

Examination #4

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. What is Action<T>? Write a code fragment to show how you would use it.

Answer:

Action is a .Net defined delegate that binds to functions that take no arguments and return no value. Action<T> is a delegate that binds to functions with a single argument of type T and return no value. There are more Action delegates defined to take multiple arguments, as Action<T1, T2> binds to functions that accept two arguments.

```
class Program
{
    void Handler(string msg)
    {
        Console.WriteLine("\n {0}, ThreadId = {1}", msg,
                          Thread.CurrentThread.ManagedThreadId);
    }
    static void Main(string[] args)
    {
        Program p = new Program();

        Action<string> act = p.Handler;
        string msg = "Hello CSE681 World";
        act.Invoke(msg);

        Task t = Task.Run(() => act.Invoke(msg));
        t.Wait();

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

2. Discuss how you would support file caching for the Code Repository Client of project #5¹. Are there any issues for your concept?

Answer:

To get a reasonable score on this you need to answer these questions: How and why will you add files to the cache and when will you remove them?

When we extract a component we need all its dependencies in order to build. But we may already have some of them. That's where the cache comes in.

A Code Repository client could implement caching by storing locally files it had received from the Repository at some earlier time. When it wants to download a component that depends on other files it requests the Repository to send a list of all the files on the component's dependency graph. It then downloads all the files on the list that it does not currently have in its cache, and adds them to the cache as well as uses them for its current purposes.

The repository builds the file list by traversing the virtual dependency graph formed by the component's metadata file and all its descendant metadata.

If the client modifies a file and checks-in a new version the client also places the modified file in its cache.

Note that this process could be part of a lazy loading mechanism to allow the client to navigate through local metadata, making a file request whenever it is unable to satisfy a reference found in one of its metadata files. This would save a lot of network traffic and server load and make client's navigation view more responsive.

One issue is how to recognize and resolve cache incoherencies. Note that it may not be terribly important that a client's cache doesn't have versions that were added to the Repository today. The client could, at midnight, request the server to send all metadata in the categories of interest to the client that have been added since the last update 24 hours ago. You probably want the client to have the ability to request metadata added to the repository since the client's last request. That is expensive enough that a client would only use when coherency is important.

Another issue is how to manage the cache contents. We probably do not want the cache to grow to become most of the Repository contents. The standard way of dealing with this is to use a "least recently used" algorithm to discard files when the cache size grows beyond some set-point.

¹ The Project #5 statement is attached at the back of this packet.

3. Discuss the syntax and semantics of passing a value type by reference to a member function of a class you are writing.

Answer:

We pass any type by reference using the `ref` keyword for both definition and invocation of the called function. When an invocation using pass by reference executes a reference to the caller's value type or reference type handle is created on the stack frame of the called function. For value types the only allowed action by the function is to read or write the value. Since that is done via a reference, both the caller and callee see the results of any write.

The syntax for function `f` declaration is:

```
T fun(ref VT vt) { ... }
```

and for invocation is:

```
VT vt = ...  
T t = fun(ref vt);
```

4. Write all the code for a lambda that accepts a string filename and concatenates that onto a string path which is a member of an enclosing class². It then causes the string to be written into a textbox in a WPF window that is part of the application. Please consider two cases: the lambda does and does not run on the Window's UI thread.

Answer:

```
public partial class MainWindow : Window
{
    Action<string> act1;

    string path = "../..";

    public MainWindow()
    {
        InitializeComponent();
    }

    public void Add(string file)
    {
        TextBox1.Text = file;
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        // Runs on UI thread

        string file = "MainWindow.xaml.cs";
        act1 = (name) =>
        {
            Add(path + name);
        };
        act1.Invoke(file);
    }

    public void ThreadProc()
    {
        // Runs on child thread

        string file = "MainWindow.xaml";
        act1 = (name) =>
        {
            this.Add(path + name);
        };
        this.Dispatcher.Invoke(act1, new object[] { file });
    }
    private void Button_Click_1(object sender, RoutedEventArgs e)
    {
        Thread t = new Thread(ThreadProc);
        t.Start();
    }
}
```

² That is, the code that constructs the lambda is in a member function of the enclosing class.

5. Write the XML for a Project #4 metadata file that refers to Text1.txt, where that file depends on Text2.txt and Text3.txt. Write LINQ queries that extract the file name and the file dependencies.

Answer:

```
<?xml version="1.0" encoding="utf-8" ?>
<metadata>
  <file>Text1.txt</file>
  <description>some text description</description>
  <keywords>
    <keyword>aKeyWord</keyword>
  </keywords>
  <dependencies>
    <dependency>Text2.xml</dependency>
    <dependency>Text3.xml</dependency>
  </dependencies>
</metadata>

class Program
{
  static void Main(string[] args)
  {
    XmlDocument doc = XmlDocument.Load("../Text1.xml");
    Console.WriteLine("\n{0}", doc.Declaration);
    Console.WriteLine("\n{0}\n\n", doc.ToString());

    var q = from x in
              doc.Elements("metadata").Elements("file")
            select x;

    Console.WriteLine("\n  filename: {0}", q.First().Value);

    q = from x in
          doc.Descendants("dependency")
        select x;

    foreach (var item in q)
      Console.WriteLine("\n  dependency: {0}", item.Value);

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

6. Write all the code to make thread safe access to a static string member variable "shared" of some class. Here, you are assuming that instances of the class may be used by more than one thread concurrently.

Answer:

```
class Program
{
    static string shared = "";
    static object locker_ = new object();

    public string SharedString
    {
        // this works only because string is immutable - see MT3-Q1
        get { lock(locker_) { return shared; } }
        set { lock (locker_) { shared = value; } }
    }

    static void Main(string[] args)
    {
        Program p = new Program();

        Action StringMod = () =>
        {
            for (int i = 0; i < 50; ++i)
            {
                if (p.SharedString.Length == 100)
                {
                    p.SharedString = "";
                    p.SharedString = p.SharedString +
                        (Thread.CurrentThread.ManagedThreadId).ToString();
                    // note: no guarantee Console will print just modified string
                    Console.WriteLine("\n {0}", p.SharedString);
                    Thread.Sleep(5 * Thread.CurrentThread.ManagedThreadId);
                }
            }
        };

        Thread t1 = new Thread(new ThreadStart(StringMod));
        t1.Start();
        Thread t2 = new Thread(new ThreadStart(StringMod));
        t2.Start();
        t1.Join();
        t2.Join();
    }
}
```

7. Draw an activity diagram for the file caching concept you discussed in Question #2.

Answer:

