Question1:

Value type vs. reference type:
Structs in C# are value types, while classes are reference types. Inheritance is designed for reference types, because it relies on polymorphism and reference semantics. Structs are copied by value, which makes inheritance impractical.

- Design choice:
  Microsoft designed structs to be lightweight containers for small pieces of data (like points, colors, etc.). Allowing inheritance would make them behave more like classes, which defeats their purpose.
- Memory management:
  Structs are usually stored on the stack and copied entirely when passed around. If inheritance were allowed, it would complicate memory layout and performance.
- Alternative:
  Structs *can* implement interfaces, which gives them polymorphic behavior without full inheritance.

Question2:

Access modifiers in C# determine where and how class members (fields, methods, properties) can be accessed:

- private → The member is accessible only inside the same class.
- internal → The member is accessible anywhere within the same project/assembly, but not from outside projects.
- public → The member is accessible from anywhere, including other projects that reference this one.

Question3:

Encapsulation is one of the core principles of object-oriented programming (OOP). It means hiding the internal details of a class or struct and exposing only what's necessary through controlled interfaces (methods/properties).

Here's why it's critical:

- Data protection:
  By keeping attributes private and exposing them through getters/setters or properties, you prevent direct modification of sensitive data. This ensures integrity and avoids accidental misuse.

- Controlled access:
  You can enforce rules when data is read or written. For example, you can validate a salary before setting it, instead of allowing any random value.
- Flexibility and maintainability:
  If you change the internal implementation later, external code doesn't break because it only interacts with the public interface.
- Encapsulation = abstraction + security:
  It hides complexity and protects the internal state of objects, making software more robust and easier to maintain.

Question4:

A constructor in a struct is a special method that runs automatically when you create a new instance of the struct. Its purpose is to initialize the fields of the struct with specific values at the time of creation.

Key points about constructors in structs (C# context):

- They have the same name as the struct.
- They do not return a value (not even void).
- Structs can have parameterized constructors (with arguments).

- Structs cannot define a parameterless constructor, because the compiler always provides a default one automatically.
- Constructors allow overloading, meaning you can have multiple constructors with different parameter lists to initialize the struct in different ways.

Question5:

Overriding methods such as ToString() makes your code clearer, more expressive, and easier to understand because:

- Human-friendly output: Instead of printing the default type name (e.g., task6csharp.Point), you can display meaningful information like "Point: (3, 7)".
- Debugging made easier: When you log or inspect objects, the overridden ToString() gives immediate insight into their state without needing extra print statements.
- Consistency: You can define a standard way to represent your objects across the whole project.

- Encapsulation of formatting: The logic for how an object should be displayed is kept inside the object itself, rather than scattered across your code.