

Examination #2

Name: _____ **Instructor's Solution** _____ SUID: _____

This is a closed book examination. Please place all your books on the floor beside you. You may keep one page of notes on your desktop in addition to this exam package. All examinations will be collected promptly at the end of the class period. Please be prepared to quickly hand in your examination at that time.

If you have any questions, please do not leave your seat. Raise your hand and I will come to your desk to discuss your question. I will answer all questions about the meaning of the wording of any question. I may choose not to answer other questions.

You will find it helpful to review all questions before beginning. All questions are given equal weight for grading, but not all questions have the same difficulty. Therefore, it is very much to your advantage to answer first those questions you believe to be easiest.

1. Write code fragments that show three distinct ways to define the processing for a delegate.

Answer:

```
class Program
{
    delegate void Traditional(string msg);
    event Traditional trad = new Traditional(handler);

    public static void handler(string msg)
    {
        Console.WriteLine("\n {0}", msg);
    }
    public class myArg : EventArgs
    {
        public string msg { get; set; }
    }
    public void handler2(object sender, EventArgs ev)
    {
        string msg = (ev as myArg).msg;
        Console.WriteLine("\n {0}",msg);
    }
    static void Main(string[] args)
    {
        Program p = new Program();
        p.trad.Invoke("Traditional delegate");

        Traditional anon = delegate(string msg)
        {
            Console.WriteLine("\n {0}", msg);
        };
        anon.Invoke("Anonymous delegate");

        Action<string> act = Program.handler;
        act.Invoke("Action delegate");

        Action<string> act2 = msg => Console.WriteLine("\n {0}", msg);
        act2.Invoke("Action using lambda");

        EventHandler eh = new EventHandler(p.handler2);
        myArg arg = new myArg();
        arg.msg = "EventHandler delegate";
        eh.Invoke(p, arg);

        Console.WriteLine("\n\n");
    }
}
```

2. What is XAML and what role does it play in a Windows Presentation Foundation application? Please be as specific as you can.

Answer:

XAML is XML markup used by Windows Presentation Foundation to describe and implement the layout of a WPF Window. The tag names of Xaml elements are .Net classes. Attributes of the elements are properties and events supported by the .Net class and given values for the element.

Typically each element represents a visible component of the Window's view, e.g., a Button, ListBox, Label, or some other control. The content of the element is the content of the control. So, for example, the content of a Button element may be the text that appears on the button surface.

Since XML is a tree structured representation, XAML provides a parse tree that describes the connectivity and inclusion of UI elements most of which are visible in the containing window.

The C# build process converts an application's XAML to C# code and compiles that with the C# code behind into MSIL which runs in the CLR virtual machine.

3. Write all the code for thread safe access to a text file. Consider opening, closing, reading, and writing contents of the file.

Answer:

```
public class ThreadSafeFile
{
    FileStream fs_;
    object locker_ = new object();
    public bool Open(string file)
    {
        for (int i = 0; i < 1000; ++i)
        {
            try
            {
                lock (locker_)
                {
                    fs_ = File.Open(file, FileMode.OpenOrCreate);
                }
                return true;
            }
            catch
            {
                Thread.Sleep(20);
            }
        }
        return false;
    }
    public byte[] Read(int size)
    {
        byte[] block;
        if (fs_.Length - fs_.Position >= size)
        {
            block = new byte[size];
            lock (locker_)
            {
                fs_.Read(block, 0, size);
            }
        }
        else
        {
            lock (locker_)
            {
                long remaining = fs_.Length - fs_.Position;
                block = new byte[remaining];
                fs_.Read(block, 0, (int)remaining);
            }
        }
        return block;
    }
    public void Write(byte[] block)
    {
        lock (locker_)
        {
            fs_.Write(block, 0, block.Count());
        }
    }
    public void Close()
    {
        lock (locker_)
        {
            fs_.Close();
        }
    }
}
```

4. Using the XDocument and XElement classes, write code that creates a Project #4 metadata file for the file Text1.txt where that file depends on Text2.txt and Text3.txt.

