Delegates

C# Programming

In this chapter we will learn ...

- What is a delegate
- How delegates can be used
- How to create a delegate reference
- How to define a lambda function

Delegates

Question

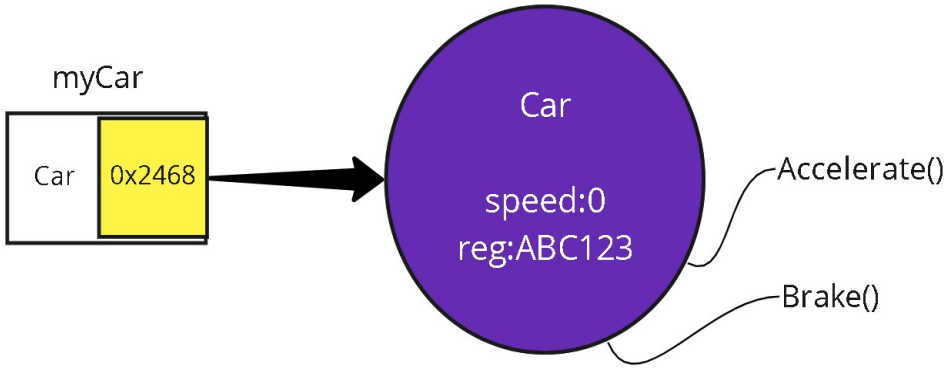
What is a delegate?

First - A Recap

We use object references to call/access methods on objects

- Object reference is just a clever pointer to the object

```
Car myCar = new Car();
myCar.Accelerate();
```



So what's a Delegate?

A delegate is another kind of object reference but ...

- It doesn't point to objects derived from classes

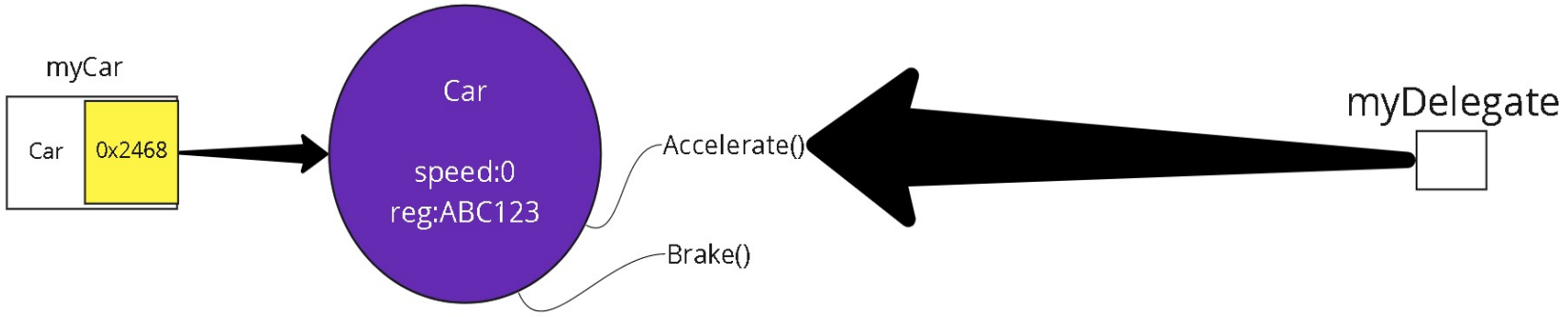
Instead delegate references point to any of...

- Static function
- Method on an object
- Global function
- Lambda Function

Delegates - Under the hood

```
Car myCar = new Car();
Action myDelegate = myCar.Accelerate; // Get reference to Accelerate method of Car
myDelegate.Invoke(); // Invoke method pointed to by delegate
```

Delegate Reference points to the method not the object!



What's the point?

It's executable

- A delegate reference can be invoked which executes the function it points to

It's focussed

- A delegate reference points to a single block of code/function only
- Object references give access to all public methods of an object

It's transportable

- Delegate references can be passed as arguments to functions
- Receiving function can invoke the delegate

No .. Really .. What's the Point??

