```
using System;
using System.Collections.Generic;
using System.IO;

namespace Wk11_Task1
{
    public class Book
    {
        private string _Author, _Title, _Edition, _Publisher, _Year;
        public string Author { get { return _Author;  } }
        public string Title { get { return _Title; } }
        public string Edition { get { return _Edition; } }
        public string Publisher { get { return _Publisher; } }
        public string Year { get { return _Year; } }

        public Book(string title = "", string author = "",  string ed = "1", string pub = "Pearson
Education", string year = "2014")
        {
            _Author = author;
            _Title = title;
            _Edition = ed;
            _Publisher = pub;
            _Year = year;
        }

        public Book(Book aBook)
        {
            _Author = aBook.Author;
            _Title = aBook.Title;
            _Edition = aBook.Edition;
            _Publisher = aBook.Publisher;
            _Year = aBook.Year;
        }

        public override string ToString()
        {
            return string.Format("{0}, Ed. {1}, {2}\n\t-Author by: {3} - {4}",
                                         _Title, _Edition, _Year, _Author, _Publisher);
        }
    }
    public class Program
    {
        static void SaveFile(Book[] list)
        {
            FileStream fs = File.Open(@"h:\book.txt", FileMode.Create, FileAccess.Write);
            StreamWriter sw = new StreamWriter(fs);
            int i;
            for (i = 0; i < list.Length - 1; i++)
                sw.WriteLine("{0}\t{1}\t{2}\t{3}\t{4}",
                        list[i].Title, list[i].Author, list[i].Edition, list[i].Publisher, list[i].Year);
            // write last record next to EOF character
            sw.Write("{0}\t{1}\t{2}\t{3}\t{4}",
                    list[i].Title, list[i].Author, list[i].Edition, list[i].Publisher, list[i].Year);

            sw.Close();
        }

        static void ReadFile(List<Book> list)
        {
            FileStream fs = File.Open(@"h:\book.txt", FileMode.Open, FileAccess.Read);
            StreamReader sr = new StreamReader(fs);
            string data = sr.ReadLine();
            while(data != null)
            {
                string[] record = data.Split('\t');
                list.Add(new Book(record[0], record[1], record[2], record[3], record[4]));
                data = sr.ReadLine();
            }
            sr.Close();
        }
        static void Main(string[] args)
        {
           /*
            Book[] BookList = new Book[]
            { new Book("Visual C# 2008: How to Program", "Deitel, P.J., and Deitel, H.M.,", "3",
"Pearson Education", "2009"),
              new Book("Game Development Essentials", "Dunniway T.and Novak J.", "2",
"Cengage", "2008"),
              new Book("JavaScript: The Complete Reference", "Powell T. and Schneider F.",
"5", "McGraw-Hill", "2013"),
              new Book("Database Systems", "Rob P. anc Coronel C.", "12", "Course
Technology", "2013")
            };
            SaveFile(BookList); */

            List<Book> BookList = new List<Book>();
            ReadFile(BookList);
            foreach(Book b in BookList)
                Console.WriteLine("* {0}\n", b);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Xml.Serialization;
```

```
//Project->Add Reference
using System.Runtime.Serialization.Formatters.Soap;

namespace Wk11_Task2
{

    [Serializable]
    public class Book
    {
        private string _Author, _Title, _Edition, _Publisher, _Year;
        public string Author { get { return _Author; } set { _Author = value; } }
        public string Title { get { return _Title; } set { _Title = value; } }
        public string Edition { get { return _Edition; } set { _Edition = value; } }
        public string Publisher { get { return _Publisher; } set { _Publisher = value; } }
        public string Year { get { return _Year; } set { _Year = value; } }

        public Book()
        {
            _Author = "No Author";
            _Title = "Not Defined";
            _Edition = "1";
            _Publisher = "Pearson Education";
            _Year = "2014";
        }
        public Book(string title, string author,  string ed, string pub, string year)
        {
            _Author = author;
            _Title = title;
            _Edition = ed;
            _Publisher = pub;
            _Year = year;
        }

        public Book(Book aBook)
        {
            _Author = aBook.Author;
            _Title = aBook.Title;
            _Edition = aBook.Edition;
            _Publisher = aBook.Publisher;
            _Year = aBook.Year;
        }

        public override string ToString()
        {
            return string.Format("{0}, Ed. {1}, {2}\n\t-Author by: {3} - {4}",
                                 _Title, _Edition, _Year, _Author, _Publisher);
        }
    }

    public class Program
```

```
{
   static void SaveBinFile(Book[] list)
   {
      FileStream fs = File.Open(@"h:\book.bin", FileMode.Create, FileAccess.Write);
      BinaryFormatter bf = new BinaryFormatter();
      bf.Serialize(fs, list);
      fs.Close();
   }

   static void SaveSoapFile(Book[] list)
   {
      FileStream fs = File.Open(@"h:\book.soap", FileMode.Create, FileAccess.Write);
      SoapFormatter sf = new SoapFormatter();
      sf.Serialize(fs, list);
      fs.Close();
   }

   static void SaveXmlFile(Book[] list)
   {
      foreach (Book b in list) Console.WriteLine(b);

      FileStream fs = File.Open(@"h:\book.xml", FileMode.Create, FileAccess.Write);
