Examination #3

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. Given a class with two public string properties and no other code, write all the code to define processing that exchanges the values of the two strings inside one function. Write a second function that accepts and executes that processing. After the second function executes the exchange must be persistent.

   Answer:

```
class PairOfStrings
{
  public string s1 { get; set; } = "s1";
  public string s2 { get; set; } = "s2";
}
class MT3Q1
{
  static Action<PairOfStrings> FunctionThatDefinesProcessing()
  {
    Console.Write("\n  define processing");

    Action<PairOfStrings> act = (pos) =>
    {
      string temp = pos.s1;
      pos.s1 = pos.s2;
      pos.s2 = temp;
    };
    return act;
  }
  static void FunctionThatUsesProcessing(Action<PairOfStrings> act, PairOfStrings pos)
  {
    Console.Write("\n  use processing");

    act.Invoke(pos);
  }
  static void Main(string[] args)
  {
    "MT2Q2 - Functions that define and invoke processing".title('=');

    PairOfStrings pos = new PairOfStrings();
    pos.s1 = "first string";
    pos.s2 = "second string";

    Console.Write("\n  original value of s1 =   \"{0}\"", pos.s1);
    Console.Write("\n  original value of s2 =   \"{0}\"", pos.s2);

    Action<PairOfStrings> act = MT3Q1.FunctionThatDefinesProcessing();
    MT3Q1.FunctionThatUsesProcessing(act, pos);
    Console.Write("\n  Persistent value of s1 = \"{0}\"", pos.s1);
    Console.Write("\n  Persistent value of s2 = \"{0}\"", pos.s2);
    Console.Write("\n\n");
  }
}
```

2. What does the following code do:

```
class ExampleClass { /* code not disclosed to you */}

Type t = typeof(ExampleClass);
```

Answer:

It creates an instance of the class Type and fills it with type information about the type ExampleClass using reflection. The Type class is a container for reflection information. The operator typeof() parses the class's metadata and fills t with type information about that class.

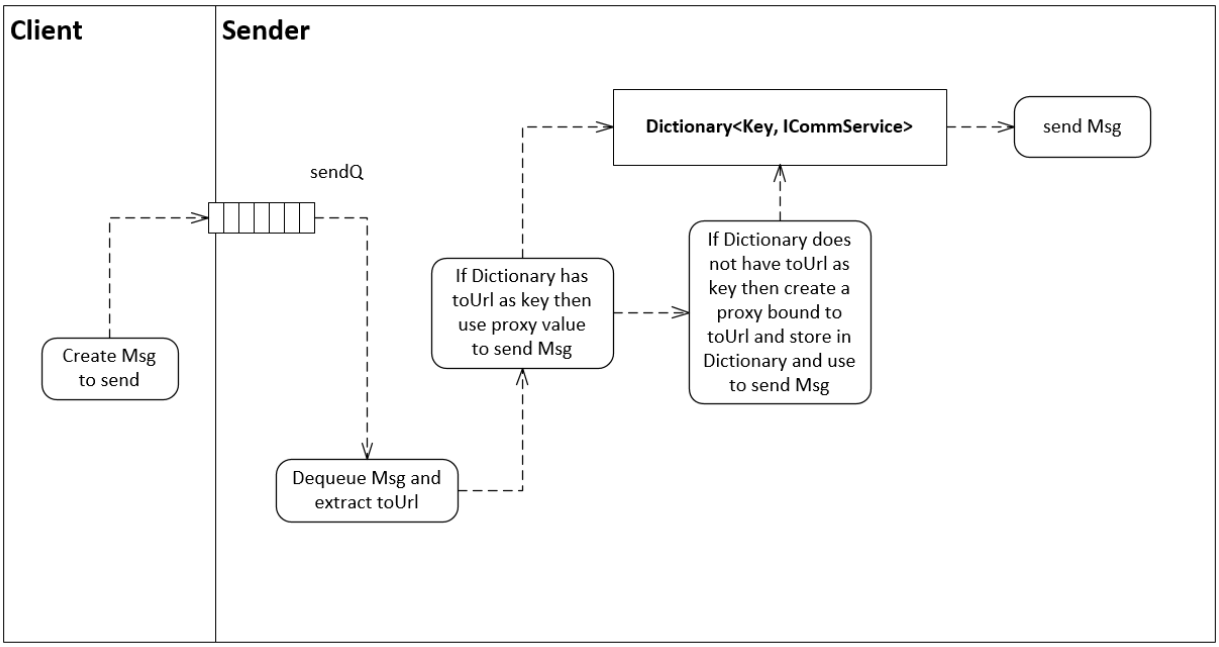
3. How would you implement a message Sender for Project #4? You may describe the functioning of the instructor's latest example code or define your own. How do instances of this Sender enable sending messages to a sequence of different Receivers? Please provide a diagram that supports your description.

   Answer:

   Sender creates a proxy for a remote object hosted by the Receiver to which it connects. The attempt to connect doesn't happen until the first call is made on the proxy. The proxy is created using the .Net ChannelFactory<ICommService> with binding and address information provided by the Sender.

   The Sender provides a send queue used by the application to hand a message to the send thread. The send thread examines the toUrl property of the message to get the address of the intended Receiver. The send thread maintains a Dictionary<Key, ICommService> proxyStore to hold onto proxies that have already been created. When the send thread dequeues a message from the send queue it does a proxyStore lookup for the msg.toUrl. If it finds the key it uses the associated proxy to send the message. If it doesn't find the toUrl in the proxyStore it creates a new proxy and stores it in the dictionary, then sends the message using the newly created proxy.

   The combination of message inspection and proxy lookup supports sending, in sequence, to any number of remote Receivers.



4. Write all the code for a C# class with an asynchronous function[1] that notifies instances of other classes that some event occurred during its processing. How do those instances request notification? You may assume that the function's processing includes a long-running loop and the events occur every N iterations. You may assume that the function's processing consists of writing a Console message for each iteration of the loop.

   Answer:

```
public class Notifier {
  Thread t = null;
  object sync_ = new object();
  public Action<string> notify { get; set; } = null;

  public void join() { t.Join(); }

  virtual public void doNotification() { Console.Write("\n  no one is listening"); }

  public void asyncRun() {
    ThreadStart ts = () =>
    {
      int count = 0;
      while (true) {
        Console.Write("\n  running");
        if(++count%5 == 0) {
          if (notify != null)
            notify.Invoke("this is a notification from delegate");  // notify with delegate
          doNotification();  // notify via virtual function call
        }
        if (count > 20)
          break;
        Thread.Sleep(100);
      }
    };
    t = new Thread(ts);
    t.Start();
  }
}
class MT3Q4 {
  void notifyHandler(string msg) { Console.Write("\n  {0}", msg); }

  class GetNotified : Notifier {
    public override void doNotification() {
      Console.Write("\n  Got notification from overload of virtual method");
    }
  }
  static void Main(string[] args) {
    "MT3Q4 - Asynchronous Notifier".title();
    MT3Q4 mt4q7 = new MT3Q4();
    GetNotified notifier = new GetNotified();
    notifier.notify += (msg) => { mt4q7.notifyHandler(msg); }; // register
    notifier.asyncRun();
    notifier.join();
    Console.Write("\n\n");
  }
}
```

[1] An asynchronous function returns almost immediately after being called without waiting for the function's processing to complete.

5. The communication system used in Project #4 sends messages with XML strings as bodies. Elements of the XML define what the receiver should do with the message and provide parameters for carrying out that request. Write the XML markup that a client could use to request the execution of a query for all the database elements that have a specified string in their metadata descriptions, sent to a server hosting the database you implemented in Project #2. Please provide all the information in the XML markup necessary to carry out that operation. You must assume that the server has no prior information about the query.

   Answer:

   Using templates for messages ensures that all parties to a communication are using the same language. Templates for messages are found in code for MT1Q4:

```
Body of request message (tag name identifies type of request):
<location></location> identifies places to look, e.g., localizes search for
efficiency.

  <queryString>
    <description>query for string in location</description>
    <testString>a specified string</testString>
    <!-- location: name, descr, children, payload -->
    <location>descr</location>
    <location>payload</location>
  </queryString>

Body of reply message:  (not required for solution)

  <queryReply>
    <description>Query reply returns matching keys</description>
    <keys>
    </keys>
  </queryReply>
```

   If there are no keys in reply there was no match.

6. Suppose that the sender of the message described in the previous question builds the XML markup from a markup template. That is, for each of the required messages a markup template exists and the client simply provides the appropriate information to be inserted into the template? Draw an activity diagram for that processing.

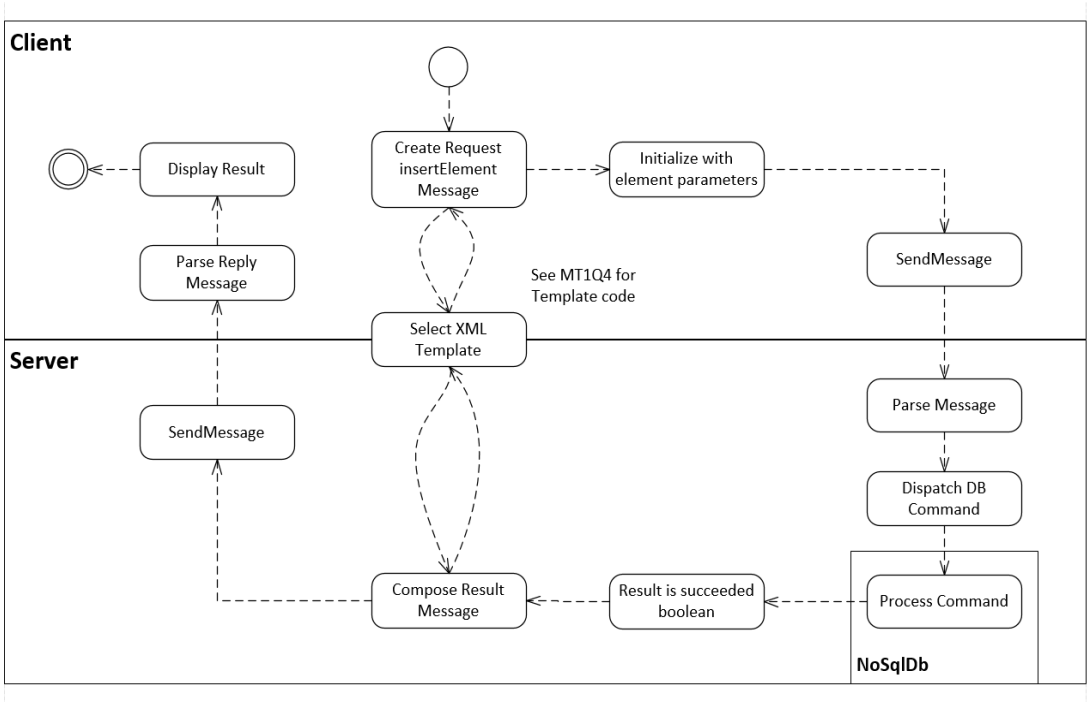
   Answer: See code for MT1Q4 for code to create message bodies from templates.

```
// doc contains templates for messages - we're using the insertElement template

XElement insert = doc.Element("messages").Element("insertElement");
insert.Element("key").Value = key;
insert.Element("name").Value = proxy.name;
insert.Element("descr").Value = proxy.descr;
insert.Element("timeStamp").Value = proxy.timeStamp;

foreach(string k in proxy.children)
{
  XElement item = new XElement("key");
  item.Value = k;
  insert.Element("children").Add(item);
}

foreach(string item in proxy.payload)
{
  XElement itm = new XElement("item");
  itm.Value = item;
  insert.Element("payload").Add(itm);
}
return insert.ToString();
}
```



Could use a few words of description instead of code, above.

7. What is meant by the term "instancing" in the context of a WCF communication channel? Please cite the instancing modes provided by WCF.

   Answer:

   Instancing refers to the mode of managing hosted object lifetimes used by the WCF framework. The WCF service has no way to know when a sender has finished with a hosted service object. To avoid the accumulation of many objects that will no longer be used, the service must provide one or more mechanisms to discard hosted objects. WCF uses an "Instancing Mode" to set a policy for that object lifetime management.

   The "PerCall" mode sets service object lifetime to be the duration of a single call, after which the object is enqueued for garbage collection. Each caller gets its own hosted object running on a dedicated WCF thread.

   The "PerSession" mode defines a leased lifetime for each created object. The lease begins with the first call when a hosted object is created. If the caller does not call back within the lease time the object is destroyed. If the caller does call back within the least time, then the lease is extended, starting from the last call. Each caller gets its own object running on a dedicated WCF thread.

   The "Single" mode manages a leased lifetime for a single remote object. A WCF thread is created for each caller that access the single object. That is, every caller is referring to the same object and consequently the object must lock all of its member data and any other shared data.