Generic Collections

You’ve done great with lists so far! It’s time to take a look at the bigger picture.

Remember the one line we mentioned at the beginning of this lesson?

```
using System.Collections.Generic;
```

The list class is in a group of classes called *generic collections*. They don’t exist in the default set of `System` classes, so we need to make a reference to them with this line.

Generic collections are data structures that are defined with a generic type. Each class is defined generally without a specific type in mind. When we make an actual instance, we define the specific type:

```
List<string> citiesList = new List<string>();
List<Object> objects = new List<Object>();
```

Imagine it like a set of general instructions: in a toy store, we can tell the employees how to add and remove items from a shelf without specifying the type of toy. In the same way, we can use `Add()` and `Remove()` without knowing a lists’s data type.

For this reason, the formal class name of lists is `List<T>`. That `T` is a type parameter: it represents some type that we can specify later. The general instructions, however are neatly contained in the generic `List<T>` class.

Let’s see why this is useful by imagining the other, more difficult ways we could create “generic” collections:

- Use type-specific classes, like `StringList`, `IntList`, etc. — We would have to make a list class for EVERY type, defining the same properties and methods for each list class.
- Use a list containing `Object` types, `List<Object>` — Using `Object` means we can’t use any of the unique functionality of each type and it takes a lot of computing power to convert references to and from the `Object` type.

As you continue coding, you’ll see for yourself how useful generic collections are!

☑Instructions

1.
Make a reference to the `System.Collections.Generic` namespace.

Hint

To reference a namespace, use the `using` keyword.

2.
Declare three empty lists:

- one should hold `bool` types
- one should hold `Random` types
- one should hold `IServiceProvider` types

That’s right, interfaces work here too!

Hint

Here’s an example that declares a list holding `char` types:

```
List<char> alphabet = new List<char>();
```