      XmlSerializer xs = new XmlSerializer(typeof(Book[]));
      xs.Serialize(fs, list);
      fs.Close();
   }

   static Book[] ReadBinFile()
   {
      Book[] array = new Book[100];
      FileStream fs = File.Open(@"h:\book.bin", FileMode.Open, FileAccess.Read);
      BinaryFormatter bf = new BinaryFormatter();
      array = (Book[])bf.Deserialize(fs);
      fs.Close();
      return array;
   }

   static Book[] ReadSoapFile()
   {
      Book[] array = new Book[100];
      FileStream fs = File.Open(@"h:\book.soap", FileMode.Open, FileAccess.Read);
      SoapFormatter sf = new SoapFormatter();
      array = (Book[])sf.Deserialize(fs);
      fs.Close();
      return array;
   }
```

```
        static void Main(string[] args)
        {

          Book[] BookList = new Book[]
            { new Book("Visual C# 2008: How to Program", "Deitel, P.J., and Deitel, H.M.,", "3",
"Pearson Education", "2009"),
              new Book("Game Development Essentials", "Dunniway T.and Novak J.", "2",
"Cengage", "2008"),
              new Book("JavaScript: The Complete Reference", "Powell T. and Schneider F.",
"5", "McGraw-Hill", "2013"),
              new Book("Database Systems", "Rob P. anc Coronel C.", "12", "Course
Technology", "2013")
            };

            //SaveBinFile(BookList);
            //SaveSoapFile(BookList);
            //SaveXmlFile(BookList);
          FileStream fs = File.Open(@"h:\book.xml", FileMode.Create, FileAccess.Write);
          XmlSerializer xs = new XmlSerializer(typeof(Book[]));
          xs.Serialize(fs, BookList);
          fs.Close();


            fs = File.Open(@"h:\book.xml", FileMode.Open, FileAccess.Read);
           // XmlSerializer xs = new XmlSerializer(typeof(Book[]));
            Book[] array = (Book[])xs.Deserialize(fs);
            fs.Close();
            foreach(Book b in array)
                if (b != null) Console.WriteLine("* {0}\n", b);




/*
            Book[] BookList = ReadSoapFile();
            foreach(Book b in BookList)
                if( b != null ) Console.WriteLine("* {0}\n", b); */



        }
    }
}


using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
```

```
namespace Prac11_Task3
{
   [Serializable]
   public class Book
   {
      private string _Author, _Title, _Publisher;
      private int _Edition, _Year;

      public string Author { get { return _Author; } set { _Author = value; } }
      public string Title { get { return _Title; } set { _Title = value; } }
      public string Publisher { get { return _Publisher; } set { _Publisher = value; } }
      public int Edition { get { return _Edition; } set { _Edition = value; } }
      public int Year { get { return _Year; } set { _Year = value; } }
      public Book()
      {
         _Title = "No Title";   _Author = "Unknow";   _Publisher = "No Publisher";
         _Edition = 1;
         _Year = DateTime.Now.Year;
      }
      public Book(string title, string author, string publisher, int ed, int year)
      {
         _Title = title;     _Author = author;   _Publisher = publisher;
         _Edition = ed;
         _Year = year;
      }
      public override string ToString()
      {
         return string.Format("{0}, {1}, {2}, {3}, {4}",
                                    _Author, _Title, _Edition, _Publisher, _Year);
      }
   }

   public class MinMaxTemplate<T> where T : IComparable<T>
   {
      private List<T> _Items = new List<T>();
      public ReadOnlyCollection<T> Items { get { return _Items.AsReadOnly(); } }
      public void Add(T data) { _Items.Add(data); }
      public T Minimum()
      {
         T data = default(T);
         if (_Items.Count > 0)
         {
            data = _Items[0];
            for (int i = 1; i < _Items.Count; i++)
               if (_Items[i].CompareTo(data) < 0) data = _Items[i];
                     // swap data to store smallest
         }
         return data;
      }
```

```
        public T Maximum()
        {
            T data = default(T);
            if (_Items.Count > 0)
            {
                data = _Items[0];
                for (int i = 1; i < _Items.Count; i++)
                    if (_Items[i].CompareTo(data) > 0) data = _Items[i];
                        // swap data to store highest
            }

            return data;
        }
    }

    public class Program
    {
        static void Main(string[] args)
        {
            Book[] BookList = new Book[] {
                new Book("C# How to Program 2010", "Deitel P.M. and J.H", "Pearson
Education", 3, 2010),
                new Book("Games Development", "Dunniway T. and Novak J.", "Cengage", 2,
2008),
                new Book("JavaScript: The complete reference", "Powell T. and Scheinder F.",
"McGraw-Hill", 2, 2004),
                new Book("Database Systems", "Rob P. and Coronel C.", "Course Technology",
8, 2007)
            };

            MinMaxTemplate<int> YearList = new MinMaxTemplate<int>();
            foreach (Book b in BookList) YearList.Add(b.Year);
            Console.WriteLine("\nThe oldest book has publised in {0}", YearList.Minimum());
            Console.WriteLine("The lastest book has publised in {0}\n", YearList.Maximum());

            MinMaxTemplate<string> TitleList = new MinMaxTemplate<string>();
            foreach (Book b in BookList) TitleList.Add(b.Title);
            Console.WriteLine("\nThe first book title\n\t\"{0}\"", TitleList.Minimum());
            Console.WriteLine("\nThe last book title\n\t\"{0}\"\n", TitleList.Maximum());
        }
    }
}
```

