Part01

- **Question: Why does defining a custom constructor suppress the default constructor in C#?**
  Defining a custom constructor in a class suppresses the default constructor because the compiler assumes you are handling object initialization yourself.
  If you need a parameterless constructor, you must define it explicitly

- **Question: How does method overloading improve code readability and reusability?**
  Method overloading improves code readability and reusability because it allows multiple methods with the same name but different parameters.
  Readability: The same method can perform the same action with different types or numbers of inputs without needing different names.
  Reusability: The same name can be used for similar operations, reducing code duplication.

- **Question: What is the purpose of constructor chaining in inheritance?**
  The purpose of constructor chaining in inheritance is to initialize the base class first before the derived class.
  This ensures all properties of the object are properly set in order.

- **Question: How does new differ from override in method overriding?**
  override:
  Used to override a method inherited from a parent class declared as virtual.
  When called from a parent class reference, it executes the child's version.

  new:
  Used to hide an inherited method instead of overriding it.
  When called from a parent class reference, it executes the parent's version, even if the object is from the child.

- **Question: Why is ToString() often overridden in custom classes?**
  To provide a clear textual
  representation of the object

  To make printing and debugging easier

- **Question: Why can't you create an instance of an interface directly?**
  The interface is just a definition; it does not have actual implementation (code) for its methods, so you cannot create an instance of it directly.

You must create a class that implements the interface, then you can create an object of that class and work with the interface through it

- **Question: What are the benefits of default implementations in interfaces introduced in C# 8.0?**
  interface:
  If you add a method with a default implementation in an interface, not every class implementing the interface has to override it.
  A class can:
  Use the default implementation as provided in the interface.
  Or override the method if it wants a different behavior.

- **Question: Why is it useful to use an interface reference to access implementing class methods?**
  Using an interface reference is useful because it allows polymorphism and flexible code:
  You can work with different objects implementing the same interface in the same way.
  Functions or variables can depend on the interface instead of specific classes, reducing coupling**.**

- **Question: How does C# overcome the limitation of single inheritance with interfaces?**
  In C#, a class can inherit from only one class, but it can implement multiple interfaces.
  Each interface defines a set of methods and properties that must be implemented.

- **Question: What is the difference between a virtual method and an abstract method in C#?**
  Virtual method:
  Can be used as-is in the base class or overridden in the derived class.

  Abstract method:
  Must be implemented in the derived class; the base class provides no implementation.

Part02

- **What is the difference between class and struct in C#?**
  Class:
  Refernce type
  Stored in heap
  Support inheritance
  If no constructor is defined, a default constructor exists

  Struct:
  Value typ
  Stored in Stack
  Cannot inherit from another struct (but can implement interfaces)
  Always has a default constructor; cannot define parameterless constructor

- **If inheritance is relation between classes clarify other relations between classes?**
  Inheritance : A class inherits from another class.
  Association : A class contains a reference to another class.
  Aggregation : A “part-of” relationship, but the part is independent.
  Composition : A “part-of” relationship where the part cannot exist without the whole.
  Dependency : A class temporarily uses another class inside a method or operation.

Part03 Bonus

- **what is static and dynamic binding?**
  Static Binding (Early Binding): The method to be called is decided at compile time. Happens with non-virtual methods.

  Dynamic Binding (Late Binding): The method to be called is decided at run time. Happens with virtual/overridden methods and allows polymorphism.