Project #2 – Instructor's Solution – Code Fragments

Package: DBExtensions.cs
Class.Method: DBEngineExtensions.FromXml(XDocument doc)

```
public static Key convertStringToKey<Key>(string val)
{
  return (Key)(object)val;  // this works for ints and strings, may not for other types
}
//----< FromXml() extension >-----------------------------------------

public static bool FromXml<Key, Value, Data>(this DBEngine<Key, DBElement<Key, Data>> newDB, XDocument doc)
  where Data : class, IClone, IPersist, new()
{
  IEnumerable<XElement> keys = doc.Root.Elements("key");

  DBElement<Key, Data> dbElem;
  List<Key> dbKeys = newDB.Keys().ToList();

  foreach (var key in keys)
  {
    var dbKey = convertStringToKey<Key>(key.Value);
    if (dbKeys.Contains(dbKey))
    {
      Write("\n  database already contains key \"{0}\" so skipping insert", key.Value);
    }

    XElement elem = key.NextNode as XElement;  // get the <element> node
    if (elem == null)
    {
      Write("\n  unexpected null element");
      break;
    }
    dbElem = new DBElement<Key, Data>();
    dbElem.payload = new Data();
    IEnumerable<XElement> children = elem.Elements();
    foreach (var child in children)
    {
      switch (child.Name.ToString())
      {
        case "name":
          dbElem.name = child.Value;
```

Project #2 – Instructor’s Solution – Code Fragments

```
          break;
        case "descr":
          dbElem.descr = child.Value;
          break;
        case "timeStamp":
          dbElem.timeStamp = DateTime.Parse(child.Value);
          break;
        case "keys":
          IEnumerable<XElement> newkeys = child.Elements();
          foreach (var newkey in newkeys)
          {
            Key childKey = convertStringToKey<Key>(newkey.Value);
            dbElem.children.Add(childKey);
          }
          break;
        case "payload":
          dbElem.payload = dbElem.payload.FromXml(child.ToString()) as Data;
          break;
      }
    }
    newDB.insert(dbKey, dbElem);
  }
  return true;
}
```

Project #2 – Instructor’s Solution – Code Fragments

Package: PayloadWrapper.cs
Interfaces: IClone, IPersist
Classes: PayloadWrapper<Data>, PL_String<StringBuilder>, PL_ListOfStrings<List<string>>

```
  ///////////////////////////////////////////////////////////////////////
  // IClone interface
  // - contract for building copies of PayloadWrapper<Data>
  //   and DBElement<Key, Data>
  //
  public interface IClone
  {
    IClone Clone();
  }
  //////////////////////////////////////////////////////////////////////
  // IPersist interface
  // - contract for Persisting and Unpersisting PayloadWrapper<Data>
  //   and DBElement<Key, Data>
  //
  public interface IPersist
  {
    string ToXml();
    IPersist FromXml(string xml);
  }
  ///////////////////////////////////////////////////////////////////////
  // PayloadWrapper<Data> class
  // - provides guaranteed implementations of the IClone and IPersist
  //   interfaces needed to build generic DBElement<Key, Data> classes.
  //
  public abstract class PayloadWrapper<Data> : IClone, IPersist where Data : new()
  {
    public Data theWrappedData { get; set; } = new Data();
    abstract public IClone Clone();
    public abstract override bool Equals(object obj);
    public abstract override int GetHashCode();
    override abstract public string ToString();
    abstract public string ToXml();
    abstract public IPersist FromXml(string xml);
```

Project #2 – Instructor's Solution – Code Fragments

```
/////////////////////////////////////////////////////////////////////////
// PL_String class
// - wraps string payloads to support cloning and persistance
//
public class PL_String : PayloadWrapper<StringBuilder>
{
  public PL_String()
  {
    theWrappedData = new StringBuilder("");
  }
  public PL_String(string aString)
  {
    theWrappedData = new StringBuilder(aString);
  }
  public override IClone Clone()
  {
    PayloadWrapper<StringBuilder> cloned =
      new PL_String(String.Copy(theWrappedData.ToString()));
    return cloned;
  }
  public override string ToString()
  {
    return theWrappedData.ToString();
  }
  public override string ToXml()
  {
    StringBuilder accum =
      new StringBuilder(
        String.Format("\n    <payload>{0}</payload>", theWrappedData.ToString())
      );
    return accum.ToString();
  }
  public override IPersist FromXml(string xml)
  {
    XDocument doc = XDocument.Parse(xml);
    XElement payloadElem = doc.Descendants("payload").First();
    return new PL_String(payloadElem.Value);
  }
  public override bool Equals(object obj)
  {
    return theWrappedData.Equals(obj as StringBuilder);
  }
```

Project #2 – Instructor's Solution – Code Fragments

```
    public override int GetHashCode()
    {
      return theWrappedData.GetHashCode();
    }
  }
```

Project #2 – Instructor’s Solution – Code Fragments

```
////////////////////////////////////////////////////////////////////////
// PL_ListOfStrings class
// - wraps List<string> payloads to support cloning and persistance
//
public class PL_ListOfStrings : PayloadWrapper<List<string>>
{
  public PL_ListOfStrings()
  {
    theWrappedData = new List<string>();
  }
  public PL_ListOfStrings(List<string> list)
  {
    theWrappedData = list;
  }
  public override IClone Clone()
  {
    PL_ListOfStrings los = new PL_ListOfStrings();
    los.theWrappedData = new List<string>();
    foreach (string item in theWrappedData)
      los.theWrappedData.Add(String.Copy(item));
    return los;
  }
  public override bool Equals(object obj)
  {
    PayloadWrapper<List<string>> plw = obj as PayloadWrapper<List<string>>;
    if (theWrappedData.Count() != plw.theWrappedData.Count())
      return false;
    for (int i = 0; i < theWrappedData.Count(); ++i)
      if (theWrappedData[i] != plw.theWrappedData[i])
        return false;
    return true;
  }
  public override int GetHashCode()
  {
    return theWrappedData.GetHashCode();
  }
  public override string ToString()
  {
    StringBuilder accum = new StringBuilder();
    bool first = true;
    foreach (string item in theWrappedData)
    {
```

Project #2 – Instructor's Solution – Code Fragments

```
        if (first)
        {
          accum.Append(string.Format("{0}", item));
          first = false;
        }
        else
        {
          accum.Append(string.Format(", {0}", item));
        }
      }
      return accum.ToString();
    }
    public override string ToXml()
    {
      StringBuilder accum = new StringBuilder();
      accum.Append("\n    <payload>");
      foreach (string item in theWrappedData)
        accum.Append(string.Format("\n      <item>{0}</item>", item));
      accum.Append("\n    </payload>");
      return accum.ToString();
    }
    public override IPersist FromXml(string xml)
    {
      XDocument doc = XDocument.Parse(xml);
      XElement payloadElem = doc.Descendants("payload").First();
      IEnumerable<XElement> newitems = payloadElem.Elements();
      PL_ListOfStrings los = new PL_ListOfStrings();
      foreach (var newitem in newitems)
        los.theWrappedData.Add(newitem.Value);
      return los;
    }
  }
```

Project #2 – Instructor's Solution – Code Fragments

```
Package:    DBEngine.cs
Interface:  IQuery<Key, Value>
```

```
  ////////////////////////////////////////////////////////////////////////
  // interface IQuery<Key, Value>
  // - a contract for all database objects that can be queried, e.g.,
  //   DBEngine, QueryEngine, and VirtualDB (used to be DBFactory)
  //
  public interface IQuery<Key, Value>
  {
    bool getValue(Key key, out Value val);
    List<Value> getValues();
    List<Key> Keys();
    bool containsKey(Key key);
  }
```

