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. Write all the code to define, create, and use, a GUI child thread that retrieves string messages from a receive queue and causes them to be inserted into a ListBox.

   Answer:

```
public MainWindow()
{
  InitializeComponent();
  Thread t = new Thread(threadProc);
  t.IsBackground = true;
  t.Start();
}

void AddMsg(string msg)
{
  listBox.Items.Insert(0, msg);
}

void threadProc()
{
  while(true)
  {
    string msg = rcvQ.deQ();
    Action act = () => { AddMsg(msg); };
    string[] args = new string[] { };
    Dispatcher.Invoke(act, args);
  }
}
```

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

Answer:

```
public class X { … }
public class Y
{
  public void someFun(X x)   // method accepting reference type X by value
  {
    …
  }
}

X x = new X();  Y y = new Y();
y.someFun(x);
```

The code snippets, above, show the syntax for passing a reference type by value.

The semantics of this operation are as follows:

- The reference, x, is copied onto the stack frame of someFun. Note that x is a handle linked to the instance of X, created by the caller, on the managed heap.
- If the function changes the value of the instance through its handle x, then the caller will see the change because it's x refers to the same instance.
- If the function assigns a new instance handle to its x:
  `x = new X();`
  the caller will not see the change because the function has a copy of caller's handle x.

3. Write an Executive Summary for the Operational Concept Document (OCD) of Project #1, as you now think is should be written (not necessarily how you wrote it in your submission).

   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.

   The final product consists of a relatively small number of packages. For most packages there already exists prototype code that show how the parts can be built. For this reason, there is very little risk associated with the Build Server development.

   Critical issues include: building source code using more than one language, scaling the build process for high volume of build requests, and using a single message structure for all message conversations between clients and servers. All of these issues have viable solutions.

   The Build Server will function as one of the principle components of a Software Development Environment Federation, the others being Repository, TestHarness, and Federation Client. Building these other Federation parts is beyond the scope of this development.

4. Describe all of the relevant contracts for the WCF channel you used in Project #3. Please be specific.

   Answer:

   There is a [ServiceContract], defining the interface IMessagePassingComm, supporting a service [OperationContract]:

   ```
   void postMessage(CommMessage msg)
   ```

   and a non service operation:

   ```
   CommMessage getMessage();
   ```

   The [ServiceContract] also supports service [OperationContract(…)]s:

   ```
   bool openFileForWrite(string name);
   bool writeFileBlock(byte[] block);
   void closeFile();
   ```

   There is a [DataContract] defining the class CommMessage with public [DataMember] properties:

   ```
   MessageType type
   string to
   string from
   string author
   string command
   List<string> arguments
   int threadId
   string errorMessage
   ```

5. What is a delegate and why is it an important part of .Net Framework? Please list all of the uses you can think of for delegates.

   Answer:

   A delegate is a reference type that declares a method signature defining the methods to which it can be bound. It contains a reference to an instance of its own type, which, when initialized, refers to another delegate of the same type which refers to a method, with the prescribed signature, of any reference type. This allows the delegate to bind to any finite number of handlers, one for each delegate in a linked list of delegates, rooted at the publisher's delegate reference.

   Delegates support the definition of processing to be invoked, along with any data needed to execute the processing. The delegate may be passed or returned from a method, stored, and executed at some later time. This allows an application to define processing that handles an event defined in a library that exposes a delegate and the processing is invoked by library code when a specific event, associated with the delegate, occurs within the library code. This behavior allows the library to be ignorant of the application's concrete types and allows the application to be ignorant of when and how the library event occurs.

   Delegates are used to support:
   - Publish and Subscribe processing
   - Binding to and transporting lambdas to scopes different from the scope of definition of the lambda
   - Starting threads and Tasks
   - Dispatching messages to processing blocks specific to a message property

6. Write all the code for a lambda that accepts a string, and writes that, concatenated[1] with a string defined in its local scope, on the Console. Show how to bind that to a delegate and execute in another scope.

   Answer:

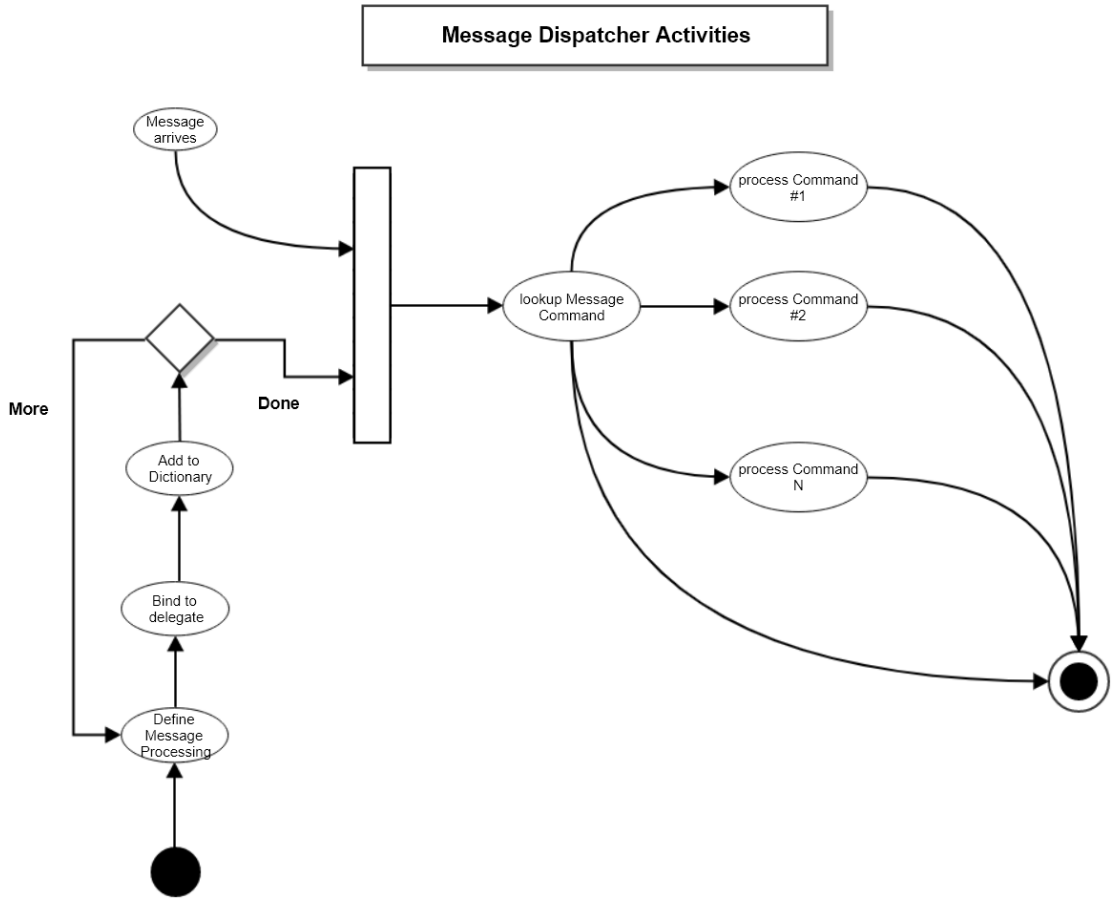
```
class MT1Q6
{
  static Action<string> makeLambda()
  {
    string staticMsg = "the lambda's message is \"";
    Action<string> act = (string msg) =>
      { Console.Write("\n  {0}\n\n", staticMsg + msg); };
    return act;
  }

  static void Main(string[] args)
  {
    Console.Write("\n  MT1Q6 - make lambda");
    Console.Write("\n ====================");

    makeLambda()("Hello CSE681 World");
  }
}
```

[1] You concatenate two strings by adding one to the end of the other.

7. Draw an activity diagram for a message dispatcher[2], used to handle messages coming into a receive queue. Describe how you would implement the message dispatcher.

   Answer:

   A message dispatcher can be implemented with a Dictionary<Msg.command, Action<Msg>>[3]. The Msg.command defines the type of processing needed for a message, and the Action<Msg> defines the processing used to handle that message.



[2] A message dispatcher invokes all the processing needed to handle each type of message, based on its command property.

[3] The Action<Msg> delegate would change to Func<Msg, Msg> for processing that returns a reply message.