Well ... because

- a delegate references a block of code to be executed

and ...

- a delegate reference can be passed as an argument to another function
- The receiving function can invoke the delegate which invokes the code

That means ..

- As well as passing data as arguments to functions
- We can pass functions to functions via delegates == POWERFUL + FLEXIBLE

Understanding Delegates

So we know - A delegate is a reference to some code

- Delegate references have a type
- Custom delegate types can be defined using the **delegate** keyword
- You can define the delegate type based on
  - Parameter signature of function you want to reference
  - Return type of function you want to reference

```
delegate <return type> <delegate type name> (<parameter type> <param name>, ...)
```

Delegate Example

Based on the following function

```
void SayHello(string name) {
    Console.WriteLine($"Hello {name}");
}
```

A delegate declaration to reference it would be

```
delegate void MyDelegateType (string p1);
MyDelegateType dlg = SayHello;
dlg.Invoke("Mina");
```

Hello Mina

Generic Delegates

Microsoft have defined Generic Delegates to make life easier

- Avoid declaring new delegate types for every function
- Enables more loosely coupled function definitions

Three main Generic Delegates (can be used with up-to 16 parameters)

- Action - for void methods
- Func - for none void methods
- Predicate - for bool returning filter delegates with on parameter

Action Delegate Type

Action Delegates

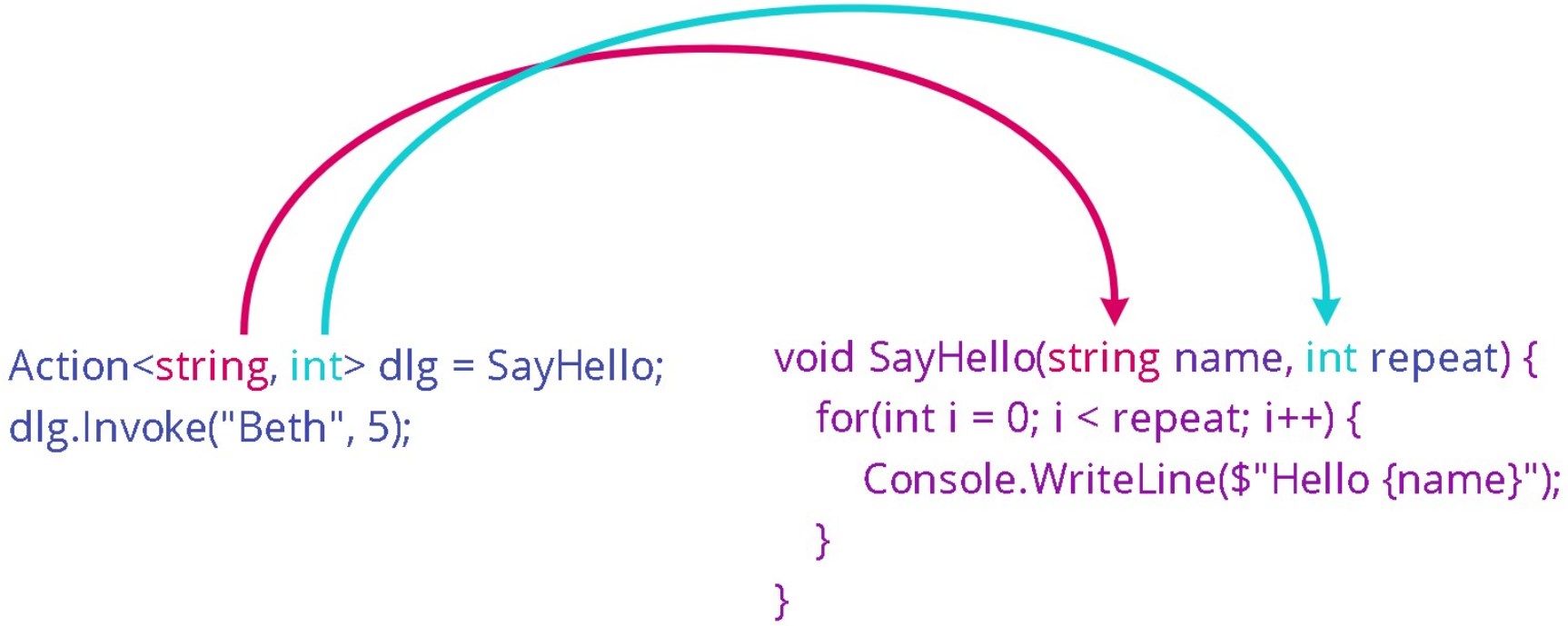
- Reference any **void** method
- Method can have from 0 to 16 generic typed parameters

```
void SayHello(string name) {
    Console.WriteLine($"Hello {name}");
}
Action<string> dlg = SayHello; dlg.Invoke("Beth");
```

Hello Beth

Hooking Up Action Delegates

Generic Parameters match up to method parameters



- 0 to 16 generic params allowed

Passing a function to a function via a delegate

Framework List class has a useful method .ForEach(Action<T> item)

- Invokes a passed in function argument for every element in list
- Expects a Generic Action delegate as an argument

```
void SayHello(string name) {
    Console.WriteLine($"Hi {name}");
}
List<string> friends = new List<string> () {
    "Fred", "Tam", "Suzy", "Kurt", "Mina", "Beth"
};

Action<string> dlg = SayHello; // Create delegate reference to SayHello function
friends.ForEach(dlg); // ForEach element in List delegate will be invoked
```

Acceptable Syntax Variations

Delegates can be declared in a more formal syntax

```
Action<string> dlg = new Action<string>(SayHello); // Is often simplified to
Action<string> dlg = SayHello;
```

It's also possible to avoid declaring an explicit delegate reference

```
// Instead of
Action<string> dlg = SayHello;
friends.ForEach(dlg);
// Just Write
friends.ForEach(SayHello); // Compiler joins the dots
```

Generic Predicates

Predicate delegates can reference

- Methods that return bool
- Have a single generic parameter

```
bool ShouldInclude(string name) {
    return name.Length > 3;
}
Predicate<string> dlg = ShouldInclude; // Create delegate reference to ShouldInclude function
Console.WriteLine(dlg.Invoke("Fred"));
Console.WriteLine(dlg.Invoke("Bob"));
```

True

False

Using a Generic Predicate for real

Framework List class has a predicate method .FindAll(Predicate<T> item)

