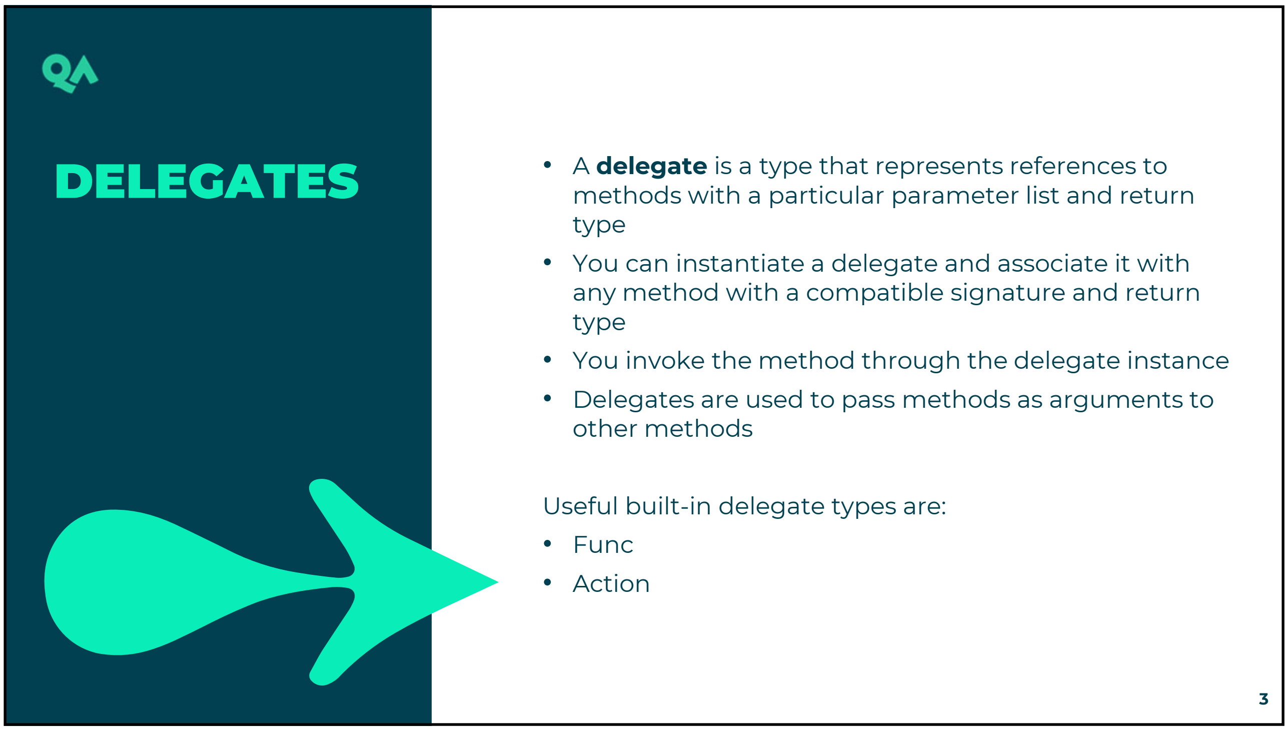
Delegates and Lambdas

1

OUTLINE

- Delegates
- The Func delegate
- The Action delegate
- Lambda expressions
- The Predicate delegate
- Delegate and Lambda examples

2



DELEGATES

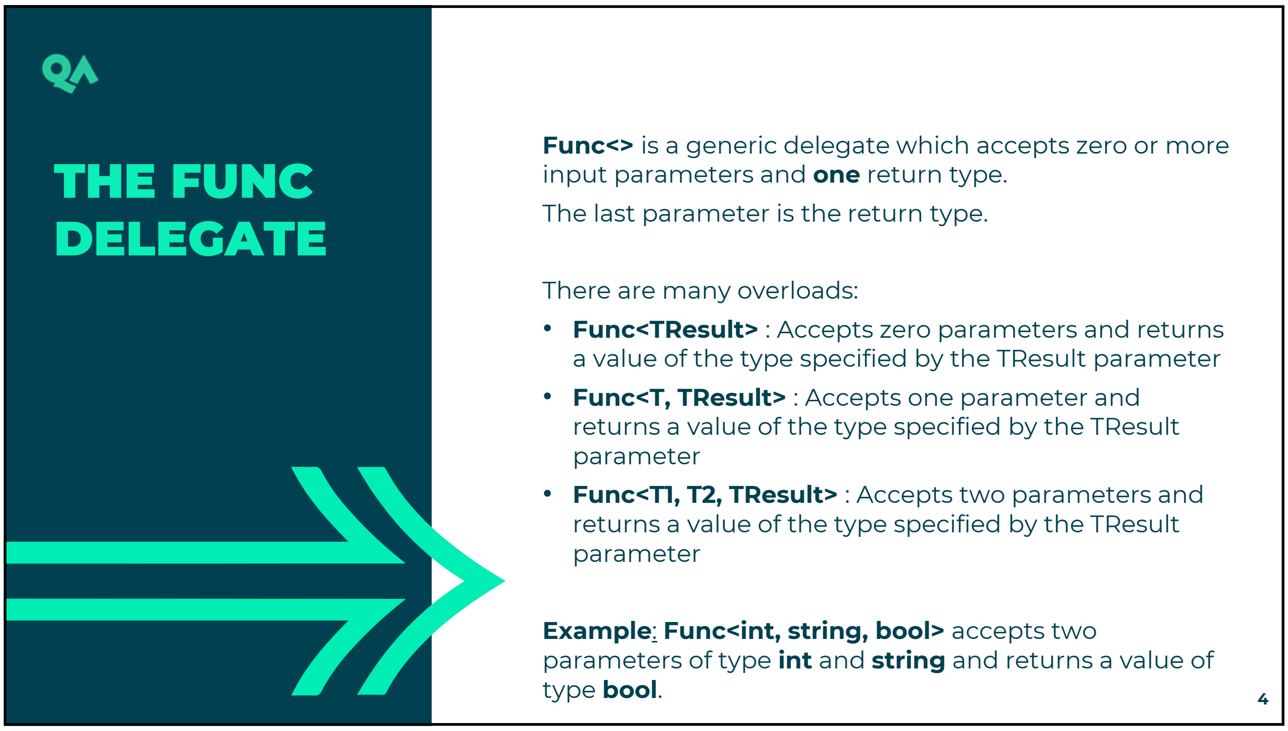
- A **delegate** is a type that represents references to methods with a particular parameter list and return type
- You can instantiate a delegate and associate it with any method with a compatible signature and return type
- You invoke the method through the delegate instance
- Delegates are used to pass methods as arguments to other methods

Useful built-in delegate types are:

- Func
- Action

3

3



THE FUNC DELEGATE

Func<> is a generic delegate which accepts zero or more input parameters and **one** return type.

The last parameter is the return type.

There are many overloads:

- **Func<TResult>** : Accepts zero parameters and returns a value of the type specified by the TResult parameter
- **Func<T, TResult>** : Accepts one parameter and returns a value of the type specified by the TResult parameter
- **Func<T1, T2, TResult>** : Accepts two parameters and returns a value of the type specified by the TResult parameter

Example: **Func<int, string, bool>** accepts two parameters of type **int** and **string** and returns a value of type **bool**.

4

4

THE ACTION DELEGATE

Action<> is a generic delegate which accepts zero or more input parameters and does **not** return a value.

There are many overloads:

- **Action** : Accepts zero parameters and does not return a value
- **Action<T>** : Accepts one parameter and does not return a value
- **Action<T1, T2>** : Accepts two parameters and does not return a value

Example: **Action<int, string, bool>** accepts three parameters of type **int, string,** and **bool** and does not return a value.

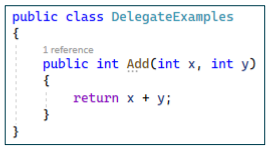
5

5

FUNC EXAMPLE

The **Add** method accepts *two* **int** parameters and has a return type of **int**.

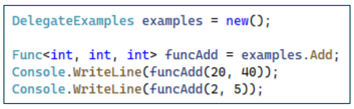
It is an instance method (not static).



```
public class DelegateExamples
{
    1 reference
    public int Add(int x, int y)
    {
        return x + y;
    }
}
```

You can instantiate an object instance and use the **Func** generic delegate to encapsulate the **Add** method.

