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 all the code to define a WCF data contract, useful for Project #4:

   Answer:

```
[DataContract]
public class CommMessage
{
  public enum MessageType
  {
    [EnumMember]
    connect,           // initial message sent on successfully connecting
    [EnumMember]
    request,           // request for action from receiver
    [EnumMember]
    reply,             // response to a request
    [EnumMember]
    closeSender,       // close down client
    [EnumMember]
    closeReceiver      // close down server for graceful termination
  }

  /*----< constructor requires message type >--------------------*/

  public CommMessage(MessageType mt)
  {
    type = mt;
  }
  /*----< data members - all serializable public properties >----*/

  [DataMember]
  public MessageType type { get; set; } = MessageType.connect;
  [DataMember]
  public string to { get; set; }
  [DataMember]
  public string from { get; set; }
  [DataMember]
  public string author { get; set; }
  [DataMember]
  public Command command { get; set; }
  [DataMember]
  public List<Argument> arguments { get; set; } = new List<Argument>();
  [DataMember]
  public int threadId { get; set; } = Thread.CurrentThread.ManagedThreadId;
  [DataMember]
  public ErrorMessage errorMsg { get; set; } = "no error";
```

2. Define the syntax and semantics for passing a reference type by reference to a C# method. Please consider all relevant cases.

Answer:

```
public class X
{
  public string name { get; set; } = "";
}
public class Y
{
  public void someFun(ref X x)  // call passing reference type X by reference
  {
    x.name = "X";   // x is a reference to the caller's reference x, in main, ,below.
    x = new X();
    x.name = "newX";
  }
}
static void Main(string[] args)
{
  Console.Write("\n  MT2Q2 - passing reference type by reference");
  Console.Write("\n ============================================");

  X x = new X();
  Y y = new Y();
  y.someFun(ref x);
  // caller sees change of state and change of object
  Console.Write("\n  after call: x.name = {0}\n\n", x.name);
}
```

3. Write a list of critical issues for an OCD describing the concept for Project #4. Please state the issues, clearly and briefly, and propose a solution for each.

   Answer:

   This development will create a Build Server, capable of building C# and C++ libraries, using a process pool to conduct multiple builds in parallel. The implementation is accomplished in three stages.

   The first, Project #2, implements a local Build Server that communicates with a mock Repository, mock Client, and mock TestHarness, all residing in the same process. Its purpose is to allow the developer to decide how to implement the core Builder functionality, without the distractions of a communication channel and process pool.

   The second, Project #3, develops prototypes for a message-passing communication channel, a process pool, that uses the channel to communicate between child and parent Builders, and a WPF client that supports creation of build request messages.

   Finally, the third stage, Project #4, completes the build server, which communicates with mock Repository, mock Client, and mock TestHarness, to thoroughly demonstrate Build Server Operation.

   Critical issues for this development are:

   - **Issue #1**
     How to define a single message structure that works for all messages used in the Federation.
   - **Solution:**
     A message that contains To and From addresses, Command string or enumeration, List of strings to hold file names, and a string body to hold logs will suffice for all needed operations.
   - **Issue #2**
     Building and testing both C# and C++ code.
   - **Solution:**
     For building, set environment variables as demonstrated in Help Session demo, and use tool chain commands for each type of source. For testing, trap exceptions on loading native code libraries in the C# TestHarness and direct to C++ TestHarness, as demonstrated in class.
   - **Issue #3**
     Managing EndPoint information for Repository, BuildServer, and TestHarness.
   - **Solution:**
     Store Enpoint information in XML file resident with all clients and servers and load at startup.

4. The System.Diagnostics.Process class provides a delegate type:

```
public event delegate void EventHandler(object sender, EventArgs e);
```

and a delegate instance:

```
public EventHandler Exited
```

that is invoked when the process exits. Write code to show how the Mother Builder you designed in Project #3 could save a reference to a created child process, and, using the Exited notification, remove the process reference from the Mother Builder's process collection when the child process terminates.

