158.212 Application Software Development

Suriadi Suriadi

Lecture 6

Test-driven development (TDD)

Unit tests

Interacting with the System Environment

Visual Studio 2012 IDE Windows Forms controls

- Menus
- Toolbars
- ToolStrips

Assignment 3

Software Development Methodologies

- There are many different ways to develop application software
    - Generally, either ad-hoc or via a particular methodology, eg. Waterfall, RAD, MDSE, Agile and its variants (XP, Scrum etc).

- All methodologies *strive to create robust and working software* and place importance on *different development requirements*

- Each methodology has an underlying assumption that defines it.

TDD – what is it?...

Test-driven development (TDD) proposed by Kent Beck in 2003.

Turns programming into a dialogue (Kerievsky, 2005):

- **Ask**: ask a question of the system by writing a test
- **Respond**: respond to the question by writing code to pass the test
- **Refine**: refine the *response* by improving code structure, weeding out inessentials, and clarifying the ambiguities (refactoring)
- **Repeat**: keep dialog going through more questions

TDD focuses on **development** rather than on testing.

TDD – why?...

- TDD encourages **simple designs** and **inspires confidence**.
- TDD **facilitates** code refactoring and incorporates it as a cornerstone practice.
- Immediate **developer feedback**.
- Errors in broken code can be easily located.

TDD – why?...

- Promotes disciplined coding practice resulting in cleaner and testable code

- TDD improves *code coverage*
    - that is, the extent to which the code has been tested.

- An industry proven methodology – **code without it often viewed as unprofessional and untrustworthy**

- Orthogonal to Object Oriented Programming.

TDD – benefits…

- Keeps defect counts low
- Testing drives the design – consequently, more modular code
- Shortens time wrestling with design questions – instead, enables quick and incremental programming of small chunks of behaviours
- Frequently removes the need for a debugger
- Produces self-documenting code
- Disadvantages?
    - … time investment, tests not always straightforward to write, UI test? false sense of correctness?

How to Do TDD-based Development

Development cycle:

1. Write the test cases for each functionality/service of your application (known as a *unit of code)*
    - The test cases should fail as no code written yet!
2. Write the actual of unit codes for your application
3. Continue testing the unit of code until it passes
4. Refactor
5. Repeat

Unit Test

- A unit test is a *non-functional* piece of code, *independent* from the core software, that verifies if a unit of code is **fit for purpose**.

- A unit is the smallest testable block of source code – usually a **single method** (functional coverage)
    - Statement coverage
    - Branching coverage, etc.

- Each unit test must be **independent** from others

- Unit tests must have the ability to be run **automatically**

- Unit tests are an **integral component** of TDD

Unit Test

Unit test components:

- Each unit test defines an **expected** value of a target function and compares it to the **actual** value
- **Assert** statements are used to compare the **expected** and **actual** values
- An **Assert** statement throws an exception when it fails, alerting the user that the unit test has found an error

Eg.

```
Assert.AreEqual(expected, actual)
Assert.AreNotEqual(expected, actual)
Assert.IsTrue(actual)
Assert.IsInstanceOfType(actual, Integer)
```

System Environment

- Frequently it is useful for programs to interact with the system environment.
- .NET provides means for interfacing with the local system environment.
- For instance, information can be gathered on the number of processors the machine has, the version of the operating system, the version of the CLR etc.

System Environment

Some useful properties include:

```
System.Environment.CurrentDirectory
System.Environment.NewLine
System.Environment.OSVersion
System.Environment.ProcessorCount
System.Environment.UserName
```

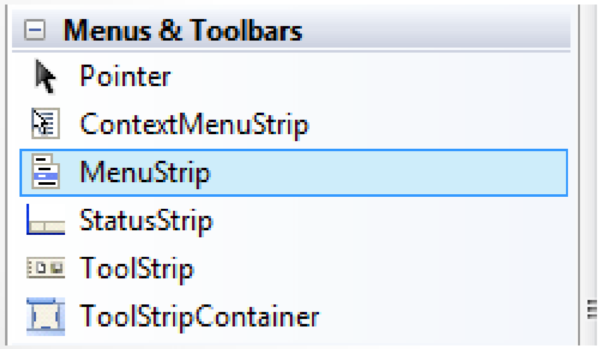
System DateTime

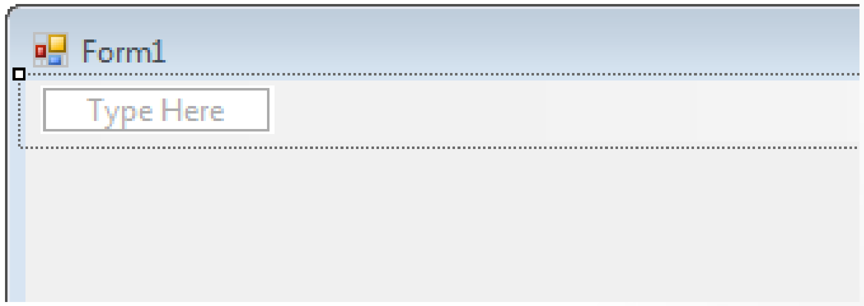
Some useful queries include:

```
System.DateTime.Now
System.DateTime.Today
```

Menus

- To add Menus to your Form, one must first add a **MenuStrip**.
- Adding this control to your application creates the menu bar along the top of the Form.