You invoke the **Add** method by invoking the **delegate** and passing the required parameters.



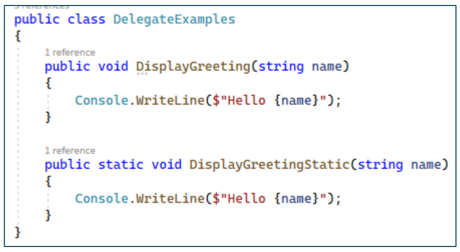
```
DelegateExamples examples = new();

Func<int, int, int> funcAdd = examples.Add;
Console.WriteLine(funcAdd(20, 40));
Console.WriteLine(funcAdd(2, 5));
```

6

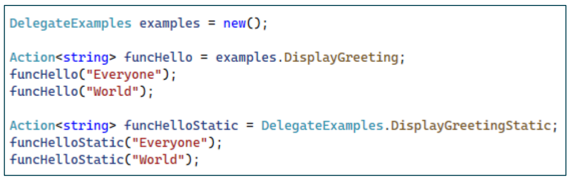
6

ACTION EXAMPLE

Delegates can encapsulate static or instance methods:



```
public class DelegateExamples
{
    1 reference
    public void DisplayGreeting(string name)
    {
        Console.WriteLine($"Hello {name}");
    }

    1 reference
    public static void DisplayGreetingStatic(string name)
    {
        Console.WriteLine($"Hello {name}");
    }
}
```

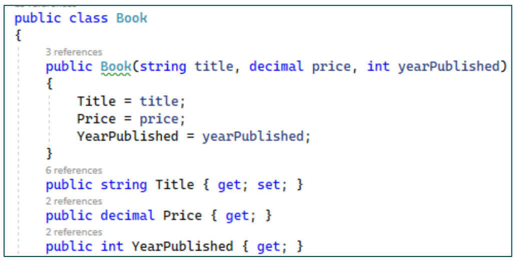


```
DelegateExamples examples = new();

Action<string> funcHello = examples.DisplayGreeting;
funcHello("Everyone");
funcHello("World");

Action<string> funcHelloStatic = DelegateExamples.DisplayGreetingStatic;
funcHelloStatic("Everyone");
funcHelloStatic("World");
```

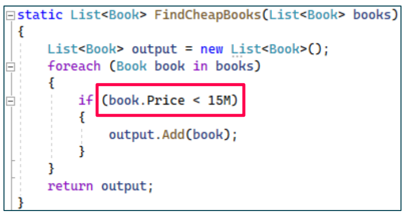
7

7

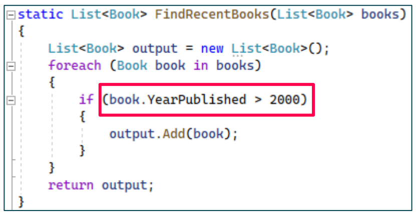
EXAMPLE PART 1



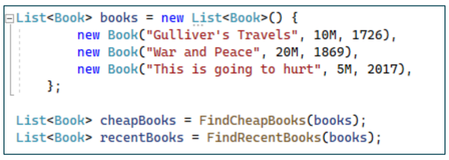
```
public class Book
{
    3 references
    public Book(string title, decimal price, int yearPublished)
    {
        Title = title;
        Price = price;
        YearPublished = yearPublished;
    }
    6 references
    public string Title { get; set; }
    2 references
    public decimal Price { get; }
    2 references
    public int YearPublished { get; }
```



```
static List<Book> FindCheapBooks(List<Book> books)
{
    List<Book> output = new List<Book>();
    foreach (Book book in books)
    {
        if (book.Price < 15M)
        {
            output.Add(book);
        }
    }
    return output;
}
```



```
static List<Book> FindRecentBooks(List<Book> books)
{
    List<Book> output = new List<Book>();
    foreach (Book book in books)
    {
        if (book.YearPublished > 2000)
        {
            output.Add(book);
        }
    }
    return output;
}
```