Answer:

```
class MockMotherBuilder
{
  Dictionary<int, Process> processMap = new Dictionary<int, Process>();
  public void createChildBuilder(int port, string fileSpec)
  {
    Process childBuilder = new Process();
    childBuilder.StartInfo.FileName = fileSpec;
    childBuilder.StartInfo.Arguments = port.ToString();
    childBuilder.EnableRaisingEvents = true;
    childBuilder.Exited += new EventHandler(childBuilder_Exited);
    childBuilder.Start();
    processMap[childBuilder.Id] = childBuilder;
  }
  private void childBuilder_Exited(object sender, EventArgs e)
  {
    Process theDeadProcess = (Process)sender;
    Console.Write("\n  ChildBuilder {0} exited", theDeadProcess.Id);
    processMap.Remove(theDeadProcess.Id);
    Console.Write("\n    {0} child processes remain", processMap.Count);
  }
}
class MT2Q4
{
  static void Main(string[] args)
  {
    Console.Write("\n  MT2Q4 - Handling Child Builder Exit Events");
    Console.Write("\n ==========================================");
    MockMotherBuilder mb = new MockMotherBuilder();
    string path = "../../../MT2Q4-child/bin/Debug/MT2Q4-child.exe";
    string absPath = System.IO.Path.GetFullPath(path);
    int count = 1, basePort = 8080;
    if (args.Length > 0)
      count = Int32.Parse(args[0]);
    for(int i=0; i<count; ++i)
    {
      try
      {
        mb.createChildBuilder(basePort + i + 1, absPath);
      }
      catch
      {
        Console.Write("\n  could not start child builder");
      }
    }
    Console.ReadKey();
    Console.Write("\n\n");
  }
}
```

5. Write all the code needed to run the method:

```
void someMessageProcessing(Message msg) { … }
```

asynchronously, where msg is an instance of a Message class. You don't need to use the details of the Message class or the body of the method to answer this question.

Answer:

```
using Message = String;
class MT2Q5
{
  public void someMessageProcessing(Message msg)
  {
    Console.Write("\n  starting message processing");
    Thread.Sleep(200);
    Console.Write("\n    {0}", msg.ToString());
    Console.Write("\n  ending message processing");
  }
  public Thread asynchronousInvoker1(Message msg)
  {
    Thread t = new Thread(() => {
      someMessageProcessing(msg); });  // run only this method
    t.Start();
    return t;
  }
  public Task asynchronousInvoker2(Message msg)
  {
    return Task.Run(() => { someMessageProcessing(msg); });  // run only this method
  }
  public Task asynchronousInvoker3(Action act)
  {
    return Task.Run(act);  // run any method
  }
  static void Main(string[] args)
  {
    Console.Write("\n  MT2Q5 - run method asynchronously");
    Console.Write("\n ===================================");

    MT2Q5 p = new MT2Q5();
    Thread t = p.asynchronousInvoker1("hello CSE681 world");
    Console.Write("\n  returned from asynchronousInvoker1");
    t.Join();
    Console.Write("\n");

    Task task = p.asynchronousInvoker2("hello again CSE681 world");
    Console.Write("\n  returned from asynchronousInvoker2");
    task.Wait();
    Console.Write("\n");

    Message msg = "hello CSE681 world one last time";
    Action act = () => p.someMessageProcessing(msg);
    task = p.asynchronousInvoker3(act);
    Console.Write("\n  returned from asynchronousInvoker3");
    task.Wait();
    Console.Write("\n\n");
  }
```

6. Is a .Net string a value or reference type? Explain how an instance of the string class behaves and why it has that behavior. The behaviors you need to discuss are creation, assignment, and modification.

   Answer:

   .Net strings are reference types that mimic value type behavior. When one string is assigned to another, both string handles point to the same instance in the managed heap. However, string contents are immutable. If one of the handles attempts to change the object, a new string object is created and the old character contents, which are value types, are copied with changes into the new string. This "copy on write" behavior makes strings appear to be value types, e.g., each handle appears to contain contents independent of any other handle.

   Strings are constructed with literal string with syntax:

```
string s = "Hello World";
```

   You cannot construct as below:

```
string s2 = new string("Hello World");  // fails to compile
```

   You can construct from an array of chars like this:

```
string s3 = new string(new char[] { 'a', 'b', 'c' });
```

7. Draw a package diagram for your design of Project #4. To save time you may represent all of Comm with a single package.

Answer: