Unit 1

Delegates

Chapter 6

Objectives

- Define and use simple delegates

What is Delegate?

- A delegate is a **type** that enables you to store **references** to functions/methods. That is, a delegate can ***refer to*** a function/method in memory
- A delegate is also known as a function/method **pointer**. That is, a delegate can ***point to*** a function/method in memory
- A delegate can be **declared** using the **`delegate`** keyword followed by a function/method **signature**
- Recall the **definition** of a function/method **signature**: *The name and parameters of a function/method (but not its return type) collectively define the signature of a function/method*
- Delegates are used in, for example, events and event handling

How to declare a Delegate?

- The delegate syntax is as follows:
  ```
  <access modifier> delegate <return type>
     <delegate_name>(<parameters>);
  ```
- *access modifier* can be: **`public`**, **`private`**, **`protected`** or **`internal`** (default)
- **`delegate`** is the **required C# keyword**
- *return type* can be any valid return type, or **`void`**
- *delegate_name* is the name of the delegate, and the same rules for naming a function/method is followed when naming a delegate
- *parameters* is the parameter list containing 0 or more function/method parameters, depending on the requirements. **Note:** just like when declaring functions/methods, even if no function/method parameters are needed, the **parentheses ()** are **required**
- **Note:** unlike a function/method declaration, a delegate **does not** have a body, thus the **semicolon ;** is **required**

How to declare a Delegate?

- Consider the following example:

```
delegate double ProcessDelegate(double param1, double param2);
```

- Here, a delegate called ***`ProcessDelegate`*** is declared. The return type is ***`double`*** and there are 2 function/method parameters of type ***`double`***. **Again, note there is no body, only the semicolon after the parentheses**
- A delegate is declared inside a class, but not inside a function/method

How to use a Delegate?

- After defining a delegate, a variable can be declared whose **type** is of the **delegate type**
- The variable can then be initialized as a **reference** to any **function/method** that has the **same return type and parameter list**
- Now, the delegate variable can be used to call/invoke the function/method it is referencing/pointing to

```
static double Multiply(double param1, double param2) => param1 * param2;
static double Divide(double param1, double param2) => param1 / param2;
```

- The two methods above both have the **same return type and parameter list** as that of the delegate ***`ProcessDelegate`***

How to use a Delegate?

- A variable of type **`ProcessDelegate`** can now be declared:

```
ProcessDelegate process;
```

- Variable **`process`** can now refer/point to any function/method with the same return type and parameter list as that of **`ProcessDelegate`**

```
process = Multiply;
```

- The above code makes **`process`** refer/point to function/method **`Multiply`**. Note that there are no parentheses and thus no parameters
- It is important to understand that the above assignment simply means: *Let variable **`process`** refer to (point to) method **`Multiply`***. It does **NOT** invoke/execute method **`Multiply`**

How to use a Delegate?

- Following the same approach, **`process`** can also refer/point to **`Divide`**, since **`Divide`** also has the same return type and parameter list:

```
process = Divide;
```

- **Optionally**, a delegate object can be created using the **`new`** operator and specifying a method name:

```
process = new ProcessDelegate(Multiply);
process = new ProcessDelegate(Divide);
```

