Lab Using Delegates

In the following exercise, you will **create a delegate** to **encapsulate a method** that displays the **time in a text box** acting as a digital clock on **a WPF form**. You will attach the delegate object to a class called *Ticker* that **invokes the delegate** every second..

Finish the digital clock application. Complete the digital clock application

1. **Start** Microsoft **Visual Studio** if it is not already running.
2. **Open** the **Delegates** project

3. On the ***Debug*** menu, click ***Start Without Debugging***.
   The project builds and runs. A form appears, **displaying a digital clock**. The clock displays the **current time** as "***00:00:00***," which is probably wrong unless you happen to be reading this lab at midnight.
4. **Click** ***Start*** to start the clock, and then **click** ***Stop*** to **stop** it again.
   Nothing happens. The *Start* and ***Stop*** methods have not been written yet. Your task is to **implement these methods**.
5. **Close** the form, and **return** to the **Visual Studio** environment.
6. **Open** the ***Ticker.cs*** fi le, and display it in the ***Code*** *and* ***Text Editor*** window. This file contains a ***class*** called ***Ticker*** that **models the inner workings of a clock**.
   Scroll to the bottom of the file. The class contains a ***DispatcherTimer*** **object** called *ticking* to arrange for a pulse to be sent at regular intervals. The **constructor for the class sets this interval** to **1 second**. The class **catches the pulse by using an event** (you will learn how events work later)

Note The .***NET*** Framework provides another timer class called ***System.Timers.Timer***. This class offers similar functionality to the ***DispatcherTimer*** class, but it is not suitable for use in a ***WPF*** application.

7. In the ***Code*** and ***Text*** Editor window, **find** the declaration of the ***Tick*** **delegate**. It is located near the top of the file and looks like this:

```
public delegate void Tick(int hh, int mm, int ss);
```

   The ***Tick*** delegate **can be used to refer to a method** that takes three integer parameters and that does not return a value. A **delegate variable** called *tickers* **at the bottom** of the file is based on this type. **By using the** ***Add*** and ***Remove*** methods in this class (shown in the following code example), you can **add methods** with **matching signatures** to (and **remove them from**) the *tickers* delegate variable:

```
class Ticker
{
...
public void Add(Tick newMethod)
{
this.tickers += newMethod;
}
public void Remove(Tick oldMethod)
{
this.tickers -= oldMethod;
}
...
private Tick tickers;
}
```

8. **Open** the ***Clock.cs*** file, and **display** it in the ***Code*** *and* ***Text Editor*** window. The ***Clock class*** **models the clock display**. It **has methods** called ***Start*** and ***Stop*** that are used to **start and stop the clock** running (after you have implemented them) **and a method** called ***RefreshTime*** that formats a **string** to depict the time specified by its three parameters (hours, minutes, and seconds) and then displays it in the ***TextBox*** field called *display*. This ***TextBox*** field is initialized in the constructor. The class also contains a ***private Ticker*** **field** called ***pulsed*** that tells the **clock when to update** its display:

```
class Clock
{
     private Ticker pulsed = new Ticker();
     private TextBox display;
...
    public Clock(TextBox displayBox)
   {
     this.display = displayBox;
   }
    private void RefreshTime(int hh, int mm, int ss )
    {
      this.display.Text = string.Format("{0:D2}:{1:D2}:{2:D2}", hh,
                                                              mm, ss);
     }
}
```

9. **Display** the code for the ***Window1.xaml.cs*** file in the ***Code*** *and* ***Text Editor*** window. **Notice** that **the constructor creates** a ***new*** instance of the ***Clock class***, passing in the ***TextBox*** field called *digital* as its **parameter**:

```
public Window1()
{
...
    clock = new Clock(digital);
}
```

The ***digital field*** is the ***TextBox*** control **displayed on the form**. The **clock will display its output** in this ***TextBox*** control.

10. **Return** to the ***Clock.cs*** fi le. **Implement** the ***Clock.Start*** method so that it **adds** the ***Clock.RefreshTime*** **method to the delegate** in the ***pulsed*** object by using the ***Ticker.Add*** method, as follows **in bold and red type**. The *pulsed* delegate is **invoked every time** a pulse occurs, and this statement causes the ***RefreshTime*** method to execute when this happens. The ***Start*** method should look like this:

```
public void Start()
{
     pulsed.Add(this.RefreshTime);
}
```

11. **Implement** the ***Clock.Stop*** method so that it **removes** the ***Clock.RefreshTime*** method from the ***pulsed delegate*** by using the ***Ticker.Remove*** method, as **follows in bold and red type**. The ***Stop*** method should look like this:

```
public void Stop()
{
    pulsed.Remove(this.RefreshTime);
}
```

12. On the ***Debug*** menu, **click** *Start* ***Without Debugging***.
13. **On the *WPF* form, click *Start*. The form now displays the correct time and updates every second.**
14. **Click *Stop***. The display **stops** responding, or "*freezes*." This is because the ***Stop*** button calls the ***Clock.Stop*** method, which removes the ***RefreshTime*** method from the ***Ticker*** delegate; ***RefreshTime*** is ***no longer being called*** every second, although the **timer continues to pulse**

15. If you click ***Start*** **more than one time**, you must click ***Stop*** the same number of times. Each time you click ***Start*** you add a reference to the ***RefreshTime*** method to the delegate. **Use the**

invocation list of ***tickers to*** **remove all the references in this list to the method name used in** `Tick oldMethod` **b**efore the clock will ***stop***.

16. **Click** ***Start*** again.

    The display **resumes processing**, corrects the time, and updates the time every second. This is because the ***Start*** button calls the ***Clock.Start*** method, which attaches the ***RefreshTime*** method to the ***Ticker delegate*** again.

17. **Close** the form, and **return** to **Visual Studio**.