Generics are a powerful feature that allow us to write more reusable code.

Generics are a technique for writing code without specifying data types but still providing type safety. Meaning everything has a type which is verified at compile time.

OperationResult – the purpose of this type of class is to return more than one value back from a method. It means we can return the data type of the method return and message.

OperationResult is a good example of the usefulness of generics. The return value type could vary per method so a generic result value would be beneficial.

<T> - by convention the capital letter T is used to represent the type variable. The variable is called the generic type. We can use it to represent any type.

How do we use it? The variable type, T, needs to be defined somewhere. We define it in the class signature. In angle brackets after the class name. This is how we define a property of type T.

A generic class can have any number of generic properties: <T, V>

Why? The class is now a generic class because it has a generic type parameter.

Do:

Use generics to build reusable, type neutral classes

Use T as the type param for classes with one type param

Prefix descriptive Type param names with the letter T. <TResult, TMessage>

Avoid:

Using generics when it is not needed.

Use single letter type params for classes with more than one. Use meaningful names.

Expression-bodied method syntax – define a method with a single line of code.

E.G.

public decimal CalculateSuggestedPrice(decimal markupPercent) =>

this.Cost + (this.Cost \* markupPercent / 100);

One setback of expression bodied method syntax is you can’t guard against the incoming args.

DRY principle – don’t repeat yourself.

When we create an instance of a generic type class we need to state the type of the param.

We can define generic methods as well. The type is defined in the method signature:

public T methodName<T>() – returns variable type T. By defining the type at method level, different methods can use different variable definitions which adds flexibility that would not occur if the type were defined at class level.

Generic constraints – limiting the types the generic property can be.

Generic constraints are defined with the ‘where’ keyword:

where T : struct – constrains T to a value type.

where T : class – constrains T to a ref type. Class, interfaces, arrays.

where T : new() – constrains to paramaterless constructors.

FAQ:

1. What are generics?

A technique for defining data types as a variable.

1. What are the benefits of generics?

We can write generic reusable code that is type safe and can work with any type.

1. What is a generic type parameter?

A placeholder for a specific type.

1. Where is a generic type parameter defined?

In the class signature or the method signature.

1. What is the purpose of a generic constraint?

To constrain the type of the generic type parameter.