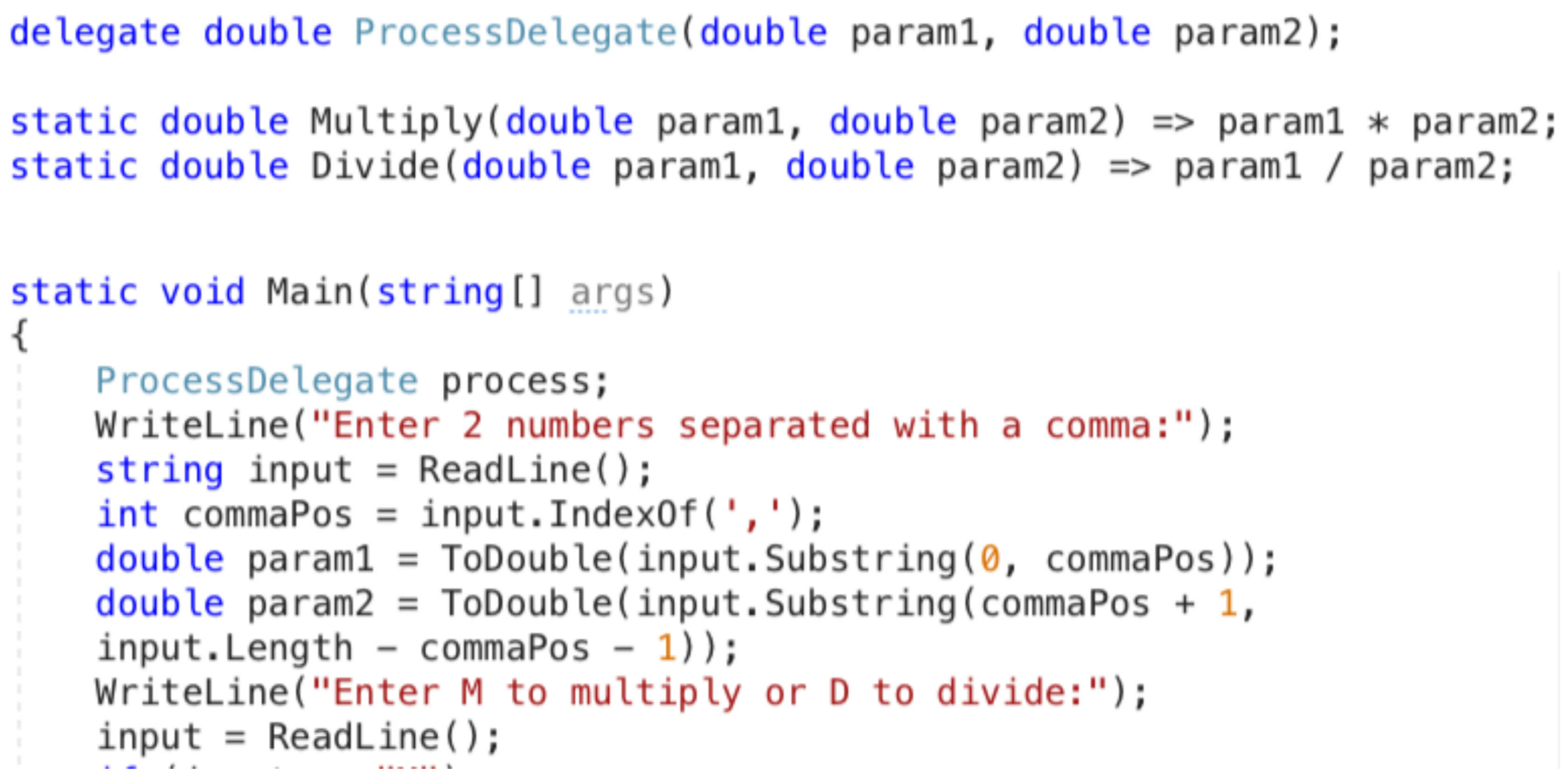
- Variable **`process`** can then be used to call/invoke the function/method it is referring/pointing to

```
double result = process(10, 10);
```

How to use a Delegate?

