

Examination #4

Name: _____ 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 all the code for a TypeTable that stores type information found in Project #2. You will need to save the type name, its namespace, and the file where it is defined. Please provide a merge method that accepts an instance of another TypeTable and adds to its own type storage any new types found in the merge function TypeTable argument.

Answer:

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

    public TypeTable()
    {
        types = new Dictionary<string, List<TypeElem>>();
    }
    public bool add(string Type, string Namespace, string Filename)
    {
        if (types.Keys.Contains(Type))
        {
            foreach (TypeElem te in types[Type])
            {
                if (te.Namespace == Namespace)
                    return false;
            }
            TypeElem insert = new TypeElem();
            insert.Namespace = Namespace;
            insert.Filename = Filename;
            types[Type].Add(insert);
            return true;
        }
        TypeElem elem = new TypeElem();
        elem.Namespace = Namespace;
        elem.Filename = Filename;
        types[Type] = new List<TypeElem>();
        types[Type].Add(elem);
        return true;
    }
    public bool remove(string Type)
    {
        return types.Remove(Type);
    }
    public bool contains(string Type)
    {
        return types.Keys.Contains(Type);
    }
    public uint merge(TypeTable srcTable)
    {
        uint count = 0;
        foreach (string Type in srcTable.types.Keys)
```

