Examination #4

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 the processing provided by your solution for Project #2 or the Instructor's Solution, write a query or method to find all of the descendants of some element of the NoSql database. That is, find all of children, grandchildren, greatgrandchildren, … of that element[1].

   Answer[2]:

   The code below is now public member functions of the QueryEngine<Key, Value> class and also of the VirtualDB<Key, Value> class. The visited member is a private List<Key> in both those classes.

```
//----< find children of element associated with key >-------
/* Psuedo code:
   Return all children of input key as keys held by VirtualDB
*/
public VirtualDB<Key, Value> children(Key key)
{
  List<Key> children_ = new List<Key>();
  Value val = null;
  if (getValue(key, out val))
  {
    IDBElement<Key> elem = val as IDBElement<Key>;
    children_.AddRange(elem.children);
  }
  VirtualDB<Key, Value> vdb = new VirtualDB<Key, Value>(db);
  vdb.addKeys(children_);
  return vdb;
}
//----< find descendents of element associated with key >----
/* Psuedo code:
   Recursively visit each child of input key and
   mark as visited. Return all visited children
   as keys held by VirtualDB
*/
public VirtualDB<Key, Value> descendants(Key key)
{
  visited_.Clear();
  return descendantsHelper(key);
}
public VirtualDB<Key, Value> descendantsHelper(Key key)
{
  VirtualDB<Key, Value> vdb = children(key);
  foreach (Key child in vdb.Keys())
  {
    if (!visited_.Contains(child))  // - need to check visitation since dependencies
    {                               //   may have cycles, e.g., graph not tree
      visited_.Add(child);          // - since visited_ is a member field it doesn't
      descendantsHelper(child);     //   lose its values when recursing because we
    }                               //   don't need to create a new list on entry
  }
  vdb.Keys().Clear();
  vdb.addKeys(visited_);
  return vdb;
}
```

[1] Remember that the children of a DBElement<Key, Value> consist of keys of the elements on which the parent depends.

[2] There is a sneaky way to do this. Persist db then load into XDocument and use its descendents() function.

2. Given: class Widget { Widget clone() { … } /* other code */ }

write code for a class Clonable, that contains a List<Widget> as its only field, and provides a deep clone() operation. Please supply all methods involved in the cloning operation. You need not provide code for the Widget class and don't need to provide any other methods for the Clonable class. You may assume that the Widget.clone() method provides a new Widget object independent of any other Widget object.

Answer:

```
class Widget
{
  string name_ = null;

  public Widget(string name) { name_ = name; }
  public Widget clone()
  {
    return new Widget(name_);
  }
  public string show() { return name_; }
}
class MT3Q1
{
  List<Widget> low { get; set; } = new List<Widget>();

  public void add(Widget wgt) { low.Add(wgt); }

  public void add(List<Widget> Low) { low.AddRange(Low); }

  MT3Q1 clone()
  {
    MT3Q1 cloned = new MT3Q1();
    List<Widget> clow = new List<Widget>();
    foreach (Widget w in this.low)
      clow.Add(w.clone());
    cloned.add(clow);
    return cloned;
  }
```

3. Given an instance of some class, how would you discover if the class of that instance provides a specified method, without directly using the class definition. Please illustrate your answer with a code fragment[3].

   Answer:

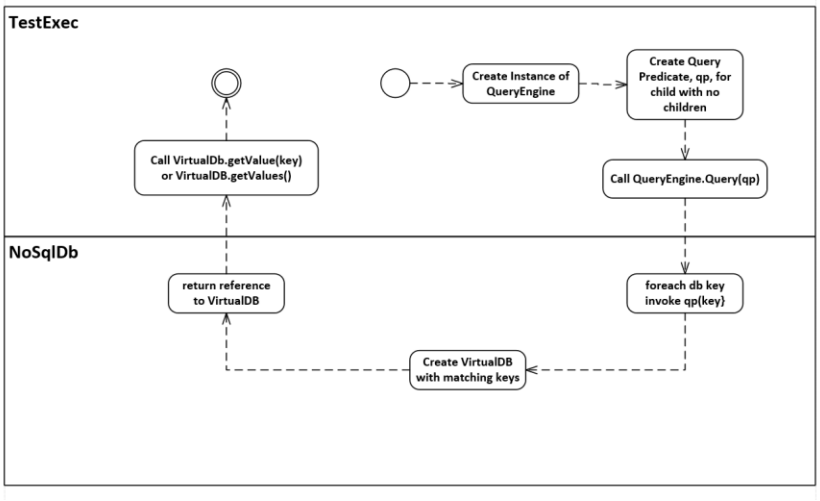
   You need to use reflection on the given instance as shown below:

```
SOMECLASS anInstance = new SOMECLASS();
System.Type t = anInstance.GetType();
Console.Write("\n  t.Name: {0}", t.Name);
Console.Write("\n  t.Namespace: {0}", t.Namespace);
System.Reflection.MethodInfo[] mi = t.GetMethods(
  BindingFlags.NonPublic | BindingFlags.Public |
  BindingFlags.Static | BindingFlags.Instance
);
foreach (var minfo in mi)
  Console.Write("\n    {0}", minfo.Name);
Console.Write("\n\n");
```

[3] A code fragment is a few lines of code that do not need to be a complete method.