- Notice that the same rules/conventions are used to call/invoke a function/method a delegate variable is referring/pointing to, as those used when calling/invoking any other function/method:
  - The parameter list used in the call must match that of the function/method declaration (same number of parameters, same sequence and same types)
  - If the function/method return type is not void, the returned value must be assigned to a variable, or used/evaluated using some other form of logic

Try It Out – Ch06Ex05

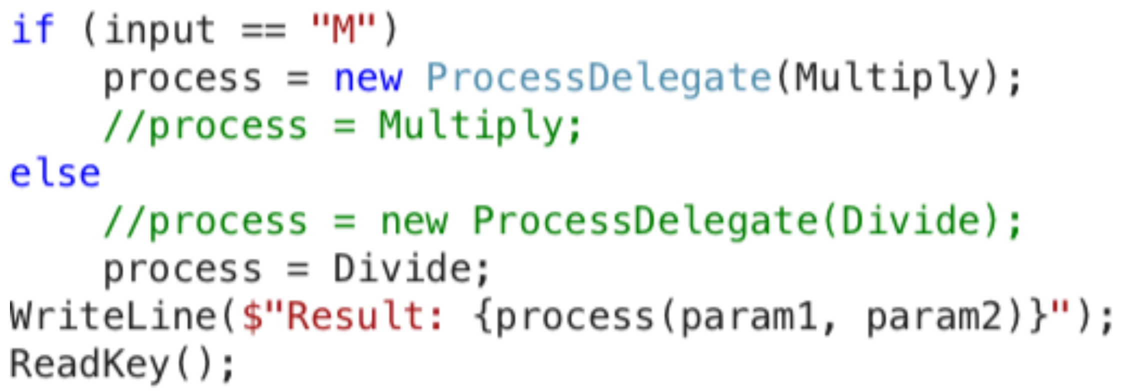


```
delegate double ProcessDelegate(double param1, double param2);

static double Multiply(double param1, double param2) => param1 * param2;
static double Divide(double param1, double param2) => param1 / param2;


static void Main(string[] args)
{
    ProcessDelegate process;
    WriteLine("Enter 2 numbers separated with a comma:");
    string input = ReadLine();
    int commaPos = input.IndexOf(',');
    double param1 = ToDouble(input.Substring(0, commaPos));
    double param2 = ToDouble(input.Substring(commaPos + 1,
    input.Length - commaPos - 1));
    WriteLine("Enter M to multiply or D to divide:");
    input = ReadLine();
```

Try It Out – Ch06Ex05



```
if (input == "M")
    process = new ProcessDelegate(Multiply);
    //process = Multiply;
else
    //process = new ProcessDelegate(Divide);
    process = Divide;
WriteLine($"Result: {process(param1, param2)}");
ReadKey();
```

- Notice that **`process`** is **ONLY** called/invoked **ONCE**, and only **AFTER** **`process`** was assigned a valid reference in the **`if`** statement

Summary

- As well as calling functions directly, it is possible to call them through delegates
- Delegates are variables that are defined with a return type and parameter list
- A given delegate type can match any method whose return type and parameters match the delegate definition

Exercise – Modify Try It Out Ch06Ex05

- Modify the program to no longer use a comma as a separator, but use a semicolon instead
- Add the following two methods, each one must have the same return type and method parameters as the existing two methods:
  - **`Add`** – add the two parameters together and return the result
  - **`Subtract`** – subtract the 2nd parameter from the 1st one and return the result
- The 2nd **`WriteLine`** must now display the following message:
  - *Enter M to multiply, D to divide, A to add, or S to subtract the 2nd number from the 1st number:*
- Replace the **`if`** statement with a correct **`switch`** statement to test for the 4 valid options. Remember to do a case-sensitive test
- If an invalid option is selected, **multiply** must be assumed

Exercise – Modify Try It Out Ch06Ex05

- Each **`switch case`** must print a meaningful message. Assume the user entered *12,10* then the corresponding messages must be:
  - Multiply: *12 x 10 =*
  - Divide: *12 / 10 =*
  - Add: *12 + 10 =*
  - Subtract: *12 – 10 =*
- **`Write`** must be used, and not **`WriteLine`** for the above messages
- If an invalid option was selected, **multiply** must be assumed, and the message must be:
  - *Invalid selection, multiply assumed... 12 x 10 =*