Task 11.4

binary file

A file containing data that is in a format optimised for a computer to read, e.g., numbers are represented using twos-complement instead of ASCII/Unicode;

generic classes

Classes where one or more data types have been deferred, i.e., the class is developed without knowledge of the actual data type/s being stored/used, the data type being specified later when the class is instantiated (object creation);

generic methods

Methods where the data types of one or more parmaters is deferred, i.e., the function is developed without knowledge of the actual data type/s being stored/used, the data type being specified instead when the method is invoked;

generics

Similar to templates in C++, allows for classes and methods to be developed with one or more unknown data types which are provided later when the classes/methods are used;

object serialisation

The process of converting an object into a sequence of bits/bytes for writing to secondary storage, providing persistence for that data (de-serialisation is the opposite, converting bits/bytes into an object);

persistence (regarding data)

The characteristic or feature of data as to whether it outlives the execution of the program which created that data, i.e., data that persists is data that exists beyond the application that creates it, such as a word processor creating a document file;

random access

The ability to read/write data stored in a file out of order, i.e., there is no need to read the data file one record at a time from start to finish;

record (random access file)

The collection of related data together into a single logical element that is stored in a file, e.g., an object may be represented by a single record. By using a fixed length record, it is possible to access records in a binary file randomly by calculating their location using the number of the record to read/write and the length of an individual record, i.e., address = (record number - 1) * record length;

sequential access

The ability to read/write data stored in a file in order, i.e., records are read/written from the beginning of the file through to the end of the file;

stream

A concept used regularly throughout programming for input/output, whereby data is represented as a sequence of bytes with no apparent beginning, ending, or record boundary;

text file

A file containing data that is in a format optimised for human readability, e.g., numbers are represented using ASCII/Unicode instead of twos-complement, requiring the computer to convert data to/from this format;