Presentation Layer

MainWindow

- Factory : FactoryTrain()
- trains = Storage().Instance

1

1

Business Layer

FactoryTrain

+ BuildTrain(string Type)

1 1

Train

- _trainsID : String
- _departure : String
- _destination : String
- _departureDay : String
- _type : String
- _departureTime : String
- _intermediate : List <String>
- _bookings : List<Booking>
- _firstClass : Boolean
- _sleeperBerth : Boolean

+ TrainID {get; set;} : String
+ Departure {get; set;} : String
+ Destination {get; set;} : String
+ DateStart .{get; set;} : String
+Type {get; set;} : String
+ Time {get; set;} : String
+ Intermediate {get; set;} List <String>
+ Bookings {get; set;} List<Booking>
+ FirstClass {get; set;} : Boolean
+ SleeperBerth {get; set;} : Boolean
+ AddBooking()
+ AddIntermediate()
+ FindBooking()
+ ToString() : String

Booking

- _trainRef : String
- _departure : String
- _arrival : String
- _name : String
- _coach : String
- _cabin : Boolean
- _firstclass : Boolean
- _seat : Integer

+ TrainID {get; set;} : String
+ Departure {get; set;} : String
+ Arrival {get; set;} : String
+ Name {get; set;} : String
+ Coach {get; set;} : String
+ Cabin {get; set;} : Boolean
+ FirstClass {get; set;} : Boolean
+ Seat {get; set;} : Integer
+ ToString() : String

1..* 0..*

0..*

parent

child

expressTrain

+ Intermediate : List<string> (override)

stoppingTrain

child

sleeperTrain

- _departureTime : String
- _sleeperBerth : Boolean

+ Time : String (override)
+ SleeperBerth : Boolean (override)

1

Data Layer

Storage

- _formatter : BinaryFormatter
- _trainsDictionary : Dictionary<String, Train>
- store : Storage
- filename : String

+ AddB()
+ AddTrain()
+ Instance()
+ SerializeFile()
+ DeserializeFile()
+ GetListOfTrains {get;} : List<Train>
+ Bookings {get;} : List<Booking>

DESIGN PATTERNS

Factory

The factory pattern is used to select the right type of train between express train, stopping train and sleeper train.

Singleton

The singleton is used to make sure that only one instance of Storage is created.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.Xml.Serialization;
7  using System.IO;
8
9
10 namespace Business
11 {
12     [Serializable]
13     public class Train
14     {
15         /*
16          * Author:               40326941 Ismael Souf
17          * Description:          This is the trains class for the booking
               which
18          *                       defines the trains attributes
19          * Date last modified:   06/12/2018
20         */
21
22         // Declare private variables
23         private string _trainsID;
24         private string _departure;
25         private string _destination;
26         private string _type;
27         private string _departureTime;
28         private string _departureDay;
29         private bool _firstClass;
30         private bool _sleeperBerth;
31         private List<Booking> _bookings = new List<Booking>();
32         private List<string> _intermediate = new List<string>();
33
34         public Train()
35         {
36
37         }
38
39         //Train ID must be a code.
40         public string TrainID
41         {
42
43             get
44             {
45                 return _trainsID;
46             }
47             set
48             {
49                 if(value.Length != 4)
50                 {
51                     throw new ArgumentException("Please enter a Train ID");
52                 }
53                 else
54                 {
55                     _trainsID = value;
```

```
56                  }
57              }
58
59          }
60
61
62          //Departure of train
63          public string Departure
64          {
65              get
66              {
67                  return _departure;
68              }
69              set
70              {
71                  if (value != null)
72                  {
73                      _departure = value;
74                  }
75
76                  else
77                  {
78                      throw new ArgumentException("Please verify the departure
                        and destination.");
79                  }
80
81              }
82          }
83
84          //Destination of train
85          public string Destination
86          {
87              get
88              {
89                  return _destination;
90              }
91              set
92              {
93                  if (value != null)
94                  {
95                      _destination = value;
96                  }
97
98                  else
99                  {
100                     throw new ArgumentException("Please verify the departure
                        and destination.");
101                 }
102             }
103         }
104
105
106         //First class of train
107         public bool FirstClass
108         {
109             get
```

```
110             {
111                 return _firstClass;
112             }
113             set
114             {
115               _firstClass = value;
116             }
117         }
118
119         //Departure date of train
120         public string DateStart
121         {
122             get
123             {
124                 return _departureDay;
125             }
126             set
127             {
128
129                 if (value != null)
130                 {
131
132                     _departureDay = value;
133                 }
134                 else
135                 {
136
137                     throw new ArgumentException("Trains start date is not
                        valid DD/MM/YYYY");
138                 }
139             }
140         }
141
142         //Departure time of train
143         public virtual string Time
144         {
145             get
146             {
147                 return _departureTime;
148             }
149             set
150             {
151
152                 if (value != null)
153                 {
154                     _departureTime = value;
155                 }
156                 else
157                 {
158
159                     throw new ArgumentException("Departure time is not
                        valid");
160                 }
161             }
162         }
163         //Type of train
```

```
164         public string Type
165         {
166             get
167             {
168                 return _type;
169             }
170             set
171             {
172 
173                 if (value != null)
174                 {
175                     _type = value;
176                 }
177                 else
178                 {
179 
180                     throw new ArgumentException("Type of trains is not
                        valid");
181                 }
182             }
183         }
184         //Sleeper berth of train
185         public virtual bool SleeperBerth
186         {
187             get
188             {
189                 return _sleeperBerth;
190             }
191             set
192             {
193                 _sleeperBerth = value;
194             }
195         }
196 
197 
198         //Method to get intermediates of train
199         public virtual string getIntermediate
200         {
201             get
202             {
203                 var intermediateStop = String.Join(",", _intermediate);
204                 return intermediateStop;
205             }
206 
207         }
208         //List of string for intermediates
209         public virtual List<string> Intermediate
210         {
211             get
212             {
213                 {
214                     List<string> res = new List<string>();
215                     foreach (var v in _intermediate)
216                     {
217                         res.Add(v);
218                     }
```

```
219                     return res;
220                 }
221             }
222 
223         }
224         //Method to add intermediates in the List
225         public virtual void AddIntermediate(string intermediates)
226         {
227             _intermediate.Add(intermediates);
228         }
229 
230         //List of bookings for train
231         public List<Booking> Bookings
232         {
233             get
234             {
235                 return _bookings;
236             }
237             set
238             {
239 
240                 if (value == null)
241                 {
242 
243                     throw new ArgumentException("Passenger does not have any
                    bookings");
244                 }
245                 else
246                 {
247                     _bookings = value;
248                 }
249             }
250         }
251         //Method bool to find if a booking is free
252         public bool FindBooking(string coach, int seat)
253         {
254 
255             foreach (Booking p in _bookings)
256             {
257                 if (coach.Equals(p.Coach) && seat == p.Seat)
258                 {
259                     return true;
260                 }
261             }
262 
263             return false;
264         }
265 
266         //Method to add a booking
267         public void AddBooking(Booking booking)
268         {
269             _bookings.Add(booking);
270 
271         }
272 
273         public override string ToString()
```

```
274         {
275             return TrainID + " " + Departure + "-" + Destination + " " +
                  DateStart + " " + Time + " " + Type + " " + FirstClass + " " +
                  SleeperBerth + " " + getIntermediate;
276         }
277
278
279     }
280 }
281
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Business
{
        /*
         * Author:              40326941 Ismael Souf
         * Description:         This is a child class of Train which is an
           expressTrain
         * Date last modified:  06/12/2018
        */
    [Serializable]
    public class expressTrain : Train
    {

