C# Intermediate: Classes, Interfaces and OOP

ASSOCIATION BETWEEN CLASSES

Class Coupling

- A measure of how interconnected classes and subsystems are.
- The more coupled classes, the harder it is to change them. A change in one class may affect many other classes.
- Loosely coupled software, as opposed to tightly coupled software, is easier to change.
- Two types of relationships between classes: Inheritance and Composition.

Inheritance

- A kind of relationship between two classes that allows one to inherit code from the other.
- Referred to as Is-A relationship: A Car is a Vehicle
- Benefits: code re-use and polymorphic behaviour.

```
public class Car : Vehicle
{
}
```

Composition

- A kind of relationship that allows one class to contain another.
- Referred to as Has-A relationship: A Car has an Engine

- Benefits: Code re-use, flexibility and a means to loose-coupling

```
public class DbMigrator
{
        // We re-use the code in the logger class without
        // the need to repeat that logic here in DbMigrator
        private Logger _logger;
}
```

Favour Composition over Inheritance

- Problems with inheritance:
    - Easily abused by amateur designers / developers
    - Leads to large complex hierarchies
    - Such hierarchies are very fragile and a change may affect many classes
    - Results in tight coupling
- Benefits of composition:
    - Flexible
    - Leads to loose coupling
- Having said all that, it doesn’t mean inheritance should be avoided at all times. In fact, it’s great to use inheritance when dealing with very stable classes on top of small hierarchies. As the hierarchy grows (or variations of classes increase), the hierarchy, however, becomes fragile. And that’s where composition can give you a better design.