1- Difference between class and struct in C#:

- A **class** is a reference type, while a **struct** is a value type.
- Objects of a class are stored on the **heap**, while objects of a struct are stored on the **stack**.
- A class supports **inheritance**, but a struct does not (except it can implement interfaces).
- Members of a class can be declared with **access modifiers** (public, protected, private, etc.), while struct members cannot be protected or protected internal.
- A class can have a **default constructor** (without parameters), but a struct cannot define a parameterless constructor (the compiler provides one automatically).

2- If inheritance is relation between classes clarify other relations between classes:

- **Association**: A general relationship between two classes, where one class uses or is connected to another. Example: A `Teacher` and a `Student` have an association.
- **Aggregation**: A "has-a" relationship where one class contains another class, but the contained object can exist independently. Example: A `Department` has `Teachers`, but teachers can exist without the department.
- **Composition**: A stronger form of aggregation where the lifetime of the contained object depends on the container. Example: A `Car` has an `Engine`; if the car is destroyed, the engine is also destroyed.
- **Dependency**: A "uses" relationship where one class depends on another to perform a task, usually shown when a class creates or calls methods of another. Example: A `Report` class uses a `Printer` class to print.

Feature	Class (Reference Type)	Struct (Value Type)
Type	Reference type	Value type
Memory location	Stored on the **heap**	Stored on the **stack**
Inheritance	Supports inheritance	Does **not** support inheritance (can implement interfaces only)
Access modifiers	Can use all access modifiers (public, private, protected, …)	Cannot use **protected** or **protected internal**
Constructors	Can define a default constructor (parameterless)	Cannot define a parameterless constructor (compiler provides one automatically)
Performance	Slightly slower due to heap allocation	Faster for small data types since stored on stack
Use case	Best for large, complex objects and inheritance	Best for small, lightweight data structures