Question Answering Bonus

1. What is static and dynamic binding?

In C#, **static binding** and **dynamic binding** refer to how method calls or property accesses are resolved in relation to classes and their relationships, such as inheritance. These concepts are particularly relevant in object-oriented programming and polymorphism. Below, I'll explain static and dynamic binding, their differences, and how they apply in C#, including their relevance to class relationships like inheritance, association, and realization (as discussed previously).

1. Static Binding

- **Definition**: Static binding (also called early binding) occurs when the method or property to be invoked is determined at **compile time** based on the type of the reference or variable. The compiler resolves the method call using the declared type of the variable, not the actual type of the object at runtime.
- **Characteristics**:
  - Resolved at compile time.
  - Faster because the method call is fixed before execution.
  - Used for methods that are static, private, final, or not overridden (e.g., non-virtual methods in C#).
  - Does not support runtime polymorphism.
- **When It Occurs**:
  - Methods marked as static or private.
  - Methods that are not marked with virtual or override.
  - When the compiler can definitively determine the method based on the reference type.
- **Example**:

```
public class Animal
{
   public void Eat() => Console.WriteLine("Animal eats");
}

public class Dog : Animal
{
   public new void Eat() => Console.WriteLine("Dog eats"); // Hides
base class method
}

class Program
{
   static void Main()
   {
       Animal animal = new Dog(); // Reference type is Animal
       animal.Eat(); // Output: Animal eats (static binding to
Animal's Eat)
   }
}
```

    - The method Eat is resolved at compile time based on the reference type Animal, not the actual type Dog. Since Eat is not virtual, static binding applies.

2. Dynamic Binding

- **Definition**: Dynamic binding (also called late binding) occurs when the method or property to be invoked is determined at **runtime** based on the actual type of the object, not the reference type. This enables runtime polymorphism.
- **Characteristics**:
    - Resolved at runtime using the object's actual type.
    - Slower than static binding due to runtime lookup (via virtual table, or vtable, in C#).

    - Used for methods marked with virtual in the base class and override in the derived class.
    - Essential for polymorphic behavior in inheritance hierarchies.

- **When It Occurs**:
    - Methods marked with virtual in the base class and override in the derived class.
    - Interface method implementations (when called through an interface reference).
    - Enables runtime flexibility in inheritance and realization relationships.

- **Example**:

```
public class Animal
{
   public virtual void Speak() => Console.WriteLine("Animal speaks");
}

public class Dog : Animal
{
   public override void Speak() => Console.WriteLine("Dog barks");
}

class Program
{
   static void Main()
   {
       Animal animal = new Dog(); // Reference type is Animal, actual
type is Dog
       animal.Speak(); // Output: Dog barks (dynamic binding to Dog's
Speak)
   }
}
```

    - The method Speak is resolved at runtime based on the actual type (Dog), not the reference type (Animal), because it's marked virtual and override.

3. **Differences**:

Resolution Time

- Static Binding: Occurs at compile time, where the method or property to be invoked is determined before the program runs.
- Dynamic Binding: Occurs at runtime, where the method or property is resolved based on the actual object type during execution.

Performance

- Static Binding: Faster, as it avoids runtime lookup and fixes the method call at compile time.
- Dynamic Binding: Slower, due to the need for runtime dispatch (e.g., via a virtual table).

Method Types

- Static Binding: Applies to static, private, non-virtual methods, or methods hidden with the new keyword.
- Dynamic Binding: Applies to virtual methods, override methods, and interface method implementations.

Polymorphism

- Static Binding: Does not support runtime polymorphism, as the method call is based on the reference type.
- Dynamic Binding: Supports runtime polymorphism, allowing method calls to resolve to the actual object's implementation.

Based On

- Static Binding: Determined by the reference type (the type of the variable or parameter).
- Dynamic Binding: Determined by the actual object type at runtime, regardless of the reference type.

Use Case

- Static Binding: Ideal for fixed, predictable method calls where polymorphism is not required (e.g., utility or private methods).
- Dynamic Binding: Suited for flexible, polymorphic behavior in inheritance hierarchies or interface implementations.