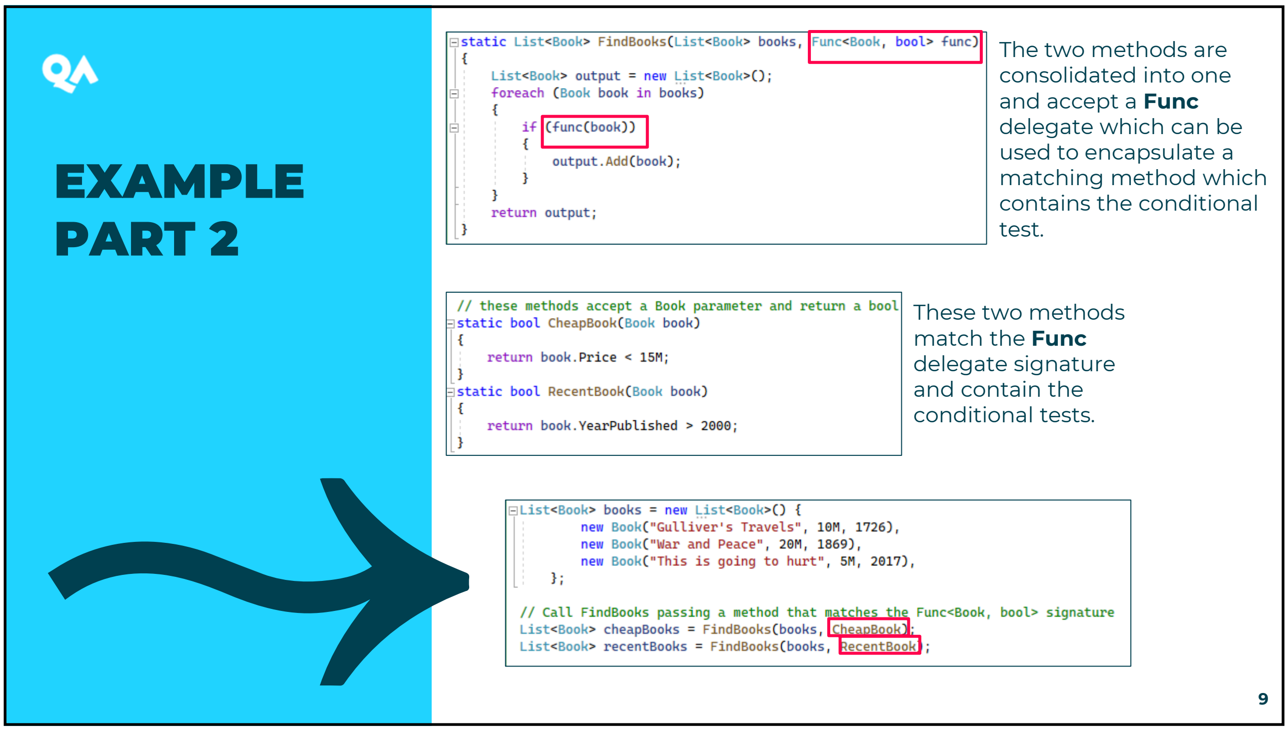


```
List<Book> books = new List<Book>() {
        new Book("Gulliver's Travels", 10M, 1726),
        new Book("War and Peace", 20M, 1869),
        new Book("This is going to hurt", 5M, 2017),
    };

List<Book> cheapBooks = FindCheapBooks(books);
List<Book> recentBooks = FindRecentBooks(books);
```

The two methods are almost identical and only differ by the conditional test

8

8



EXAMPLE PART 2

```
static List<Book> FindBooks(List<Book> books, Func<Book, bool> func)
{
    List<Book> output = new List<Book>();
    foreach (Book book in books)
    {
        if (func(book))
        {
            output.Add(book);
        }
    }
    return output;
}
```

The two methods are consolidated into one and accept a **Func** delegate which can be used to encapsulate a matching method which contains the conditional test.

