## 0. Introduction

### 1. Essentials deep dive

### 2. What you should know

### 3. CoderPad tour

## 1. Generics

### 1. Understanding generics

### 2. Working with generics

### 3. Defining generic types and methods

### 4. Generic constraints

### 5. <> Code Challenge - Generic type with constraints

### 6. Solution - Generic type with constraints

### Chapter Quiz

Question 1 of 4

In C#, what do generics allow you to do?

Create a type that is both a value type and reference type at the same time.

Incorrect

Generics do not have an impact on value or reference type definition.

Define methods that can return multiple output values.

Incorrect

Generic methods can still only have a single return type.

Create methods and types that can support many types while also being type-specific when invoked.

Correct

Generics enable you to define types or methods that are flexible in the types used for parameters and return values.

Create non-nullable value types.

Incorrect

Non-nullable value types are supported by Nullable<T>, but that is not the only use for generics.

Question 2 of 4

When invoking a generic method, how are generic types applied?

They can only specify return types.

Incorrect

Generic types can be applied to return types or parameters.

They can indicate the parameters or return types.

Correct

Generics enable you to define types or methods that are flexible in the types used for parameters and return values.

They can only specify parameters.

Incorrect

Generic types can be applied to return types or parameters.

Question 3 of 4

Generic types can only have a single type parameter.

FALSE

Correct

Generic types can have numerous type parameters.

TRUE

Incorrect

Generic types can have numerous type parameters.

Defining generic types and methods

Replay

Review this video

Defining generic types and methods

7m 27s

Question 4 of 4

What is the purpose of a constraint in a generic definition?

It ensures that a generic type parameter has a certain property.

Incorrect

Constraints do not enforce certain properties or methods.

It restricts a generic type so that it can only be used in an interface definition.

Incorrect

There is no constraint to enforce that the generic type is used as an interface.

It requires consumers to use types that meet the requirements for how the type is used.

Correct

Constraints ensure the type parameter can be used correctly in your method and won't cause complication errors.

## 2. Working with Collections

### 1. Understanding collection types in the BCL

### 2. Choosing the right collection type

### 3. Working with collections

### 4. Working with keyed collections

### 5. Understanding concurrent collections

### 6. <> Code Challenge - Choose the right collection

#### **Instructions**

**Choose the right collection type**

You're given a class that allows items to be added and retrieved (and removed).

**Your task**: Implement the Add and Retrieve methods such that items are returned in the order they were entered; the first item entered should be the first returned.

**Parameters**

**product**: An item to be added to the list.

Result

**Product**: The next item to retrieve from the list, or null if no items remain.

**Constraints**

The product parameter **will not be null**.

**Example 1:**

Input: productA then productB

Result (first): productA

**Example 2:**

Input: productD then productC

Result (first): productD

Want a hint?

Learn about .NET Collections [this course](https://www.linkedin.com/learning/c-sharp-essential-training-2-generics-collections-and-linq/understanding-collection-types-in-the-bcl) on LinkedIn Learning.

#### **Answer**

public class Product {

    public string Name {get;set;}

    public int ProductID {get;set;}

}

public class Answer {

    // Change these Boolean values to control whether you see

    // the expected result and/or hints.

   public  static Boolean ShowExpectedResult = false;

   public  static Boolean ShowHints = false;

    // Return the largest number in the 'numbers' array.

    public static void Add  (Product p) {

        // Your code goes here.

        // Add the item to a collection

        list.Enqueue(p);

    }

    public static Product Retrieve() {

        //Your code goes here to retrieve the next object

    }

}

#### **Test code**

// You can edit the array of products to test other use cases.

var products = new Product[] {

    new Product{Name="Widget", ProductID = 1},

    new Product{Name="Gizmo", ProductID = 73}

};

for(int productIndex=0; productIndex< products.Length; productIndex++){

    Answer.Add(products[productIndex]);

}

Product learnerResult = Answer.Retrieve();

### 7. Solution - Choose the right collection

### Chapter Quiz

Question 1 of 5

Why are generic collection types preferred over non-generic collection types?

Non-generic collections are not supported in .NET anymore.

Incorrect

Non-generic collections are still supported, mostly for backwards compatibility, and because some, like NameValueCollection, are still useful and need no generic equivalent.

Non-generic collections are not recommended because of the performance and usability implications.

Correct

Non-generic collections are legacy objects that have performance and usability implications for modern C#.

Because generic collections are ordered by default.

Incorrect

Some collections are ordered by default, but that is not true of all generic collections.

Question 2 of 5

How can you use the IEnumerable interface?

By using the Enumerate method on the ICollection interface.

Incorrect

ICollection does not provide enumeration of items, that is supported by the IEnumerable interface intrinsically.

automatically in a foreach statement, or manually with the GetEnumerator method

Correct

IEnumerable is recognized by the compiler in the `foreach` statement but can be manually iterated with the associated IEnumerator.

By calling the Sort or Reverse methods on your collection.

Incorrect

`Sort` and `Reverse` are internal implementations of some collection types but not part of the IEnumerable concept.

Working with collections

Replay

Review this video

Working with collections

8m 15s

Question 3 of 5

What is the function of the IList interface?

It allows for a collection to be enumerated in a foreach loop.

Incorrect

The IEnumerable interface enables the `foreach` loop iterating.

It provides the method definitions for adding, removing, and finding items in a collection.

Correct

The IList interface provides the common methods and properties for inserting or removing items in a collection.

Every collection must implement the IList interface.

Incorrect

Collections are not required to implement the IList interface. They may only implement ICollection or IEnumerable.

Question 4 of 5

What's the difference between Dictionary and NameValueCollection?

Dictionary is a generic collection replacement for NameValueCollection.

Incorrect

Dictionary might be a replacement for NameValueCollection in some scenarios, but it is not a complete replacement because it does not allow for multiple values to be associated with a single key.

NameValueCollection can have multiple values for the same name or key while Dictionary cannot.

Correct

NameValueCollection is unique in that it allow for multiple values to be associated with the same key.

NameValueCollection is a generic collection while Dictionary is non-generic.

Incorrect

NameValueCollection is not a generic type.

Question 5 of 5

What is the purpose of concurrent collections?

They are used in multi-threaded scenarios to synchronize collection access across threads.

Correct

Concurrent collections have additional safeguards in place to enable their use from multiple threads.

They are used to keep items sorted by an ID property.

Incorrect

Concurrent collections are not directly related to ordering of items.

They provide both first-in-first-out and first-in-last-out behavior depending on their configuration.

Incorrect

While ConcurrentQueue<T> and ConcurrentStack<T> are useful collections, they are distinct types for different collection semantics. No single collection type provides both first-in-first-out and first-in-last-out.

Understanding concurrent collections

Replay

Review this video

Understanding concurrent collections

3m 53s

## 3. Exception Handling

### 1. Basics of try-catch-finally

### 2. Exception filtering by type with catch

### 3. Exception filtering with when

### 4. Throwing and rethrowing exceptions

### 5. Creating custom exceptions

### 6. <> Code Challenge - Multiple exception types

### 7. Solution - Multiple exception types

### Chapter Quiz

## 4. Programming Threads with async and await

### 1. Threading basics

### 2. Basics of using async and await

### 3. Task management

### 4. Handling errors when using await

### 5. <> Code Challenge - Task management

### 6. Solution - Task management

### Chapter Quiz

## 5. Delegates and Events

### 1. Understanding delegates

### 2. Understanding events

### 3. Common delegates with Action<T> and Func<T>

### 4. Understanding lambda expressions

### 5. Expressions as delegates

### 6. Other uses for expressions

### 7. <> Code Challenge - Delegate using Func<T>

### 8. Solution - Delegate using Func<T>

### Chapter Quiz

## 6. Extension Methods

### 1. The case for extension methods

### 2. Writing an extension method

### 3. Using an extension method

### 4. Common extension methods on collections

### 5. <> Code Challenge - Extension method

### 6. Solution - Extension method

### Chapter Quiz

## 7. Language Integrated Query (LINQ)

### 1. Understand the purpose of LINQ

### 2. Querying collections with LINQ

### 3. Filtering and ordering queries

### 4. Other sources for LINQ

### 5. <> Code Challenge - LINQ query to filter and sort

### 6. Solution - LINQ query to filter and sort

### Chapter Quiz

## 8. Conclusion

### 1. Build something and continue learning