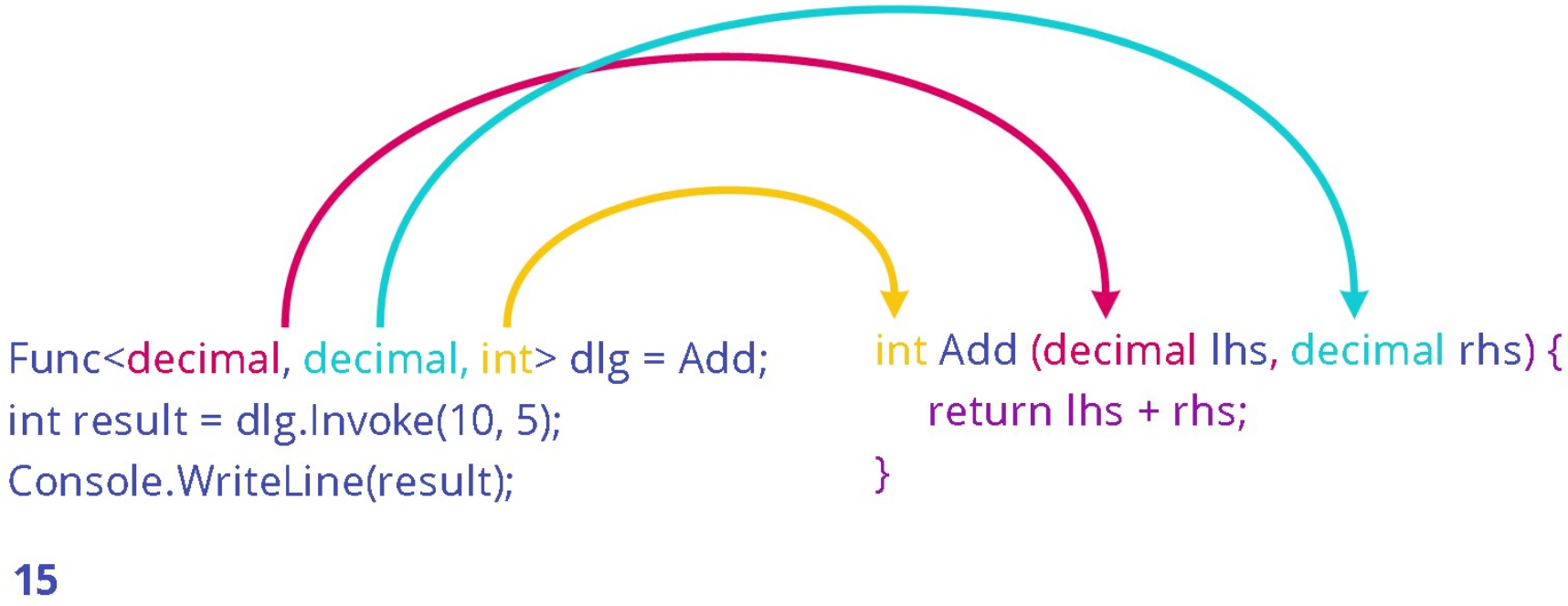
- Filters the List based on passed in predicate
- Invokes passed in function argument for every element in list
- If function returns True for element it is included in return list

```
List<string> friends = new List<string> () {
    "Fred", "Tam", "Suzy", "Kurt", "Mina", "Bob"
};

bool ShouldInclude(string name) {
    return name.Length > 3;
}
Predicate<string> dlg = ShouldInclude; // Create delegate reference to ShouldInclude function
var filteredList = friends.FindAll(dlg); // Tam and Bob aren't included in the filteredList
```

Func Delegate

Used to reference any function that isn't void



- Return Type declared as last Generic type argument in Func declaration

Using a Func Delegate

Here we use List's Sum(Func<T, TReturn?>)

```
List<string> friends = new List<string>() {
    "Fred", "Tam", "Suzy", "Kurt", "Mina", "Bob"
};

decimal? NameLength(string name)
{
    return name.Length;
}
decimal? lengthOfNames = friends.Sum(NameLength);
Console.WriteLine(lengthOfNames);
```

22

Why are delegates useful

We can write functions that allow code to be passed in and invoked

- Receiving function has no knowledge of what type the code is part of
- It just knows it can invoke the code and it's method signature
- Allows us to write hybrid functions
    - Our code does some of the work
    - We also invoke the code passed to us

Makes our code more re-usable

- Helps decouple the code we write from the code that calls it

Lambda Functions

Lambda Functions

What is a Lambda Function/Expression?

- Lambdas are concise functions that
  - are anonymous
  - have implicit parameter types
  - Often have concise parameter names - brevity is king!
- Lambda Expressions are defined on a single line
  - return statement is implicit and its use is invalid
- Lambda Statements are defined in one or more lines and use { }
  - return statement must be explicitly called

Lambda Syntax

Lambda Expression

(<parameters>,) => <Code to Execute>

```
(x, y) => x + y; // Single line with implicit return statment
```

