

Examination #1

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. For Project #4 you are required to develop a message-passing communication system. Please specify some task a server will be requested to carry out, describe the message needed to make that request, and write the WCF service declaration needed to pass the message from client to server.

Answer:

Let's request a list of Projects. The SvcMsg class defines messages that can be used for all requests needed for Project #4. We've made the body, the part that carries message information, a string, expecting to supply an XML string to represent any information that needs to be transmitted. That allows the service contract use a single PostMessage(SvcMsg msg) method for any communications needed for the project.

```
[DataContract(Namespace="MT2Q2")]
public class SvcMsg
{
    public enum Command { Projects, ProjectList }; // Prj #4 needs bigger list of enums
    [DataMember]
    public Command cmd;
    [DataMember]
    public Uri src;
    [DataMember]
    public Uri dst;
    [DataMember]
    public string body;

    public void ShowMessage()
    {
        Console.WriteLine("\n Received Message:");
        Console.WriteLine("\n    src = {0}\n    dst = {1}", src.ToString(), dst.ToString());
        Console.WriteLine("\n    cmd = {0}", cmd.ToString());
        Console.WriteLine("\n    body:\n{0}", body);
    }
}

[ServiceContract(Namespace="MT2Q2")]
public interface IMessageService
{
    [OperationContract(IsOneWay=true)]
    void PostMessage(SvcMsg msg);
}
```

2. Write all the code for a thread-safe TypeTable designed to hold type information for the analyses in Projects #2 and #4.

Answer: (See MTF14-Code -> MT4Q1.cs for a solution using a list of namespaces for each type)

```

public class TypeElem
{
    public string Namespace { get; set; }
    public string Filename { get; set; }
}
public class TypeTable
{
    object locker_ = new object();
    public Dictionary<string, TypeElem> types { get; set; }

    public TypeTable()
    {
        types = new Dictionary<string, TypeElem>();
    }
    //-----< f will use lambda capture to acquire its arguments >-----

    T lockit<T>(Func<T> f)
    {
        lock (locker_) { return f.Invoke(); }
    }
    public bool add(string Type, string Namespace, string Filename)
    {
        TypeElem elem = new TypeElem();
        elem.Namespace = Namespace;
        elem.Filename = Filename;
        return lockit<bool>(() =>
        {
            if (types.Keys.Contains(Type) && types[Type].Namespace == Namespace)
                return false;
            types[Type] = elem;
            return true;
        });
    }
    public bool remove(string Type)
    {
        return lockit<bool>(() => { return types.Remove(Type); });
    }
    string namespace(string Type)
    {
        return lockit<string>(() =>
        {
            return (types.Keys.Contains(Type)) ? types[Type].Namespace : "";
        });
    }
    public string filename(string Type)
    {
        return lockit<string>(() =>
        {
            return (types.Keys.Contains(Type)) ? types[Type].Filename : "";
        });
    }
    public bool contains(string Type)
    {
        return lockit<bool>(() => types.Keys.Contains(Type));
    }
}

```

3. Given some type T with instance t, where `T t = new T()`, show how you would use reflection to find any of the properties of t that are contained in its assembly's metadata. Show how you would get values for two of those properties. You may choose any two of the many such properties.

Answer:

```
class MT1Q3
{
    public uint count { get; set; }

    public MT1Q3()
    {
        count = 0;
    }
    public void foobar()
    {
        // here just to give class a member function
    }
    static void Main(string[] args)
    {
        Console.WriteLine("\n MT1Q3 - Simple Reflection");
        Console.WriteLine("\n =====");

        MT1Q3 q3 = new MT1Q3();
        //---- Properties of Type t ----
        System.Type t = q3.GetType();
        Console.WriteLine("\n Writing t: {0}", t); // writes FullName
        Console.WriteLine("\n {0, 10} {1}", "Name:", t.Name);
        Console.WriteLine("\n {0, 10} {1}", "Namespace:", t.Namespace);
        Console.WriteLine("\n {0, 10} {1}", "Assembly:", t.Assembly);
        Console.WriteLine("\n {0, 10} {1}\n", "BaseType:", t.BaseType);
        //-----
        MemberInfo[] mi = t.GetMembers();
        foreach(var info in mi)
        {
            ++q3.count;
            Console.WriteLine("\n {0}", info.ToString());
        }
        Console.WriteLine("\n Class {0} has {1} public members", t.Name, q3.count);
        //---- Properties of MT1Q3 q3 ----
        q3.count = 0;
        PropertyInfo[] pi = t.GetProperties();
        foreach(var info in pi)
        {
            ++q3.count;
            Console.WriteLine("\n {0}", info.ToString());
        }
        //-----
        Console.WriteLine("\n Class {0} has {1} public properties", t.Name, q3.count);
        Console.WriteLine("\n\n");
    }
}
```