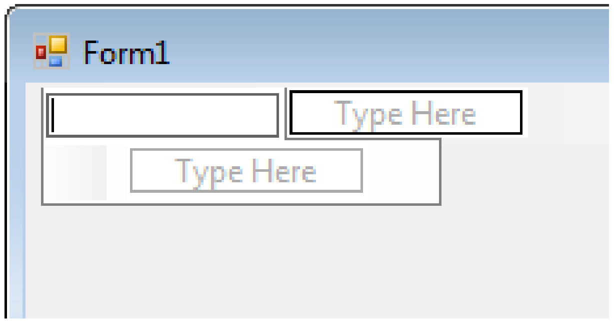




Menus

- Once a MenuStrip has been added to a Form, items can be added into the Menus by selecting the MenuStrip in the Designer and typing the desired name.

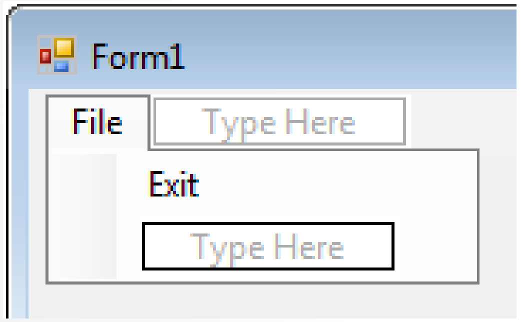
- This will add a **MenuStripItem** to the **MenuStrip**.

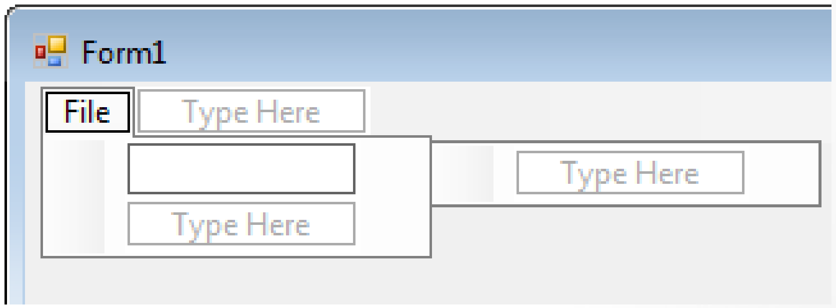


Menus

- This method can be continued to add items to these menus or other sub menus.

- Adding a simple Item to a Menu creates a MenuItem.

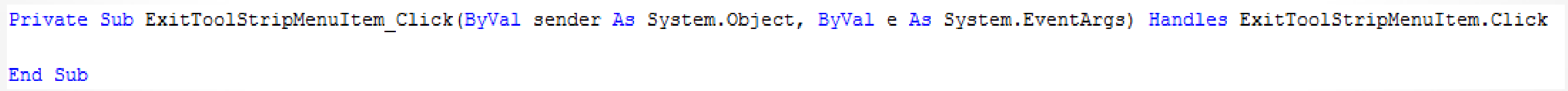




Menus

- The default event for a MenuItem is the **Click** event.

- This event will be called every time the user clicks on that MenuItem. This is very similar to the button click event.



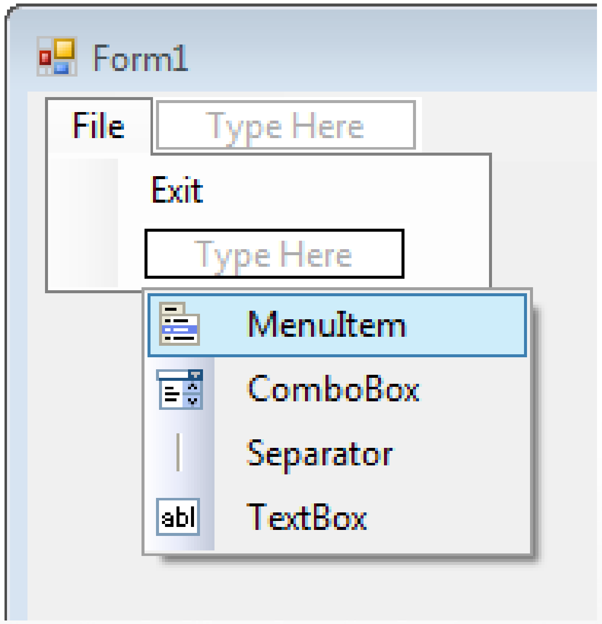
```
Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click

End Sub
```

Menus

- MenuItems are not the only things that can be added to a menu.