Project #2 – Instructor's Solution – Code Fragments

Package: QueryEngine.cs
Classes: VirtualDB<Key, Value>, QueryEngine<Key, Value>

```
/////////////////////////////////////////////////////////////////////////
// class VirtualDB is a queryable container of keys
// - returned by Query() and doQueries() with database with keys of
//   elements in original database that match query(s).
// - can be cloned so users of clone can't change original database
//   values.
// - was called DBFactory but that name caused a lot of confusion so ...
//
public class VirtualDB<Key, Value> : IQuery<Key, Value>
  where Value : class, IClone, IPersist, new()
{
  private DBEngine<Key, Value> db;
  List<Key> keys = new List<Key>();
  public VirtualDB(DBEngine<Key, Value> database) { db = database; }
  public bool getValue(Key key, out Value value)
  {
    if (db.getValue(key, out value))
    {
      //value = value.Clone() as Value;  // client can't change db element
      return true;
    }
    value = null;
    return false;
  }
  public List<Value> getValues()
  {
    List<Value> values = new List<Value>();
    Value val;
    foreach (Key key in Keys())
    {
      getValue(key, out val);
      if (val != null)
      {
        //values.Add(val.Clone() as Value);
        values.Add(val);
      }
    }
    return values;
```

Project #2 – Instructor's Solution – Code Fragments

```
    }
    public List<Key> Query(Func<Key, bool> f)
    {
      List<Key> matchingKeys = new List<Key>();
      foreach (Key key in Keys())
      {
        if (f.Invoke(key))
        {
          matchingKeys.Add(key);
        }
      }
      return matchingKeys;
    }
    public List<Key> Keys()
    {
      return keys;
    }
    public bool containsKey(Key key)
    {
      return Keys().Contains(key);
    }
    public void addKey(Key key)
    {
      keys.Add(key);
    }
    public void addKeys(List<Key> addkeys)
    {
      keys.AddRange(addkeys);
    }
    public void clear() { keys.Clear(); }
    public void cloneDB()
    {
      var dbTemp = new DBEngine<Key, Value>();
      foreach(Key key in keys)
      {
        Value val;
        if (getValue(key, out val))
          dbTemp.insert(key, val.Clone() as Value);
      }
      db = dbTemp;
    }
  }
```

Project #2 – Instructor’s Solution – Code Fragments

```
/////////////////////////////////////////////////////////////////////////
// class QueryEngine<Key, Value> is responsible for handling
// queries into DBEngine<Key, Value> instances.
// - The test stub gives good examples of using this class
//
public class QueryEngine<Key, Value> : IQuery<Key, Value>
  where Value : class, IClone, IPersist, new()
{
  private DBEngine<Key, Value> db;
  private List<Func<Key, bool>> queryPredicates = new List<Func<Key, bool>>();
  public QueryEngine(DBEngine<Key, Value> database)
  {
    db = database;
  }
  public VirtualDB<Key, Value> Query(Func<Key, bool> f)
  {
    List<Key> matchingKeys = new List<Key>();
    foreach(Key key in db.Keys())
    {
      if(f.Invoke(key))
      {
        matchingKeys.Add(key);
      }
    }
    VirtualDB<Key, Value> vdb = new VirtualDB<Key, Value>(db);
    vdb.addKeys(matchingKeys);
    return vdb;
  }
  public void add(Func<Key, bool> qp)
  {
    queryPredicates.Add(qp);
  }
  public VirtualDB<Key, Value> doQueries()
  {
    VirtualDB<Key, Value> vdb = new VirtualDB<Key, Value>(db);
    IEnumerable<Key> temp = db.Keys();
    vdb.addKeys(temp.ToList<Key>());
    List<Key> matchingKeys = new List<Key>();
    foreach(var qp in queryPredicates)
    {
      matchingKeys = vdb.Query(qp).ToList<Key>();
      vdb.clear();
```