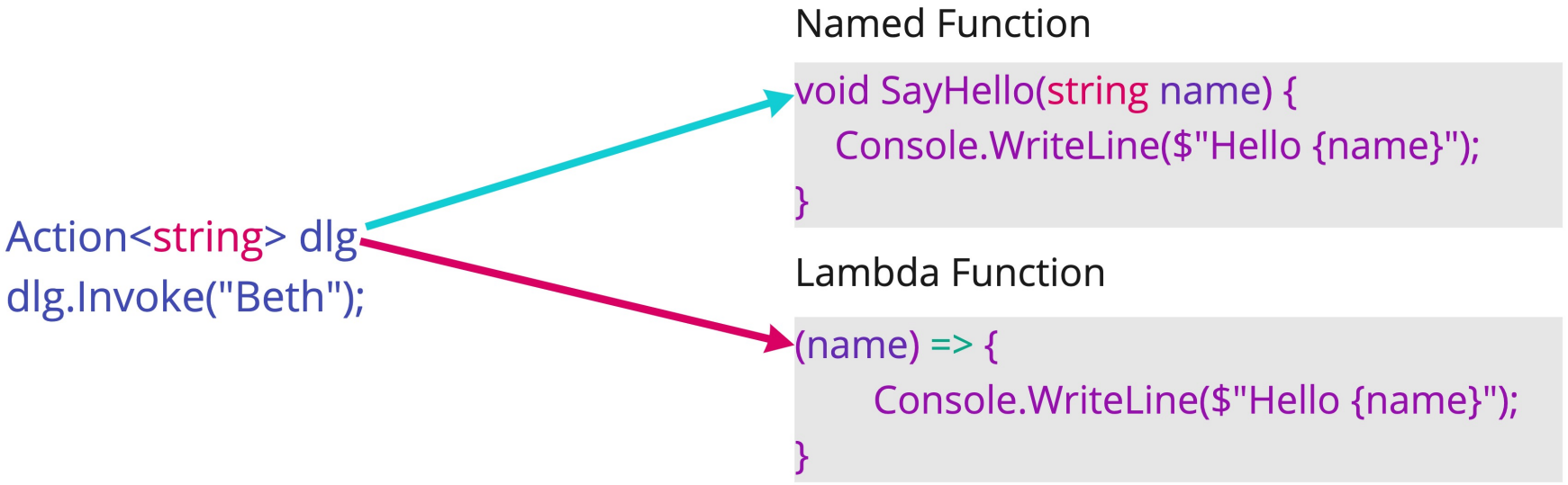
Lambda Statement

(<parameters>,) => { <Code to Execute> }

```
(x, y) => {
    return x + y; // Multline with explicit return statement
}
```

Lambdas can be assigned to delegate references

Action delegate can reference either named or lambda variants



- Lambda is a condensed form of the named function

Example - Without Lambda

```
List<string> friends = new List<string> () {
    "Fred", "Tam", "Suzy", "Kurt", "Mina", "Beth"
};

void SayHello(string name) {
    Console.WriteLine($"Hi {name}");
}

Action<string> dlg = SayHello; // Create delegate reference to SayHello function
friends.ForEach(dlg); // ForEach
```

Example - With Lambda Function

```
List<string> friends = new List<string> () {
    "Fred", "Tam", "Suzy", "Kurt", "Mina", "Beth"
};

Action<string> dlg = (name) => {
    Console.WriteLine($"Hi {name}");
};
friends.ForEach(dlg);
// OR - Could be written concisely as one line lambda expression with no {}
friends.ForEach((name) => Console.WriteLine($"Hi {name}"));
```

Anatomy of a Lambda

The following lambda expression

- Uses a Func delegate to act as a reference to the expression
- Multiplies the two input parameters together
- Returns result implicitly

```
Func<decimal, int, byte> op = (a, b) => a * b;
Console.WriteLine(op(10, 5));
```

50

Parameterless Expressions

How do we define a parameterless expression?

- Just use empty parentheses ()

```
Action dlg = () => Console.WriteLine("Hello Jerry!");
dlg.Invoke();
```

Async Delegate

Mark the Lambda as async

```
public partial class Form1 : Form {
    public Form1() {
        InitializeComponent();
        button1.Click += async (sender, e) => {
            await DoSomething();
            messageText.Text = "I'm back! ...";
        };
    }
    private async Task DoSomething() {
        await Task.Delay(5000); // Simulate something taking a long time
    }
}
```

In this chapter we learned ...

- What is a delegate
- How delegates can be used
- How to create a delegate reference
- How to define a lambda function