Question01:

If I have defined a constructor of my own, especially one with parameters, then I most likely have logic of my own that must be executed on creating the class. If the compiler were to create an empty, parameter less constructor in this case, it would allow someone to *skip* the logic that I had written, which might lead to my code breaking in all number of ways. If I want a default empty constructor in this case, I will create one of my own.

Question02:

Method overloading improves code readability and reusability by allowing multiple methods that perform conceptually similar tasks to share the same name but accept different parameters.

Question03:

The primary purpose of constructor chaining is to ensure proper and consistent initialization of an object across its entire class hierarchy while also promoting code reusability and reducing redundancy.

Question04:

override: overrides the functionality of a virtual method in a base class, providing different functionality.

new: hides the original method (which doesn't have to be virtual), providing different functionality.

Question05:

ToString() is often overridden in C# to provide a meaningful, human-readable string representation of the object's current state, which is vital for debugging, logging, and display purposes.

Part02:

Question01:

Class is a reference type stored in heap and supports inheritance and polymorphism. Struct is a value type stored in stack, copied by value, lightweight, and doesn't support inheritance.

Question02:

While Inheritance can be defined as → IS-A there are other relationships between classes like:
Association which can be defined as→ Uses-a

Aggregation (Has-a) but it represents a week relationship as the child class can exist without the base class.

But is Composition (Has-a) it represents a strong relationship because the child class can not exist without the parent class existing.

Dependency → Temporary use

Part03:

Question02:

Static Binding (Early Binding)

Definition:

Static binding means the method call is resolved at compile time. The compiler already knows exactly which method to execute. No runtime decision.

Happens with:

normal methods, static methods, private methods, sealed methods, method overloading, methods hidden using new

Behavior:

Decision is based on reference type, NOT the actual object.

Example:

If Parent reference → Child object

Compiler still calls parent method, Because it checks the reference type only.

Dynamic Binding (Late Binding)

Definition: Dynamic binding means the method call is resolved at runtime. The program decides while running which method to execute.

Happens with:

virtual methods, override methods, abstract methods, interfaces

Behavior

Decision is based on actual object type, not the reference type.

Example:

If Parent reference → Child object

Runtime calls child method, Because the real object is Child.