

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 for a thread-safe stack in C#. Use the `System.Collections.Generic.Stack<T>` as the basis for your implementation.

Answer:

```
public class ThreadSafeStack<T>
{
    Stack<T> stack_ = new Stack<T>();
    object locker_ = new object();

    public void Push(T t)
    {
        lock (locker_)
        {
            stack_.Push(t);
            Monitor.Pulse(locker_);
        }
    }
    public T Pop()
    {
        lock (locker_)
        {
            while (stack_.Count == 0)
            {
                Monitor.Wait(locker_);
            }
            return stack_.Pop();
        }
    }
    public int Size()
    {
        lock(locker_)
        {
            return stack_.Count();
        }
    }
}
```

2. Discuss two distinct ownership policies for contents of the Code Repository of Project #5¹.

Answer:

Ownership describes who is allowed to check-in code to the repository. Some sensible policies are:

- Single owner:
Each item, e.g., document or source code package, has a single owner. Only that owner is allowed to check-in modifications to the item. Since one person is responsible for all changes to the item no check-out is required, nor should one be provided.
- Group owner with lock sub-policy:
Any member of the owning group can check-out, modify, and check-in an item owned by the group. That implies that forking and merging of the version chain will be necessary if concurrent check-outs are allowed. If not, then once a member checks-out an item it becomes locked and no other group member can check-out that item until the member holding the lock checks-in the item.
- No owner with lock sub-policy:
Any authenticated member of the project may check-out and check-in a document. This is equivalent to the group policy where there is only one group encompassing all members of the project. The same two cases apply, e.g., locking or no locking.

¹ The Project #5 statement is attached.

3. Show three distinct ways of defining and using C# delegates.

Answer:

```
class Program
{
    delegate void Traditional(string msg);
    event Traditional trad = new Traditional(handler);

    public static void handler(string msg)
    {
        Console.WriteLine("\n {0}", msg);
    }
    public class myArg : EventArgs
    {
        public string msg { get; set; }
    }
    public void handler2(object sender, EventArgs ev)
    {
        string msg = (ev as myArg).msg;
        Console.WriteLine("\n {0}", msg);
    }
    static void Main(string[] args)
    {
        Program p = new Program();
        p.trad.Invoke("Traditional delegate");

        Traditional anon = delegate(string msg)
        {
            Console.WriteLine("\n {0}", msg);
        };
        anon.Invoke("Anonymous delegate");

        Action<string> act = Program.handler;
        act.Invoke("Action delegate");

        Action<string> act2 = msg => Console.WriteLine("\n {0}", msg);
        act2.Invoke("Action using lambda");

        EventHandler eh = new EventHandler(p.handler2);
        myArg arg = new myArg();
        arg.msg = "EventHandler delegate";
        eh.Invoke(p, arg);

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

4. Describe the syntax and semantics for passing a reference type by reference to a member function of a class you are writing.

Answer:

Suppose that RT is a reference type. We declare and pass an instance of this type by reference to function fun with the syntaxes:

```
class X { public void fun(ref RT rt) { ... } ... }
```

```
RT rt = new RT();  
X x = new X();  
x.fun(ref rt);
```

When fun executes a new reference to the caller's rt is created on fun's stack frame. If the function changes the state of rt both caller and callee will see the change. If the function creates a new RT instance attached to its rt reference both the caller and callee will see the new instance.

5. What are the main parts of an Operational Concept Document? Why do we write OCDs?

Answer:

- Executive summary – summarizes concept
- Uses and users – how will users interact with product, what do they need? Impact on design of supporting those needs.
- Optional user interface views
- Partitions – top-level package diagram, responsibilities of each package, interactions between packages.
- Optional Activities – one or more activity diagrams, discussion of activities
- Critical Issues – statement of each issue, proposed solution(s)

We write OCDs to publish the concept for a product. That provides an invariant goal for development even if the product specifications are modified. It also helps to avoid one development team making invalid assumptions about what other team's work will provide.

6. Windows Communication Foundation supports three modes of invocation. One of these is OneWay which you set with an operation contract like this: `[OperationContract(IsOneWay=true)]`. Name and describe the behavior of each of the three modes.

Answer:

- Invoke and wait for response – this is the default invocation behavior even if the function called returns nothing. In that case the caller waits for fault messages should they be sent.
- OneWay – user invokes and returns immediately, not waiting for anything from the callee. If fault messages are sent the caller will not see them.
- Duplex – user invokes and returns immediately. Results are returned via a callback. Thus, the service declares two interfaces, the service contract and a callback contract. The service implements the first and the caller is expected to implement the second.

7. Draw an activity diagram for your planned implementation of Project #4. Briefly describe each of the activities.

Answer:

These activities focus on message-passing operations:

- Initialize Client and Server Objects – Receiver and Communicators
- Request Categories – ask server for category list for initial navigation view
- Listen for Message – enter ProcessMessages loop on child thread
- Interpret Messages – dispatcher posts messages to appropriate communicator for processing
- Prepare Reply – server’s communicator evaluates results and wraps in a message for the client
- Prepare View – client communicator uses Dispatcher.Invoke to make appropriate changes to view.
- User input – an external event so has no visible input flow.

