Interfaces

Quack, quack!

- We have seen `List`, `Dictionary`, and `Queue`.
- C# has many other collections as well.
- These include `SortedList`, `Stack`, `HashSet`, and `LinkedList`.

- How can `C#` know how to use `foreach` and other methods that work for all?

The answer is: interfaces

Interfaces

An interface is a description of a set of behaviors that a type can have. The interface defines the behaviors that a type must have, and the type must implement these. Later on, we'll learn that methods implement behaviors. An interface describes a set of methods.

ALERT: This is likely an interview question.

Interface Example

Let's use a real-world example to help understand interfaces.

- We'll use the example of an `Animal`.

Common Behaviors

We know all `Animals` have certain behaviors.

- `eat`
- `sleep`
- `breathe`

However, all animals have different ways of doing these.

Define an Interface

We could define an "interface" that describes these generic behaviors and call this an `IAnimal`.

Notice the `I` in the name. The `I` indicates that this is an `Interface`.

We can't make an instance of an `IAnimal`; it wouldn't be able to do anything.

Concrete Implementations

We could create a `Bear` and a `Cat` and `Human` and say that they all have the behaviors of an`IAnimal`.

The `IAnimal` is a *generic* type, and the `Bear` and `Cat` and `Human` are *concrete* types.

Writing code that uses Interfaces

We could code our software to use an `IAnimal`, and we wouldn't care if our code received a `Bear` or a `Cat` or a `Human`.

We would be able to say that they have the behaviors of an `IAnimal`.

Quack!

In fact, in other languages, this is known as `duck typing`! This name comes from the phrase:

`If it walks like a duck, quacks like a duck, and looks like a duck, it must be a duck.`

In our case, if it `eats` and `sleeps` and `breathes` like an `IAnimal`, it must be an `IAnimal`.

Huh, what, why?!?!?

We'll be **using** interfaces more than we will be creating them in our work.

- They are a compelling language feature and are the type of tool you'll use more as you grow in your programming skills.

Ok, so besides the definition, what else do I need to know?

You may have noticed when adding a `using` statement to the top of your code, `List`, and `Dictionary`, and `Queue` come from `using System.Collections.Generic`.

There is that `Generic` word again.

Ok, so besides the definition, what else do I need to know?

In a later lesson, we'll learn about a `C#` feature named `LINQ`, and we will see that `List`, `Dictionary`, and `Queue` all adhere to the `IEnumerable` interface.

And since all these types are `IEnumerable` `LINQ` will apply to each.

Takeway

- Different types in C# share common behaviors. An `Interface` allows these types to share behaviors.
- If two things share a common `Interface`, they work similarly, and each does the things the `Interface` describes.
- We'll see them a lot more than we create them.
- When we see that some type is an `ISomething`, the `I` is a convention that says, "This is an `Interface`".