- By selecting the drop-down box in the Designer, a ComboBox, Separator or a TextBox can also be added.

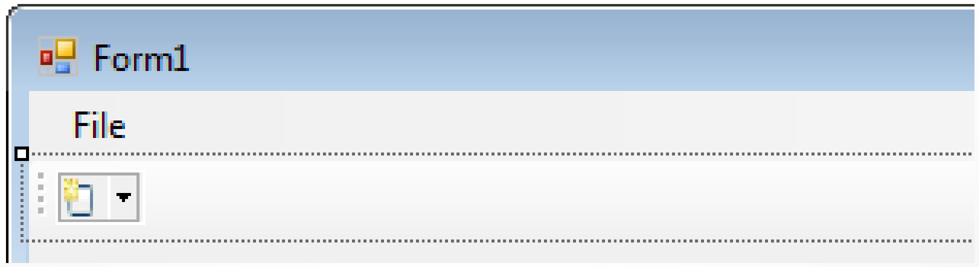


Menus

- A ComboBox in a menu allows the user to select one of several options in the Menu, exactly the same as a normal ComboBox control.
- A Separator simply displays a horizontal line to allow one to make menus easier to read and use (e.g. grouping menu items).
- A TextBox is just like a normal TextBox and allows the user to type something into it.

ToolStrip

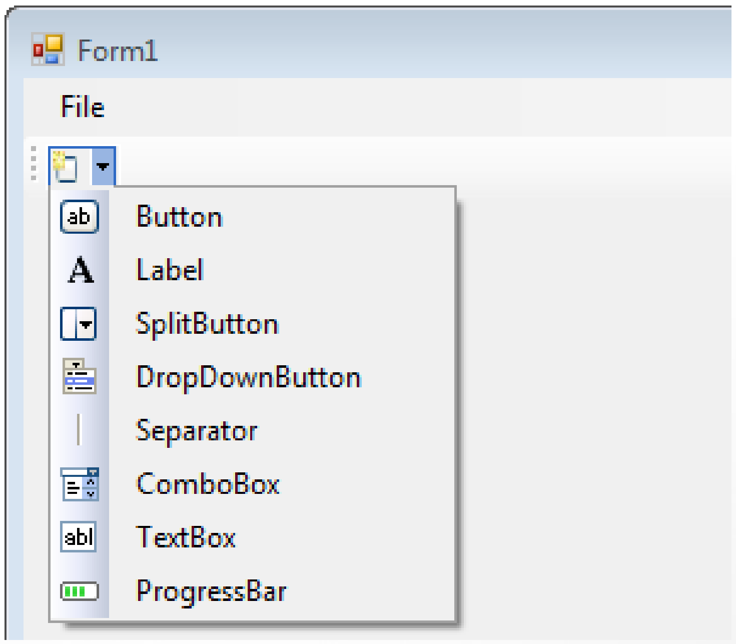
- Similar to a Menu is a ToolStrip.
- A ToolStrip is a set of buttons, drop-down boxes etc. that can be added to an application.
- While a Form can only have one Menu, it can have **many** ToolStrips.



ToolStrip

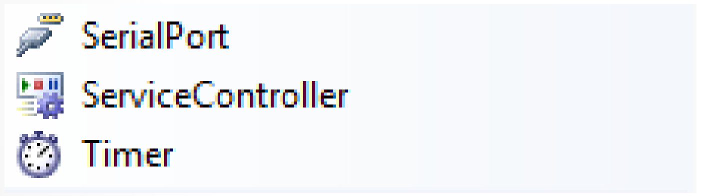
- Items can be added to a ToolStrip in the same way that items are added to a Menu.

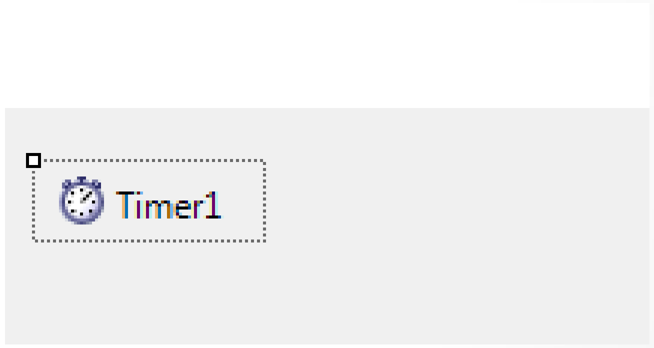
- The Visual Studio Designer makes it very easy to make Menus and ToolStrips.