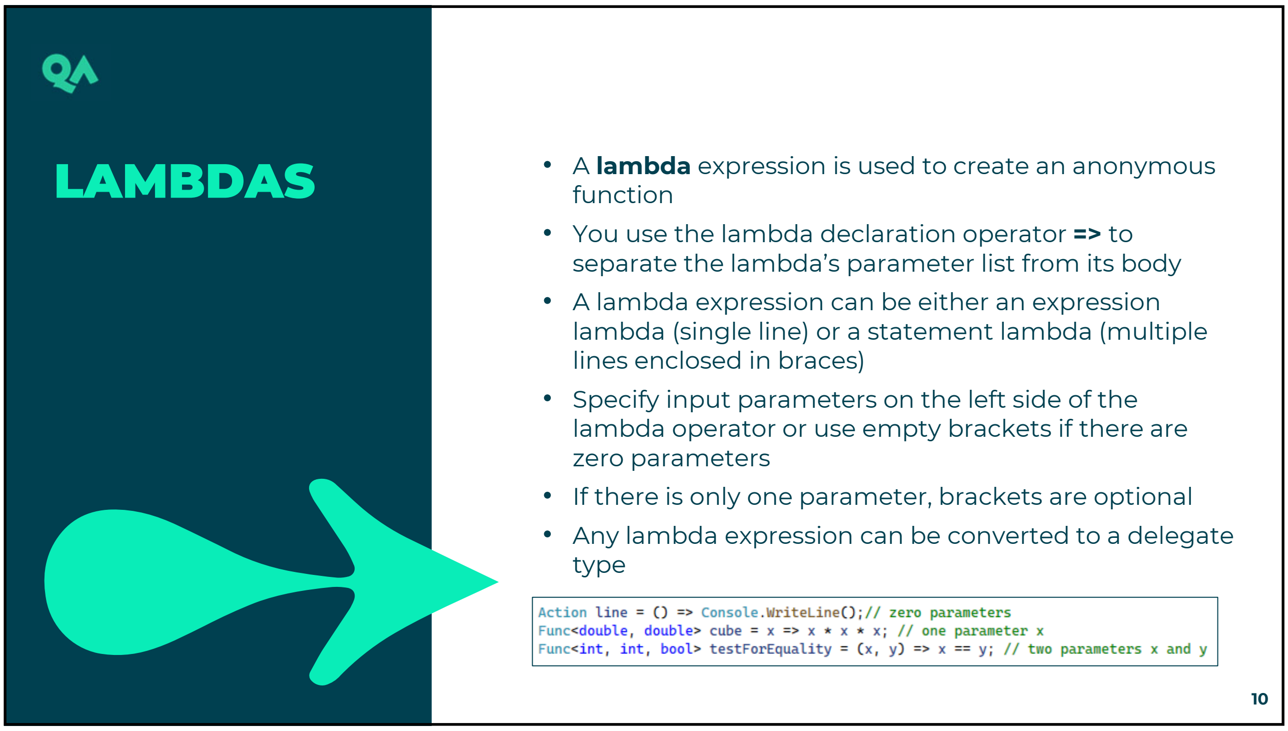
```
// these methods accept a Book parameter and return a bool
static bool CheapBook(Book book)
{
    return book.Price < 15M;
}
static bool RecentBook(Book book)
{
    return book.YearPublished > 2000;
}
```

These two methods match the **Func** delegate signature and contain the conditional tests.

```
List<Book> books = new List<Book>() {
        new Book("Gulliver's Travels", 10M, 1726),
        new Book("War and Peace", 20M, 1869),
        new Book("This is going to hurt", 5M, 2017),
    };

// Call FindBooks passing a method that matches the Func<Book, bool> signature
List<Book> cheapBooks = FindBooks(books, CheapBook);
List<Book> recentBooks = FindBooks(books, RecentBook);
```

9

9



LAMBDAS

- A **lambda** expression is used to create an anonymous function
- You use the lambda declaration operator **=>** to separate the lambda's parameter list from its body
- A lambda expression can be either an expression lambda (single line) or a statement lambda (multiple lines enclosed in braces)
- Specify input parameters on the left side of the lambda operator or use empty brackets if there are zero parameters
- If there is only one parameter, brackets are optional
- Any lambda expression can be converted to a delegate type

```
Action line = () => Console.WriteLine();// zero parameters
Func<double, double> cube = x => x * x * x; // one parameter x
Func<int, int, bool> testForEquality = (x, y) => x == y; // two parameters x and y
```