Answer:

```
class MetaData
{
    string file_ = "default.txt";
    string descrip_ = "default description";
    List<string> keys_ = new List<string>();
    List<string> depends_ = new List<string>();

    public void AddFileRef(string file) { file_ = file; }
    public void AddDescription(string description) { descrip_ = description; }
    public void AddKeyword(string Keyword) { keys_.Add(Keyword); }
    public void AddDependency(string file) { depends_.Add(file); }

    public string CreateMetadataContents()
    {
        XDocument doc = new XDocument();
        XElement root = new XElement("MetaData");
        XElement fileRef = new XElement("file", file_);
        root.Add(fileRef);
        XElement descrip = new XElement("description", descrip_);
        root.Add(descrip);
        XElement keywords = new XElement("keys");
        foreach (string key in keys_)
        {
            XElement keyword = new XElement("key", key);
            keywords.Add(keyword);
        }
        root.Add(keywords);
        XElement depends = new XElement("dependencies");
        foreach (string depend in depends_)
        {
            XElement dependency = new XElement("dependency", depend);
            depends.Add(dependency);
        }
        root.Add(depends);
        doc.Add(root);
        return doc.ToString();
    }
}

class TestMetaData
{
    static void Main(string[] args)
    {
        MetaData md = new MetaData();
        md.AddFileRef("Text1.txt");
        md.AddDescription("some description for file Text1.txt");
        md.AddKeyword("someKeyword");
        md.AddDependency("Text2.txt");
        md.AddDependency("Text3.txt");
        string xml = md.CreateMetadataContents();
        Console.WriteLine("{0}\n\n", xml);
        File.WriteAllText("../foo.xml", xml);
    }
}
```

5. How do you write the Critical Issues section of an Operational Concept Document, e.g., what do you say, what do you emphasize? Why do we write this section?

Answer:

Critical issues are concerns the OCD author has about the development team's ability to correctly implement all the features of the product with adequate security, safety, performance, simplicity, and ease of use. Note that critical issues are NOT simply design issues for which there is no significant difficulty handling effectively by the development team.

For each critical issue identified, the OCD author writes a brief statement describing the issue and its effects on uses and users, and proposes one or more solutions. The solutions need to be clear and complete enough that they provide significant direction for developers.

You are trying to get the developers to focus on important issues that may cause problems for the product if not handled properly.

6. When implementing a Windows Communication Foundation application how do you handle long-running CPU intensive tasks? Please discuss all the important issues associated with this implementation.

Answer:

WPF applications run in a single-threaded apartment. That means that the UI thread that created the window pulls event messages out of the window's message queue and acts on them to react to user and system inputs. If the application has to execute long-running CPU intensive tasks they should not be run on the Window's UI thread. If they are, the window will freeze while the tasks run because the UI thread is not available to process event messages.

For this reason the application should create child threads to handle CPU-intensive tasks so that the UI thread remains responsive to events. Those child threads must not call any functions on the window as WPF applications do not do any locking or other synchronization. The .Net framework will not allow that to happen by throwing exceptions when a non-UI thread tries to access the window's functions.

Each child thread returns its results by registering a delegate for the function it needs called and passing it to the UI thread via a Dispatcher.Invoke call. The UI thread extracts a system created message to access the delegate and calls it on behalf of its child thread.

7. Draw a package diagram for your planned implementation of Project #2. Briefly describe their responsibilities and interactions.

Answer:

- Executive – Direct Query activities
- CommandLineProc – Parse commandline to extract path, pattern, recursion, and query information
- FileMgr – collect references to files defined by path, pattern, and recursion information
- QueryProc – Delegate query actions to TextSearch and MetadataSearch
- TextSearch – open all files in fileset held by FileMgr and search for query strings using /O and /A commandline arguments to decide when a match is found
- MetadataSearch – search metadata files looking for key words, descriptions, and dependencies
- XmlProc – executes LINQ queries against individual metadata files
- Display – clearly and succinctly display evidence that each of the functional requirements of Project #2 have been met.
- DemoDriver – Spawn an instance of the Project code, with appropriate command line arguments to support the required display generation. This package is a batch file or script and will redirect the Display output to a file for inspection.