Timer

The .NET class library provides a Timer class that can be added to a Form for applications that need this functionality.

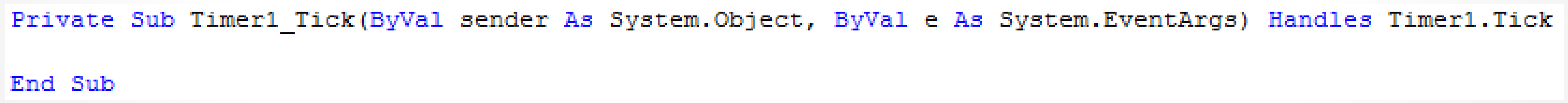




Timer

- This Timer can be started and stopped.

- An event handler for it can be created by double clicking on the Timer. This will be called every time the Timer <u>ticks</u> (default 1 second).



```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer1.Tick

End Sub
```

Timer

- Timers can be used to show animations, show the clock, measure time between files being saved etc. For instance, one could use a timer to make a stopwatch application.
- The Timer *does not keep a track of the time* but will simply call a subroutine once for every tick.
    - Commonly used for something that happens regularly (certain time interval)
- To keep track of the time we can use the **Stopwatch** class. The form can be updated with the **Stopwatch** time whenever the timer tick handler is called.

Events

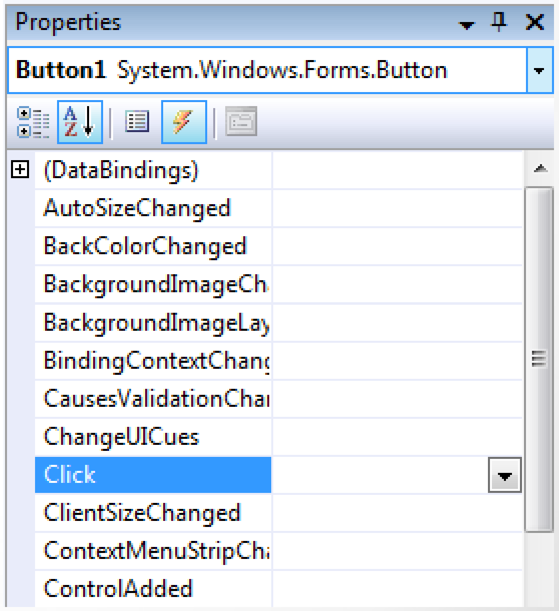
- Default event handlers are the most common and usually enough.
- Controls however have many more event handlers.
- The non-default event handlers need to be added manually instead of through the Designer.

Events

- There are dozens of events that can occur for a control like a Button.

- These events include Click, MouseDown, MouseEnter, SizeChanged etc.

- Event handlers can be added for specific events by selecting the events tab of the properties window.



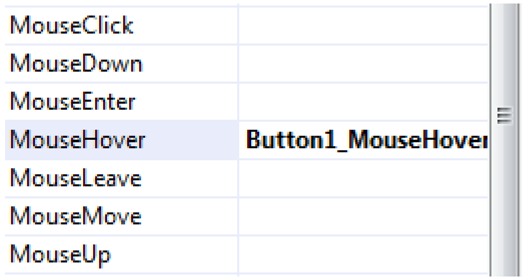
Properties

Button1 System.Windows.Forms.Button

(DataBindings)	
AutoSizeChanged	
BackColorChanged	
BackgroundImageCh	
BackgroundImageLay	
BindingContextChan	
CausesValidationCha	
ChangeUICues	
Click	
ClientSizeChanged	
ContextMenuStripCh	
ControlAdded	

Events

The tab below can be used to **associate an existing subroutine to handle an event or create a new Event Handler** by double clicking on the event.



MouseClick	
MouseDown	
MouseEnter	
MouseHover	Button1_MouseHover
MouseLeave	
MouseMove	
MouseUp	

```
Private Sub Button1_MouseHover(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.MouseHover

End Sub
```

Summary

TDD is a methodology focusing on writing tests before the target code.

TDD is a software development methodology, NOT a testing methodology.

Unit tests are methods that verify the functionality of target source code methods.

VS2012 windows forms provides an easy way for creating : Menus, Toolbars, ToolStrips etc.

References

[1] Kerievsky, J. (2005). Refactoring to patterns. Pearson Deutschland GmbH.

Housekeeping

- Assessment criteria for assignment 2 has been released.
- Assignment 3 has been released
  - Read assessment criteria carefully
- The following lecture will take place on Monday 5th January 2015. No more lecture for this month (Dec 2014).
- There is still one practical session on Monday 15th Dec 2014
  - Assignment 2 will be graded during the practical session.