10

10

FUNC EXAMPLE AS A LAMBDA

This example uses a **delegate** that encapsulates a *named* Add method.

```
public class DelegateExamples
{
    1 reference
    public int Add(int x, int y)
    {
        return x + y;
    }
}
```

```
DelegateExamples examples = new();

Func<int, int, int> funcAdd = examples.Add;
Console.WriteLine(funcAdd(20, 40));
Console.WriteLine(funcAdd(2, 5));
```

This example uses a **lambda** expression to encapsulate an *anonymous* method.

```
Func<int, int, int> add = (x, y) => x + y;
Console.WriteLine(add(20, 40));
Console.WriteLine(add(2, 5));
```

ACTION EXAMPLE AS A LAMBDA

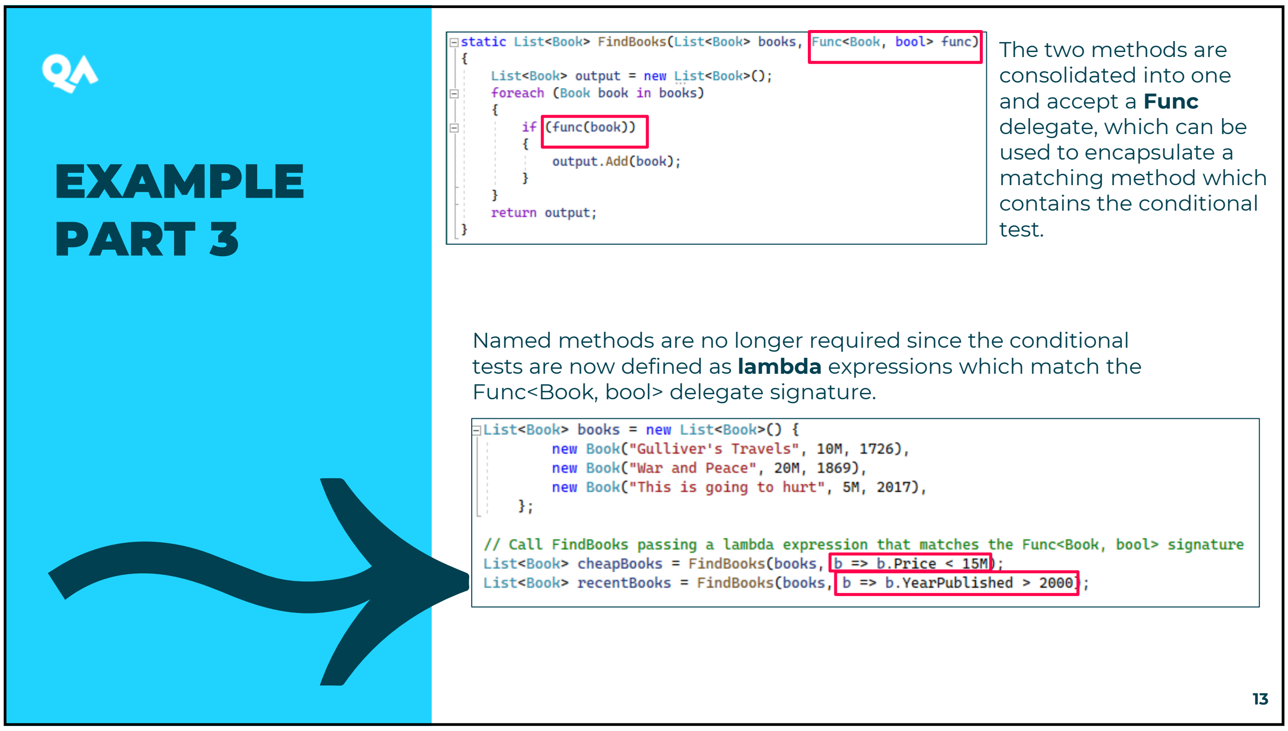
These examples use **delegates** that encapsulate *named* methods:

```
public class DelegateExamples
{
    1 reference
    public void DisplayGreeting(string name)
    {
        Console.WriteLine($"Hello {name}");
    }

    1 reference
    public static void DisplayGreetingStatic(string name)
    {
        Console.WriteLine($"Hello {name}");
    }
}
```

```
DelegateExamples examples = new();

Action<string> funcHello = examples.DisplayGreeting;
funcHello("Everyone");
funcHello("World");

Action<string> funcHelloStatic = DelegateExamples.DisplayGreetingStatic;
funcHelloStatic("Everyone");
funcHelloStatic("World");
```

