**Composition Over Inheritance: A C# Example**

**Understanding the Concept**

* **Inheritance:** An "is-a" relationship. A derived class inherits the properties and methods of a base class.
* **Composition:** A "has-a" relationship. A class contains instances of other classes as members.

**Why Composition Over Inheritance?**

* **Flexibility:** Composition allows for more flexible and adaptable designs.
* **Reusability:** Composed objects can be reused in different contexts.
* **Avoids Tight Coupling:** Composition reduces tight coupling between classes.
* **Prevents Inheritance Hierarchies:** Overly complex inheritance hierarchies can be difficult to maintain.

**Example: Vehicle and Engine**

Let's consider a simple example: a Vehicle class. We could use inheritance to create specific vehicle types like Car and Motorcycle, but this might lead to tight coupling and inflexible designs.

**Inheritance-Based Approach:**

C#

public class Vehicle

{

public virtual void Start()

{

Console.WriteLine("Vehicle started.");

}

}

public class Car : Vehicle

{

public override void Start()

{

Console.WriteLine("Car started.");

}

}

public class Motorcycle : Vehicle

{

public override void Start()

{

Console.WriteLine("Motorcycle started.");

}

}

Use code [with caution.](/faq#coding)

**Composition-Based Approach:**

C#

public class Engine

{

public void Start()

{

Console.WriteLine("Engine started.");

}

}

public class Vehicle

{

private Engine \_engine;

public Vehicle(Engine engine)

{

\_engine = engine;

}

public void Start()

{

\_engine.Start();

}

}

public class Car : Vehicle

{

public Car() : base(new Engine())

{

}

}

public class Motorcycle : Vehicle

{

public Motorcycle() : base(new Engine())

{

}

}

Use code [with caution.](/faq#coding)

**Key Differences and Benefits of Composition:**

1. **Flexibility:** We can create different vehicle types by simply composing them with different engine types.
2. **Reusability:** The Engine class can be reused in other contexts, such as boats or airplanes.
3. **Loose Coupling:** The Vehicle class is loosely coupled to the Engine class, making it easier to modify or replace the engine implementation without affecting the vehicle's behavior.

**When to Use Composition:**

* When a class needs to use the functionality of another class but doesn't have an "is-a" relationship.
* When you want to avoid tight coupling between classes.
* When you want to create more flexible and reusable designs.

By favoring composition over inheritance, you can create more modular, maintainable, and adaptable software systems.