```
{
    if(!types.Keys.Contains(Type))
    {
        ++count;
        List<TypeElem> elems = new List<TypeElem>();
        foreach(TypeElem te in srcTable.types[Type])
        {
            TypeElem elem = new TypeElem();
            elem.Filename = te.Filename;
            elem.Namespace = te.Namespace;
            elems.Add(elem);
        }
        types[Type] = elems;
    }
    /*
    * Its left as an exercise for students to handle the case
    * where both tables contain the same type but with different
    * namespace(s). That will result in another entry.
    */
}
return count;
}
```

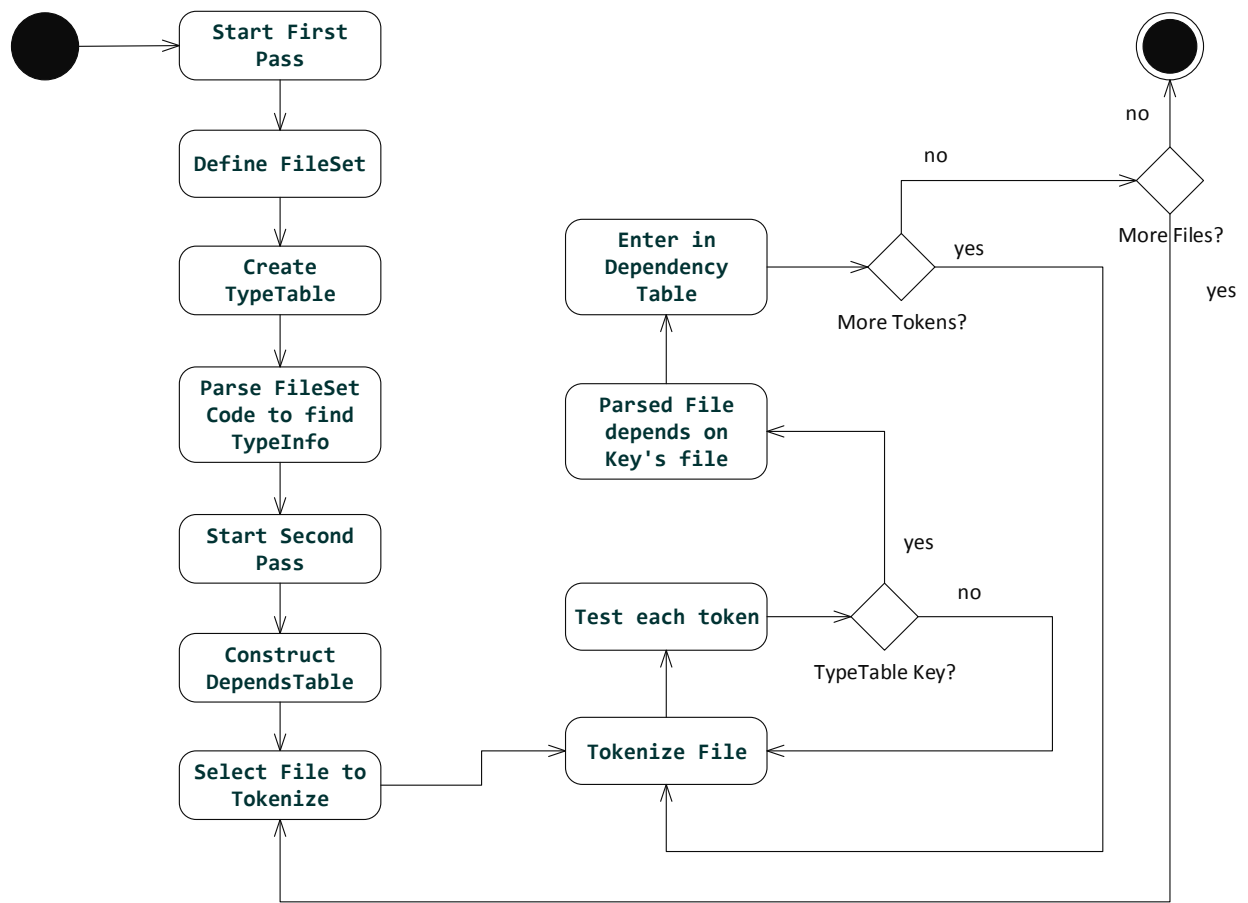
2. When finding package dependency relationships in Project #4 you start by constructing a type table. Assuming that has been done, describe the processing necessary to find all the dependencies between packages in the file set used to build the type table. Do that by constructing a task list and then drawing an activity diagram.

Answer:

Task List:

- In first pass parse all files to extract type information, e.g., type name, namespace, and file.
- Save each type in TypeTable
- In Second pass tokenize each file. Test each token to see if it is a key of the TypeTable, e.g., it is a type defined in the fileset.
- If it is, the file being tokenized depends on the file where that type was defined which is stored as part of the value of the TypeTable's key.
- Handling the possibility of the same type name being used to represent another type in another namespace is slightly more complicated. Data structure in solution 1, above starts that process.

Note that you could also do this by evaluating the type dependencies, as you did in Project #2, and examining to which package each type belonged.



3. When writing software for some significant project we often need to share data in the form of variables, data structures, queues, or databases. List the significant issues associated with such sharing between functions, threads, processes, and machines. Please describe the issue and ways of mitigating any associated risks. Please address each issue with brief discussions, e.g., a sentence or two. Your complete answer should fit on this page.

Answer: Note: I'm really asking you to discuss critical issues as you do in an OCD. I didn't expect you to provide all of the discussion below, but some of it. All may be useful for Project #5.

At all levels (functions, threads, processes, and machines) sharing of resources may have problems due to changes made by several parties. Assuming the changes succeed, one user's change may be inconsistent with the needs of another of the user's of the shared resource. This can only be handled by policy that defines a process to prevent conflicts were possible and resolve conflicts when prevention is not possible.

Sharing data between functions in a single threaded program has these issues:

- If the data is shared by reference, then one function may change the state of the data that was already set by another function possibly causing errors in program execution.
- Sharing references makes analysis of correctness difficult.
- **Solution:** try to make the class defining a reference type the only one mutating its instance.

Sharing data between threads has these issues:

- Each thread needs sequential access to the shared data to avoid read/write conflicts and potential data corruption.
- **Solution:** use locks and thread safe containers like the `BlockingQueue<T>`.
- A child thread cannot return data to a Window's controls. That has to be done by the UI thread.
- **Solution:** use `Dispatcher.Invoke` for WPF (that is much slower than a direct function call).

Sharing data between processes has these issues:

- Data must be marshalled between the processes either through shared memory, perhaps by using a pipe, or through the tcp stack perhaps using sockets or .Net Remoting or WCF.
- **Solution:** use prepackaged frameworks like WCF when feasible.
- Sharing data between processes is much slower than sharing data between functions in the same process.
- **Solution:** think about the data design before committing to code.

Sharing Data between machines has these issues:

- This must be done using network communication, e.g., through the tcp stack of both machines. This is much slower than interprocess communication on the same machine.
- **Solution:** think even more about data design before committing to code.
- Sharing between machines is less reliable than sharing between functions and processes on the same machine. For example the target machine may not be available when the attempt to share is executed.
- **Solution:** try to make processing asynchronous and tolerant of incremental updates

Sharing files in the same process has these issues:

- If one part of the code opens many files without closing them quickly all the file handles available to the process may be used, preventing other code from accessing any other files.
- **Solution:** Close a file as soon as you have completed using it. Don't wait to leave the local scope.

Sharing files between processes on same machine has these issues:

- If two processes attempt to open the same file only one will succeed, causing the other to block or to throw an exception.
- **Solution:** Use facilities like the `ConcurrentFileAccess` demo which catches exceptions, sleeps for a brief time, and retries.

Sharing files between machines has these issues:

- To share files between machines we need to implement some form of file transfer. When many files are traveling back and forth it is likely that a file is sent that may already be at the destination.
- **Solution:** Use WCF to implement a file sharing service (see `WCF_Fawcett_Examples`). Sending files as message content may exceed message size limits so file transfer with chunking is advisable. We can implement file caching with pull communication rather than push to avoid receiving the same file multiple times with the corresponding network overhead.

4. Write a function wrapper for the dependency analysis for Project #4 providing correct input arguments and return type. You do not need to provide any contents for the function, just show its signature. You need to think carefully about the return type. Now write code to run that function on a threadpool thread and eventually retrieve the result.

Answer:

```
using Dependencies = Dictionary<string, List<string>>;

public static class Extensions
{
    public static void ShowDeps(this Dependencies deps)
    {
        "Package Dependencies".Title('-', false);
        foreach (string key in deps.Keys)
        {
            Console.WriteLine("\n {0}", key);
            foreach (string pkg in deps[key])
            {
                Console.WriteLine("\n    {0}", pkg);
            }
        }
        Console.WriteLine("\n");
    }
}

class MT4Q4
{
    Dependencies deps = new Dependencies();

    Dependencies analyze()
    {
        // simulate dependency analysis - needs access to appropriate member data
        deps["Pkg1.cs"] = new List<string>();
        deps["Pkg1.cs"].Add("Pkg2.cs");
        deps["Pkg1.cs"].Add("Pkg3.cs");
        deps["Pkg2.cs"] = new List<string>();
        deps["Pkg2.cs"].Add("Pkg3.cs");
        deps["Pkg2.cs"].Add("Pkg4.cs");
        deps["Pkg3.cs"] = new List<string>();
        deps["Pkg3.cs"].Add("Pkg5.cs");
        deps["Pkg4.cs"] = new List<string>();
        deps["Pkg5.cs"] = new List<string>();
        return deps;
    }

    static void Main(string[] args)
    {
        "MT4Q4 - Dependency Analysis on Threadpool Thread".Title();
        MT4Q4 q1 = new MT4Q4();
        Console.WriteLine("\n show parent-child relationships");
        Task<Dependencies> t1 = Task<Dependencies>.Run(() => q1.analyze());
        t1.Result.ShowDeps();
        Console.WriteLine("\n");
    }
}
```

5. Describe the characteristics of a C# class that allow you to make a clone object that is entirely independent of its source object. Describe the process of making a clone for one of these classes.

Answer:

Any C# reference type is clonable if all its properties and fields are value types. In that case we create a clone by making a new instance of the type and copying its properties and fields to the new instance. We could do that, for example, in a clone() member function that returns a reference to the new type created on the heap with the new operator.

Now, any other C# type will be clonable if all its properties and fields are either value types or references to clonable types. We create a new instance of the type in its clone() function and copy all its value types to the new instance. Now, for each reference to a clonable type we attach the reference to the type in the new instance to the output of the clone() operation.

The basic requirement is that, when you consider the tree of a class and all its child types and all their types recursively, all of the leaf types have to be value types and accessible. Otherwise you cannot make a clone of that type that is independent of the original.

The main issue is that for many reference types not all the leaf types are accessible for either copying, if value types, or cloning if clonable.

6. Project #4 servers will need to exchange type table information. Write a WCF service contract for sending the appropriate messages.

Answer:

```
//---- answer starts here -----
[DataContract(Namespace="MT2Q2")]
public class SvcMsg
{
    public enum Command { Projects, ProjectList }; // Needs bigger set of enums
    [DataMember]
    public Command cmd; // [DataMember] tells WCF to serialize for transmission
    [DataMember]
    public Uri src;
    [DataMember]
    public Uri dst;
    [DataMember]
    public string body; // Used to send XML for structured data
//---- start not needed for correct answer -----
    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);
    }
}
//---- end not needed for correct answer -----
[ServiceContract(Namespace="MT2Q2")]
public interface IMessageService
{
    [OperationContract(IsOneWay=true)]
    void PostMessage(SvcMsg msg);

    SvcMsg GetMessage();
}
//---- answer ends here -----
[ServiceBehavior(Namespace="MT2Q2")]
public class MessageService : IMessageService
{
    static SWTools.BlockingQueue<SvcMsg> Q = new SWTools.BlockingQueue<SvcMsg>();
    public void PostMessage(SvcMsg msg)
    {
        Q.Enqueue(msg);
    }

    public SvcMsg GetMessage()
    {
        return Q.Dequeue();
    }
}
```

7. When a Project #4 server shuts down it will probably save its type table in an XML file to be read when it starts the next time. Write the XML needed to describe type table information, e.g., type, namespace, and file where defined. You may use a couple of sample types for this answer. Now show how the server would query the XML for its type information when restarting. This is just asking you to write one or more LINQ queries.

Answer:

```
//----< partial test of type-safe TypeTable >-----
```

```
static void Main(string[] args)
{
    TypeTable tt = new TypeTable();
    tt.add("Type1", "myNamespace", "someFile.cs");
    tt.add("Type2", "myNamespace", "someOtherFile.cs");
    tt.add("Type1", "yourNamespace", "anotherFile.cs");
    tt.show();
    Console.WriteLine("\n");
```

```
    "XML representation of TypeTable".Title('-', false);
    string Xml = tt.save();
    Console.WriteLine("\n{0}\n", Xml);
```

```
    "Making LINQ Queries".Title('-', false);
    XDocument doc = XDocument.Parse(Xml);
```

```
    // query for types
    var q1 = from x in
                doc.Elements("typeInfo")
                .Elements("type")
                .Elements("name")
                select x;
```

```
    Console.WriteLine("\n query for types:\n");
    foreach (var elem in q1)
        Console.WriteLine("{0} ", elem.Value);
```

```
    // query for filenames
    var q3 = from x in
                doc.Elements("typeInfo")
                .Elements("type")
                .Elements("filename")
                select x;
```

```
    Console.WriteLine("\n query for filenames:\n");
    foreach (var elem in q3)
        Console.WriteLine("{0} ", elem.Value);
```

```
    // query for distinct namespaces
    var q2 = (from x in
                doc.Descendants()
                where (x.Name == "namespace")
                select x.Value).Distinct();
```

```
    Console.WriteLine("\n query for namespaces:\n");
    foreach (var elem in q2)
        Console.WriteLine("{0} ", elem);
    Console.WriteLine("\n\n");
}
```

```
C:\WINDOWS\system32\cmd.exe

TypeTable Contents
-----
Type1
  Namespace = myNamespace      File = someFile.cs
  Namespace = yourNamespace    File = anotherFile.cs
Type2
  Namespace = myNamespace      File = someOtherFile.cs

XML representation of TypeTable
-----
<typeInfo>
  <type>
    <name>Type1</name>
    <namespace>myNamespace</namespace>
    <filename>someFile.cs</filename>
    <namespace>yourNamespace</namespace>
    <filename>anotherFile.cs</filename>
  </type>
  <type>
    <name>Type2</name>
    <namespace>myNamespace</namespace>
    <filename>someOtherFile.cs</filename>
  </type>
</typeInfo>

Making LINQ Queries
-----
query for types:
  Type1 Type2
query for filenames:
  someFile.cs anotherFile.cs someOtherFile.cs
query for namespaces:
  myNamespace yourNamespace

Press any key to continue . . .
```