These examples use **lambda** expressions to encapsulate *anonymous* methods:

```
Action<string> greet = (string name) => Console.WriteLine($"Hello {name}");
greet("Everyone");

Action<string> greet2 = name => Console.WriteLine($"Hello {name}");
greet2("World");
```



EXAMPLE PART 3

```
static List<Book> FindBooks(List<Book> books, Func<Book, bool> func)
{
    List<Book> output = new List<Book>();
    foreach (Book book in books)
    {
        if (func(book))
        {
            output.Add(book);
        }
    }
    return output;
}
```

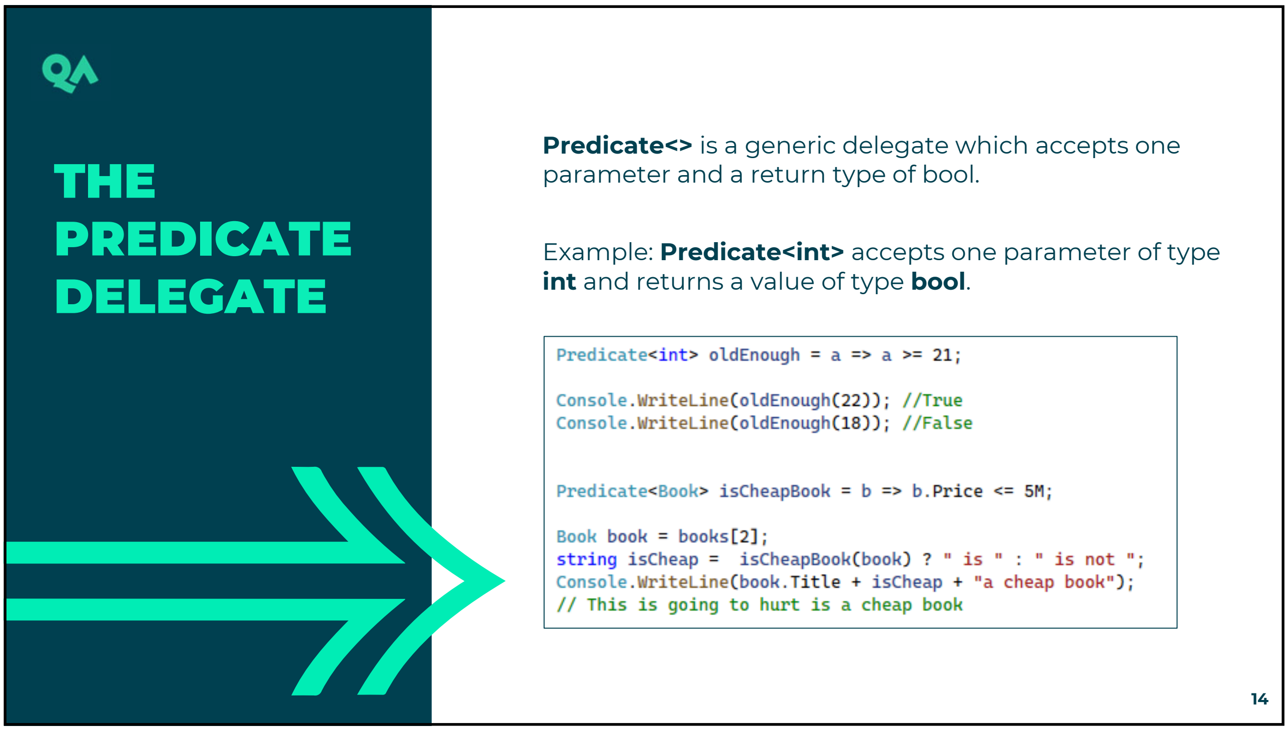
The two methods are consolidated into one and accept a **Func** delegate, which can be used to encapsulate a matching method which contains the conditional test.