4. Draw an activity diagram for the TestExec and NoSqlDb processing, from Project #2, required to demonstrate making a query for all elements that have a child key that corresponds to an element which has no children[4]. Discuss each of those activities briefly. You may use either your design or the design of the Instructor's Solution as a basis for your answer.

Answer:




```
//----< QueryPredicate as answer for MT4Q4, F2015 >-----
/* Pseudo Code:
   foreach element in db
   if element has children
     foreach element in children collection
       if element has no children
         return key of this element
   return empty keylist
*/
Func<string, bool> MakeQueryForNoGrandChildren(DBL pdb)
{
  Func<string, bool> qp = (queryKey) =>
  {
    DBElemL qelem;
    if (!pdb.getValue(queryKey, out qelem))
    {
      return false;
    }
    if (qelem.children.Count() > 0)
    {
      foreach(string key in qelem.children)
      {
        DBElemL testElem;
        if(pdb.getValue(key, out testElem))
        {
          if (testElem.children.Count() == 0)
            return true;
        }
      }
    }
    return false;
  };
  return qp;
}
```

[4] You may assume that an element with no children has an empty list of keys. That is, the list exists but has no elements.

5. Describe three of the instancing modes provided by Windows Communication Foundation (WCF) services. Why is an instancing mode important for a WCF service?

   Answer:

   The "**PerCall**" mode sets the 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.

   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. Enforcing an instancing mode is WCF's mechanism to manage the accumulation of objects within the service host.

6. Describe critical issues, along with proposed solutions, for the NoSql database you developed in Project #2.

   Answer:

   - DBElements are reference types. Each time a client executes getValue, using a key that exists in the database, the client gets a handle to the instance of that element that exists on the manage heap. The client can change the element without doing a specific insert of the edited element back into the database. This makes it very likely that users will accidentally make changes to stored data which may be very dangerous for applications that use the data.
     Solution:
     The solution provided by the Instructor's prototype supports returning query results as a clone of the existing database data. So even if the code making the query modifies the data it is modifying a clone, not the actual data in the application's data store.

   - The NoSql database implementation discussed in class does not provide indexes, transactions, or stored procedures – Things that most data designers use to manage the integrity and performance of their data applications.
     Solution:
     Transactions are fairly simple to implement as an add-on capability.
     We make whatever changes are required by the transacted operations using temporary keys. That is, the original data is not modified until we are sure that all the steps in the transaction have completed successfully. When all operations complete successfully we delete the original data and reinsert the modifications using the original keys. Note that does require removal of the old data, insertion of the modifications, and then removal of the temporary storage.

     If the transactions are distributed across several databases, nothing changes except we have to communicate the success or failure of the transaction to all machines involved in the operation.

     Indexes are nothing more than query results that we cache, so that the query only has to examine the values of the query's keys, not the values of all the keys in the database.

     Query predicates, as implemented in the Instructor's solution are essentially **stored procedures**. We simply need to provide a mechanism to name them, save them, and make them easy to execute.

   - How do we select contents of data shards – files persisted from selected parts of a database?
     Solution:
     Know your data, e.g., Repository would shard by subsystem, TestHarness by day.

   - How do we efficiently query information distributed across shards?
     Solution:
     Use another NoSqlDb. Each element captures the memory location of a shard as payload. We can use element metadata to support selection of specific shards for detailed queries.

7. Write all the code for a C# asynchronous member function[5] that can be stopped by a using application before its processing has completed. You may assume that the asynchronous method contains a loop that will take a significant amount of time to finish.

   Answer:

```
public class Stoppable
{
  Thread t = null;
  object sync_ = new object();
  Action act_ = null;
  bool stop = false;

  public void setAction(Action act) { act_ = act; }
  public void join() { t.Join(); }
  public void doStop()
  {
    lock(sync_) { stop = true; }
  }
  bool stopping()
  {
    lock(sync_) { return stop; }
  }
  public void asyncRun()
  {
    ThreadStart ts = () =>
    {
      while (true)
      {
        if (act_ == null || stopping())
          break;
        act_.Invoke();
        Thread.Sleep(5);
      }
    };
    t = new Thread(ts);
    t.Start();
  }
}
class MT4Q7
{
  static void Main(string[] args)
  {
    "MT4Q7 - Stoppable Asynchronous Function".title('=');
    Stoppable s = new Stoppable();
    Action act = () => Console.Write("\n  I'm running");
    s.setAction(act);
    s.asyncRun();
    Thread.Sleep(100);
    Console.Write("\n  -- stopping now --");
    s.doStop();
    s.join();
    Console.Write("\n\n");
  }
}
```

[5] An asynchronous method returns almost immediately without waiting for its code to complete.