Project #2 – Instructor's Solution – Code Fragments

```
        vdb.addKeys(matchingKeys);
      }
      return vdb;
    }
    public bool getValue(Key key, out Value value)
    {
      if (db.getValue(key, out value))
        return true;
      value = null;
      return false;
    }
    public List<Value> getValues()
    {
      List<Value> values = new List<Value>();
      foreach(Key key in Keys())
      {
        Value val;
        if (getValue(key, out val) && val != null)
          values.Add(val);
      }
      return values;
    }
    public List<Key> Keys()
    {
      return db.Keys() as List<Key>;
    }
    public bool containsKey(Key key)
    {
      return db.containsKey(key);
    }
  }
```

Project #2 – Instructor’s Solution – Code Fragments

Package: **QueryEngine.cs**
Class.Method: **TestQueryEngine.Main**

```
    "Testing Queries".title();
      WriteLine();

      QueryEngine<string, DBElement<string, PL_ListOfStrings>> qe =
        new QueryEngine<string, DBElement<string, PL_ListOfStrings>>(db);

      Write("\n --- Testing QueryEngine<Key, Value>.Query(Func<Key, bool>) ---");
      WriteLine();

      "simple query for elements with children".title();

      Func<string, bool> qp = (string key) =>
      {
        DBElement<string, PL_ListOfStrings> qelem;
        if (db.getValue(key, out qelem))
          if (qelem.children.Count() > 0)
            return true;
        return false;
      };

      List<string> keys = qe.Query(qp).Keys() as List<string>;

      // Lambda to display query results

      Action display = () =>
      {
        foreach (string key in keys)
        {
          DBElement<string, PL_ListOfStrings> qelem;
          qe.getValue(key, out qelem);
          {
            Write("\n  {0} has {1} children", qelem.name, qelem.children.Count());
          }
        }
        WriteLine();
      };
```

Project #2 – Instructor’s Solution – Code Fragments

```
display.Invoke();

Write("\n --- Testing QueryEngine<Key, Value>.doQueries() ---");
WriteLine();

Func<string, bool> qp2 = (string key) =>
{
  DBElement<string, PL_ListOfStrings> qelem;
  if (db.getValue(key, out qelem))
    if (qelem.name.IndexOf('3') > -1)
      return true;
  return false;
};

"query for elements with names that contain the letter 3".title();
keys = qe.Query(qp2).Keys() as List<string>;
display.Invoke();

"query for elements with children".title();
qe.add(qp);
"query for elements with names that contain the letter 3".title();
qe.add(qp2);

keys = qe.doQueries().Keys() as List<string>;
display.Invoke();
Write("\n\n");
```

Project #2 – Instructor’s Solution – Code Fragments

```
Package:          Display
Class.Method:     DisplayExtensions.showView<Key, Value, Data>(Action<Key, DBElement<Key, Data>> view)
```

```
    //----< show user defined view - done by supplying a lambda >--------

    public static void showView<Key, Value, Data>(this IQuery<Key, Value> db, Action<Key, DBElement<Key, Data>> view)
      where Data : class, IClone, IPersist, new()
    {
      foreach (Key key in db.Keys())
      {
        Value val;
        if (db.getValue(key, out val) && val != null)
        {
          DBElement<Key, Data> elem = val as DBElement<Key, Data>;
          view.Invoke(key, elem);
        }
      }
    }
```

Package: Display
Class.Method: TestDisplay.showView<string, DBElemL, PL_ListOfStrings>(Action<string, DBElemL> view)

```
      Write("\n  --- Test Views ---");
      WriteLine();
      Action<string, DBElement<string, PL_ListOfStrings>> view =
        (string vKey, DBElement<string, PL_ListOfStrings> e) =>
        {
          Write("\n  Key:   {0}", vKey);
          Write("\n  Name:  {0}", e.name);
          Write("\n  Descr: {0}", e.descr);
          WriteLine();
        };
      newdb.showView<string, DBElemL, PL_ListOfStrings>(view);
```