Part01 Questions

1. Why does defining a custom constructor suppress the default constructor in C#?

When we define any custom constructor in a class, C# does not automatically generate the default constructor anymore.
And this happens because the compiler assumes that since we defined our own constructor, we want full control over object initialization.
If we still want a default constructor, we must explicitly define it.

2. How does method overloading improve code readability and reusability?

Method overloading allows us to use the same method name with different parameters.
and this improves readability because:

- The method name stays consistent.
- The purpose of the method is clear.

It improves reusability because:

- We avoid writing different method names for similar operations.
- The same logic concept is reused with different inputs.

Also it makes the code cleaner and easier to understand.

3. What is the purpose of constructor chaining in inheritance?

Constructor chaining allows a derived class to call the constructor of its base class.
Its purpose is:

- To initialize inherited properties properly.
- To avoid code duplication.
- To ensure that the base class part of the object is initialized before the child class.

And it guarantees correct object construction in inheritance.

4. How does “new” differ from “override” in method overriding?

“override” is used when we want to provide a new implementation of a virtual method from the base class. It supports runtime polymorphism.

While “new” is used to hide a method from the base class. So it doesnt override it, but creates a separate version. And It supports compile-time behavior.
The main difference:

- override replaces the base method.
- new hides the base method.

5. Why is “ToString()” often overridden in custom classes?

ToString() is overridden to provide a meaningful string representation of an object.
By default, ToString() returns the class name. But when we override it, we can display useful information like property values.
This improves debugging, and readability.

6. Why can't you create an instance of an interface directly?

An interface cannot be instantiated because:

- It does not contain implementation.
- It only defines method signatures.

Since there is no actual code inside the interface, we must create an instance of a class that implements it.

7. What are the benefits of default implementations in interfaces introduced in C# 8.0?

Default implementations allow interfaces to provide method bodies.
Benefits:

- We can add new methods to interfaces without breaking existing classes.
- It improves backward compatibility.
- It reduces code duplication.

This makes interfaces more flexible.

8. Why is it useful to use an interface reference to access implementing class methods?

Using an interface reference provides abstraction.
Benefits:

- It allows loose coupling.
- It makes code more flexible.
- We can switch implementations easily.
- It supports polymorphism.

It makes the system more maintainable and scalable.

9. How does C# overcome the limitation of single inheritance with interfaces?

C# does not allow multiple inheritance between classes while a class can implement multiple interfaces. So this allows a class to inherit behavior contracts from multiple sources without ambiguity.

10. What is the difference between a virtual method and an abstract method in C#?

A virtual method has a body in the base class. And it can be overridden while the overriding is optional.
An abstract method has no body. And must be overridden in the derived class.
Virtual = optional override.
Abstract = mandatory override.

Part02

1. What is the difference between class and struct in C#?

A class is a reference type. This means when we create an object from a class, the object is stored in the heap, and the variable stores a reference to that object. Because it is a reference type, if we assign one object to another variable, both variables will refer to the same object in memory.

A struct is a value type. This means it is usually stored in the stack, and when we assign it to another variable, a copy of the value is created. Each variable has its own separate copy.

Another difference is that a class supports inheritance, meaning it can inherit from another class. A struct does not support inheritance from another struct or class, but it can implement interfaces.

Also, a class object can be null because it is a reference type. A struct cannot be null unless we use nullable struct syntax.

In general, a class is used for complex objects and larger systems, while a struct is used for small, simple data structures like coordinates or small data models.

2. If inheritance is relation between classes clarify other relations between classes

Besides inheritance (IS-A relationship), we have the Association which is a general relationship between two classes.

And Aggregation**,** A “has-a” relationship where objects can exist independently.

For example: Car has Engine.

And Composition, A stronger “has-a” relationship. The contained object cannot exist without the parent.

Like House has Rooms.

Part03

1. what is static and dynamic binding

- Static Binding (Compile-time binding): Occurs at compile time.
  Example:
    - Method overloading
    - Using new

The method to call is decided during compilation.

- Dynamic Binding (Runtime binding): Occurs at runtime.
  Example:
    - Method overriding
    - Using virtual and override

The method to call is decided when the program runs.
