for deriving new classes.
* **Instance vs. Type members:** In a normal class, you can define both instance members (fields, properties, methods) and type members (static fields, properties, methods). Instance members are associated with objects of the class and can be accessed through object references, whereas type members are associated with the type itself and can be accessed through the type name. In a static class, you can only define static members (fields, properties, methods) that are associated with the type itself and can be accessed through the type name.
* **Initialization:** For a normal class, objects of the class need to be instantiated before their members can be accessed or modified. However, for a static class, static members are initialized automatically when the type is accessed for the first time, and they are shared across all instances of the class.
* **Memory usage:**Objects of a normal class consume memory for their instance members, and each object has its own memory footprint. On the other hand, a static class does not have any instance members and does not consume memory for object instances. Instead, static members in a static class are stored in the memory as part of the type itself and are shared across all instances of the class.
* **Usage:** A normal class is typically used when you need to create multiple instances of the class with their own state (i.e., data members), and you want to encapsulate behavior (i.e., methods) related to that state. On the other hand, a static class is typically used when you want to provide utility methods, constants, or configuration settings that are not associated with object instances, but rather with the type itself and are shared across all instances of the class.

In summary, the main differences between a normal class and a static class in C# are related to instantiation, inheritance, instance vs. type members, initialization, memory usage, and usage scenarios.

What are partial classes in C# and when would you use them?

Partial classes in C# allow a class to be split across multiple files, where each part can contain its own members, such as properties, methods, and events. They are combined at compile-time to form a single class. Partial classes are commonly used in scenarios where code generation tools or multiple developers need to work on different parts of a class simultaneously, but need to combine their work into a single class definition. For example, in a large-scale application with complex business logic, one team can work on the data access layer, while another team can work on the user interface layer, and both parts can be combined using partial classes.

Explain real world scenarios where partial classes are useful

Partial classes in C# are useful in various real-world scenarios where you want to split the definition of a class across multiple files. Here are some examples:

* **Code generation:** Partial classes are commonly used in code generation scenarios where a portion of the class implementation is generated automatically, and another portion is written by a developer. For example, if you are using a tool or framework that generates code, such as a UI designer or an ORM (Object-Relational Mapping) tool, it may generate a portion of the class definition, and you can extend it by implementing the remaining portion in a separate partial class file. This allows you to modify the generated code without losing your changes when the code is regenerated.
* **Separation of concerns:** Partial classes can be used to separate different concerns or aspects of a class into separate files. For example, a large class with multiple responsibilities, such as a Windows Form or a web page, may have separate partial class files for handling UI events, data access, business logic, or validation. This can make the codebase more organized and maintainable by keeping related code in separate files, rather than having one large monolithic class.
* **Collaboration between developers:** Partial classes can be used to enable multiple developers to work on different parts of a class simultaneously. For example, in a large team where different developers are responsible for different aspects of a class, using partial classes can allow them to work independently on their respective parts of the class without stepping on each other's toes. This can lead to more efficient development workflows and reduce conflicts when merging changes from multiple developers.
* **Readability and maintainability:** Partial classes can be used to improve the readability and maintainability of the codebase by logically grouping related code into separate files. For example, you can have one partial class file for defining properties and fields, another partial class file for defining methods, and yet another partial class file for implementing interfaces or handling events. This can make the codebase easier to navigate and understand, especially in large and complex projects.
* **Third-party library customization:** Partial classes can be used to customize the behavior of third-party libraries or frameworks by extending their classes with additional functionality. For example, if you are using a third-party library that provides a base class or an abstract class, you can create a partial class that extends that class and provides additional methods or properties specific to your application's needs, without modifying the source code of the third-party library.

In summary, partial classes in C# are useful in real-world scenarios where you need to split the definition of a class across multiple files for code generation, separation of concerns, collaboration between developers, readability and maintainability, or third-party library customization. They provide a way to logically group related code in separate files, making the codebase more organized, maintainable, and extensible.

What are partial methods in C# and when would you use them?

Partial methods in C# are used to declare a method in one part of a partial class but allow the implementation of the method to be defined in another part of the same partial class. Partial methods must have a declaration in one part of the partial class and an implementation in another part, but if the implementation is not provided, the compiler removes the method during compilation, resulting in no runtime overhead. Partial methods are commonly used in code generation scenarios, where the generated code can provide hooks for developers to implement custom logic. For example, a code generation tool may generate a partial class for database access, with partial methods for pre-processing and post-processing of data, allowing developers to provide custom logic for data validation or transformation.

Explain real world scenarios where partial methods are useful?

Partial methods in C# are used in scenarios where you want to provide a way to optionally extend or customize a class's behavior in a separate file or assembly. Here are some real-world scenarios where partial methods can be useful:

* **Code generation:** Partial methods are commonly used in code generation scenarios where a portion of the method is generated automatically, and another portion is intended to be implemented by a developer. For example, if you are using a tool or framework that generates code, such as a code generator or an ORM (Object-Relational Mapping) tool, it may generate partial methods with the intention that you can implement the missing logic in a separate partial class file. This allows you to customize the generated code without modifying the generated portion, which can be helpful when the code is regenerated.
* **Framework or library hooks:** Partial methods can be used as hooks or callbacks in frameworks or libraries to allow developers to provide custom logic at specific points in the framework or library's execution flow. For example, a framework or library may provide partial methods that are called before or after certain events or operations, allowing developers to extend or customize the behavior of the framework or library without modifying its source code directly. This can be useful in cases where you want to add custom logic to a framework or library's behavior without having to modify its core implementation.
* **Optional behavior:** Partial methods can be used to provide optional behavior that can be implemented or omitted based on the needs of the application or module. For example, in a large application with multiple modules or plugins, you may want to provide optional functionality in certain modules without requiring all modules to implement the same logic. Partial methods can be used to provide a template for the optional behavior that can be implemented by modules that need it, and omitted by modules that don't.
* **Code separation and organization:** Partial methods can be used to separate and organize code related to a specific feature or aspect of a class in a separate file. For example, if you have a class with multiple responsibilities or features, you can use partial methods to separate the implementation of each feature into separate partial class files. This can help in keeping the codebase organized and maintainable, as each partial class file can contain the implementation of a specific feature or aspect of the class.
* **Legacy code integration:** Partial methods can be used to integrate legacy or third-party code into a new codebase in a more controlled manner. For example, if you are migrating a legacy system to a new codebase, you may want to gradually replace legacy code with new code without disrupting the existing functionality. Partial methods can be used as a transitional mechanism to gradually refactor the legacy code into the new codebase, allowing you to implement new logic in partial methods while keeping the existing legacy code intact.

In summary, partial methods in C# are useful in real-world scenarios where you need to provide a way to optionally extend or customize a class's behavior, such as in code generation, framework or library hooks, optional behavior, code separation and organization, or legacy code integration. They provide a mechanism to implement optional logic in a separate file or assembly, allowing for flexibility, code separation, and customization without modifying the core implementation of a class or framework.

Can partial methods have implementation in more than one part of a partial class? If not, why?

No, partial methods in C# can have at most one implementation in one part of a partial class. If an implementation is provided in one part, the other parts must not provide any implementation for the same partial method. This is because partial methods are implicitly private, and the compiler removes them during compilation if they have no implementation. This allows for optional implementation of partial methods, where the presence or absence of an implementation does not affect the behavior of the calling code. If multiple implementations were allowed, it would introduce ambiguity and could result in unexpected behavior.