I meant to ask for properties of Type, not t, but that isn't what the question asked for. I'll accept either properties of Type or properties of t as correct.

4. Show how you define processing that is not a method of some class and can display on the console the message "Hello world" + str where str is a string that you will specify later. Now execute that processing on a threadpool thread.

Answer:

```
class MT1Q4
{
    static void Main(string[] args)
    {
        "MT1Q4 - Run lambda on Threadpool Thread".Title();

        // Here's the processing that isn't defined as a member function:

        Action<string> act = (string str) => Console.WriteLine("\n Hello World {0}", str);

        // Now we run it on a threadpool thread

        string s = "CSE681";
        Task t1 = Task.Run(() => Console.WriteLine("\n Hello World {0}", s));
        t1.Wait();

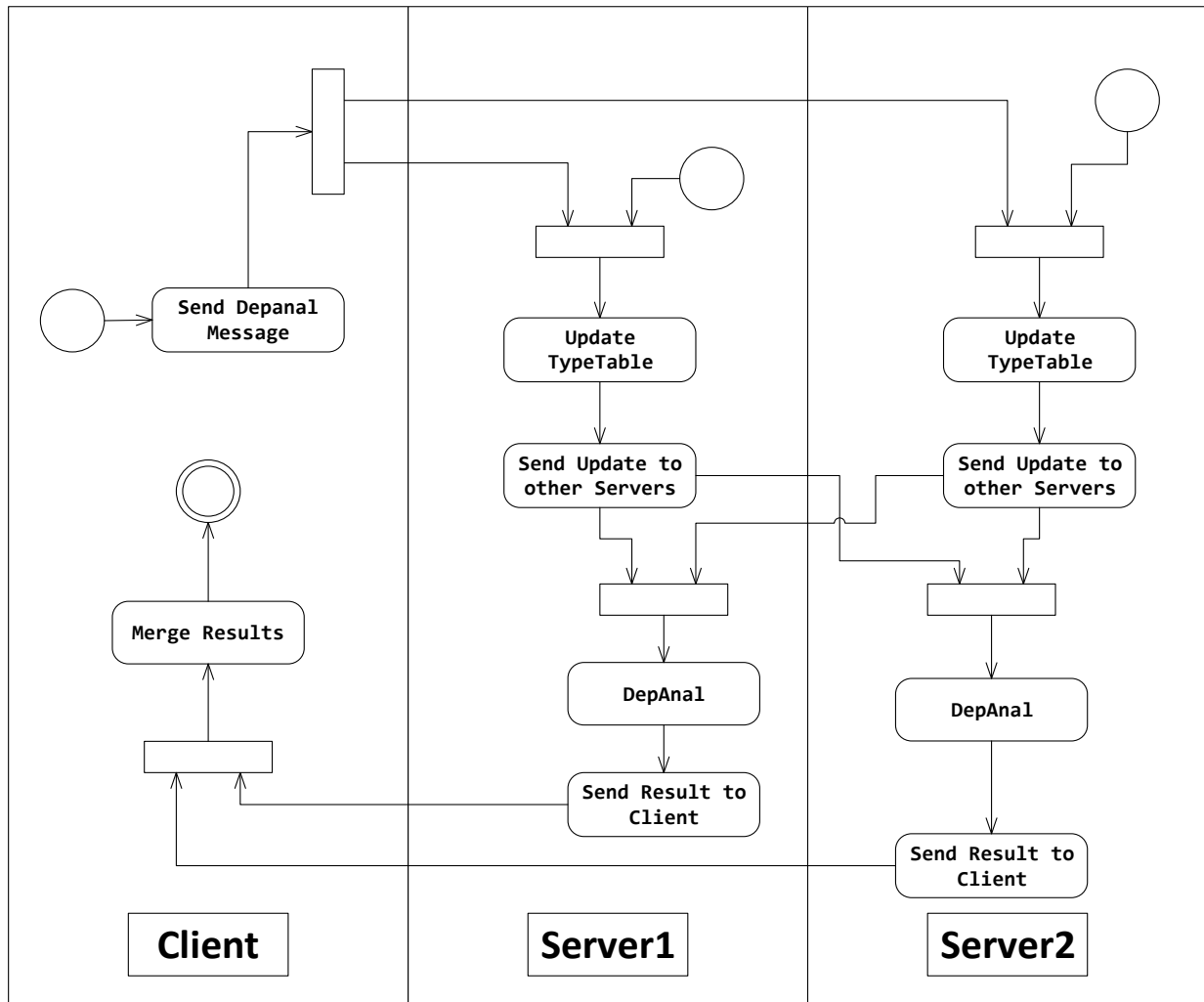
        // alternately

        Task t2 = Task.Factory.StartNew(() => Console.WriteLine("\n Hello World {0}", s));
        t2.Wait();
        Task t3 = Task.Run(() => act.Invoke(s));
        t3.Wait();

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

5. Draw an activity diagram that represents the processing for Project #4 required to find dependency relationships between packages in two projects where one project is located on Server #1 and the other on Server #2. You may assume that the dependency analysis on each server is a single activity so you don't have to show its details. This question is asking you to show the activities required to initiate the analyses and merge their results into one dependency description.

Answer:



6. Describe all of the significant critical issues for the development of Project #2.

Answer:

Critical Issues:

- Issue: Handling namespaces
It's easy to capture namespace information when defining a type – just look at the parser's scope stack. However, when a type is used it often is not decorated with its namespace.
- Solution:
Make the undecorated type the key in the TypeTable dictionary and make the corresponding value a list of namespaces in which types with that name are defined (along with the filename). Now when checking a type you see if it's in the type table and if so you check to see which using declaration matches a namespace in the TypeTable list, if any.
- Issue: Complexity of implementation
Parsing is a relatively complicated process for languages like C#. How can we implement this in a relatively short period of time?
- Solution:
Use the parser provided on the college server. Code reuse is one of the most effective ways of building complex projects, but the reused code needs to be carefully constructed to support reuse.
- Issue: Turning results from the code analysis into information
Parsing will generate Type information, function size and complexity, and relationship data for multiple files. How do we turn that data into information useful for the user?
- Solution:
Remove all output that is not specific to these required displays. Then organize the output into two parts:
First part - Types and their functions, naming the file where defined to the same line as the type, leaving no blank lines and indenting the function information, one line for each function.
Second part – Name the source type and its file one one line, then, in succeeding lines show the relationship and target type, indenting and leaving no blank lines.
- Issue: Quality Assurance will need to run analyses on many hundreds of files. How do we make that relatively simple to do.
- Solution:
Provide command line mode that will perform analysis for each file and write the final results in an XML file. This way the tool can be used in an unattended batch mode, perhaps scheduled to run automatically. Provide summary information of any defects detected, e.g., public data and functions that are too large or too complex. Make sure the results are named and dated.
- Issue: Quality Assurance analyses make have to deal with very large filesets, so performance may be an issue.
- Solution:
Since analysis of each file is independent of that for other files this processing is an ideal candidate for processing on multiple threads. That should improve performance by an amount approaching the number of available cores on the analysis machine. We could also use a machine cluster for analysis, merging the final results on a master machine.

Things that a compiler can check like junk files and incorrect syntax are not critical. Design issues are not critical. Large files are not critical. Poor error handling is not a critical issues – it's just poor design.

Each critical issue should be described briefly and that should be followed with a brief proposed solution.

7. Write a sample of the XML required to capture all of the type information developed in the first pass of Project #2. Now write C# code to create a file containing that XML, assuming you have already stored and have available all the data needed to do that. You may wish to loop over each record in the TypeTable you described in a previous question, writing out the XML elements to save that information to a file.

Answer:

```
class MT1Q7
{
    /*
    * <types>
    *   <type>
    *     <name>myType1</name>
    *     <namespace>MT1</namespace>
    *     <filename>file1.cs</filename>
    *   </type>
    *   <type>
    *     <name>myType2</name>
    *     <namespace>MT1</namespace>
    *     <filename>file2.cs</filename>
    *   </type>
    * </types>
    */
    static void Main(string[] args)
    {
        XDocument doc = new XDocument();
        doc.Declaration = new XDeclaration("1.0", "utf-8", "yes");
        XElement root = new XElement("types");
        doc.Add(root);
        XElement type = new XElement("type");
        XElement name = new XElement("name");
        name.Value = "type1";
        XElement namespace = new XElement("namespace");
        namespace.Value = "MT1";
        XElement filename = new XElement("filename");
        filename.Value = "file1.cs";
        root.Add(type);
        type.Add(name);
        type.Add(namespace);
        type.Add(filename);

        type = new XElement("type");
        root.Add(type);
        name = new XElement("name");
        name.Value = "type2";
        namespace = new XElement("namespace");
        namespace.Value = "MT1";
        filename = new XElement("filename");
        filename.Value = "file2.cs";
        type.Add(name);
        type.Add(namespace);
        type.Add(filename);

        doc.Save("MT1Q7.xml");

        System.IO.StreamReader rdr = new System.IO.StreamReader("MT1Q7.xml");
        string xml = rdr.ReadToEnd();
        Console.Write(xml);
        Console.WriteLine("\n\n");
    }
}
```

Here's an alternate somewhat more flexible solution:

```

public class TypeElem
{
    public string namespace_ { get; set; }
    public string filename_ { get; set; }
    public TypeElem(string namespc, string filenm)
    {
        namespace_ = namespc;
        filename_ = filenm;
    }
}

public class TypeTable
{
    public Dictionary<string, List<TypeElem>> types { get; set; }

    public TypeTable()
    {
        types = new Dictionary<string, List<TypeElem>>();
    }
    // other methods elided
}

class MT1Q7b
{
    static public void FakeTypeAnalysis(ref TypeTable tt)
    {
        tt.types["myType1"] = new List<TypeElem>();
        tt.types["myType1"].Add(new TypeElem("MT1", "file1.cs"));
        tt.types["myType2"] = new List<TypeElem>();
        tt.types["myType2"].Add(new TypeElem("MT1", "file2.cs"));
        tt.types["myType2"].Add(new TypeElem("MT2", "file3.cs"));
    }

    static public void CreateAndSaveXml(string dest, TypeTable tt)
    {
        //---- Answer starts here -----

        XDocument doc = new XDocument();
        XElement types = new XElement("types");
        doc.Add(types);
        foreach(string type in tt.types.Keys)
        {
            XElement t = new XElement("type");
            types.Add(t);
            XElement name = new XElement("name");
            name.Value = type;
            t.Add(name);
            XElement typeInfo = new XElement("typeInfo");
            t.Add(typeInfo);
            foreach(TypeElem te in tt.types[type])
            {
                XElement ns = new XElement("namespace");
                ns.Value = te.namespace_;
                typeInfo.Add(ns);
                XElement fn = new XElement("filename");
                fn.Value = te.filename_;
                typeInfo.Add(fn);
            }
        }
        doc.Save(dest);

        //---- Answer ends here -----
    }
}

```

```

/*
 * <types>
 *   <type>
 *     <name>myType1</name>
 *     <typeInfo>
 *       <namespace>MT1</namespace>
 *       <filename>file1.cs</filename>
 *     </typeInfo>
 *   </type>
 *   <type>
 *     <name>myType2</name>
 *     <typeInfo>
 *       <namespace>MT1</namespace>
 *       <filename>file2.cs</filename>
 *     </typeInfo>
 *     <typeInfo>
 *       <namespace>MT2</namespace>
 *       <filename>file3.cs</filename>
 *     </typeInfo>
 *   </type>
 * </types>
 */

```