Named methods are no longer required since the conditional tests are now defined as **lambda** expressions which match the Func<Book, bool> delegate signature.

```
List<Book> books = new List<Book>() {
        new Book("Gulliver's Travels", 10M, 1726),
        new Book("War and Peace", 20M, 1869),
        new Book("This is going to hurt", 5M, 2017),
    };

// Call FindBooks passing a lambda expression that matches the Func<Book, bool> signature
List<Book> cheapBooks = FindBooks(books, b => b.Price < 15M);
List<Book> recentBooks = FindBooks(books, b => b.YearPublished > 2000);
```

13

13



THE PREDICATE DELEGATE

Predicate<> is a generic delegate which accepts one parameter and a return type of bool.

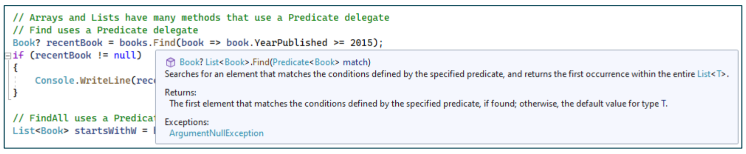
Example: **Predicate<int>** accepts one parameter of type **int** and returns a value of type **bool**.

```
Predicate<int> oldEnough = a => a >= 21;

Console.WriteLine(oldEnough(22)); //True
Console.WriteLine(oldEnough(18)); //False


Predicate<Book> isCheapBook = b => b.Price <= 5M;

Book book = books[2];
string isCheap =  isCheapBook(book) ? " is " : " is not ";
Console.WriteLine(book.Title + isCheap + "a cheap book");
// This is going to hurt is a cheap book
```

14

14

DELEGATE/ LAMBDA EXAMPLES

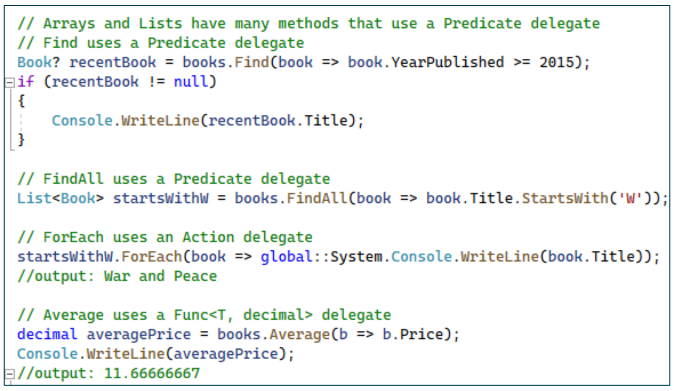


```
// Arrays and Lists have many methods that use a Predicate delegate
// Find uses a Predicate delegate
Book? recentBook = books.Find(book => book.YearPublished >= 2015);
if (recentBook != null)
{
    Console.WriteLine(rec
}

// FindAll uses a Predicat
List<Book> startsWithW = 
```

Book? List<Book>.Find(Predicate<Book> match)
Searches for an element that matches the conditions defined by the specified predicate, and returns the first occurrence within the entire List<T>.

Returns:
The first element that matches the conditions defined by the specified predicate, if found; otherwise, the default value for type T.

Exceptions:
ArgumentNullException



```
// Arrays and Lists have many methods that use a Predicate delegate
// Find uses a Predicate delegate
Book? recentBook = books.Find(book => book.YearPublished >= 2015);
if (recentBook != null)
{
    Console.WriteLine(recentBook.Title);
}

// FindAll uses a Predicate delegate
List<Book> startsWithW = books.FindAll(book => book.Title.StartsWith('W'));

// ForEach uses an Action delegate
startsWithW.ForEach(book => global::System.Console.WriteLine(book.Title));
//output: War and Peace

// Average uses a Func<T, decimal> delegate
decimal averagePrice = books.Average(b => b.Price);
Console.WriteLine(averagePrice);
//output: 11.66666667
```

15

15

SUMMARY

- Delegates
- The Func delegate
- The Action delegate
- Lambda expressions
- The Predicate delegate
- Delegate and Lambda examples

16

Activity:
Exercise 11

17

17