        public override List<string> Intermediate
        {
            get
            {
                return null;
            }
        }
    }
}

```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Business
8  {
9      /*
10          * Author:               40326941 Ismael Souf
11          * Description:          This is a child class of Train which is a
               stopping Train
12          * Date last modified:   06/12/2018
13         */
14
15     [Serializable]
16     public class stoppingTrain : Train
17     {
18
19     }
20
21 }
22
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Business
8  {
9         /*
10          * Author:               40326941 Ismael Souf
11          * Description:          This is a child class of Train which is an
             sleeperTrain and override methods
12          * Date last modified:   06/12/2018
13         */
14
15     [Serializable]
16     public class sleeperTrain : Train
17     {
18         private bool _sleeperBerth;
19         private string _departureTime;
20
21         public override bool SleeperBerth
22         {
23             get
24             {
25                 return _sleeperBerth;
26             }
27             set
28             {
29                 _sleeperBerth = value;
30             }
31         }
32
33         public override string Time
34         {
35             get
36             {
37                 return _departureTime;
38             }
39             set
40             {
41                 string strTime_Start = value;
42                 DateTime dateTime_Start = Convert.ToDateTime(strTime_Start);
43                 if (dateTime_Start.Hour >= 21)
44                 {
45                     _departureTime = value;
46                 }
47                 else
48                 {
49                     throw new ArgumentException("Sleeper departs after 21:00");
50                 }
51             }
52         }
53
54
55     }
```

```
56  }
57
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Business
8  {
9
10         /*
11          * Author:               40326941 Ismael Souf
12          * Description:          This is the FactoryTrain class which build the
               type of train
13          * Date last modified:   06/12/2018
14         */
15     [Serializable]
16     public class FactoryTrain
17     {
18         public Train BuildTrain(string type)
19         {
20             if (type.Contains("Express"))
21             {
22                 return new expressTrain();
23             }
24             else if (type.Contains("Stopping"))
25             {
26                 return new stoppingTrain();
27             }
28             else if (type.Contains("Sleeper"))
29             {
30                 return new sleeperTrain();
31             }
32             else
33             {
34                 throw new ArgumentException("Type does not exist");
35             }
36
37         }
38     }
39 }
40
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.IO;
7  using System.Runtime.Serialization;
8  using System.Runtime.Serialization.Formatters.Binary;
9  using Business;
10
11 namespace Data
12 {
13     public class Storage
14     {
15         /*
16          * Author:               40326941 Ismael Souf
17          * Description:          This is my storage class which add trains to dictionary.
18          *                       This class also save and load to a bin file using singleton and BinaryFormatter
19          * Date last modified:   06/12/2018
20         */
21
22
23         private static Storage store;
24
25         private Dictionary<string, Train> _trainDictionary;
26         private BinaryFormatter _formatter;
27
28         private const string filename = "trainsList.bin";
29
30         private static readonly object padlock = new object();
31
32         //Initializing instance of the class if there isn't one already
33         public static Storage Instance()
34         {
35             if (store == null)
36             {
37                 lock (padlock)
38                 {
39                     if (store == null)
40                     {
41                         store = new Storage();
42                     }
43                 }
44             }
45
46             return store;
47         }
48
49         //private constructor
50         private Storage()
51         {
52             _trainDictionary = new Dictionary<string, Train>();
53             _formatter = new BinaryFormatter();
54         }
```

```
55
56          //Method to add train into Dictionary
57          public void AddTrain(string trainID, Train train)
58          {
59              // Check if train with that ID already exists in the
                  trainDictionary
60              if (_trainDictionary.ContainsKey(trainID))
61              {
62                  throw new ArgumentException("Train with that ID already
                      exists");
63
64
65              }
66              else
67              {
68                  // Add train in the dictionary
69                  _trainDictionary.Add(trainID, train);
70              }
71          }
72
73          //Method to add booking train
74          public void AddBooking(string trainID, Booking booking)
75          {
76              // Check if train with that ID already exists in the
                  trainDictionary
77              if (_trainDictionary.ContainsKey(trainID))
78              {
79                  //Check if the Seat and Coach are free in order to book, if
                      yes add booking
80                  if (!_trainDictionary[trainID].FindBooking(booking.Coach,
                      booking.Seat))
81                  {
82
83                      _trainDictionary[trainID].AddBooking(booking);
84
85                  }
86                  else
87                  {
88                      throw new ArgumentException("Booking with that ID already
                      exists");
89
90                  }
91              }
92              else
93              {
94                  throw new ArgumentException("Train with that ID not found");
95              }
96          }
97
98
99          public void SerializeFile()
100         {
101             try
102             {
103                 // Create a FileStream that will write data to file.
104                 FileStream writer = new FileStream(filename, FileMode.Create,
```

```
                    FileAccess.Write);
105                 // Save our dictionary of trains to file
106                 this._formatter.Serialize(writer, _trainDictionary);
107                 // Close the writer FileStream when we are done.
108                 writer.Close();
109             }
110             catch (Exception)
111             {
112                 throw new ArgumentException("Unable to save our trains
                      informations.");
113             }
114         }
115
116         public void DeserializeFile()
117         {
118
119             // Check if we had previously save informations of our trains
120             if (File.Exists(filename))
121             {
122
123                 try
124                 {
125                     // Create a FileStream that will read access to the data
                        file.
126                     FileStream reader = new FileStream(filename,
                        FileMode.Open, FileAccess.Read);
127                     // Reconstruct from file.
128                     _trainDictionary = (Dictionary<String, Train>)
129                     _formatter.Deserialize(reader);
130                     // Close the reader FileStream when we are done
131                     reader.Close();
132
133                 }
134                 catch (Exception)
135                 {
136                     throw new ArgumentException("Problem occurred with the
                        file.");
137                 }
138
139             }
140
141         }
142
143         //Get our list of trains
144         public List<Train> GetListOfTrains
145         {
146             get
147             {
148                 // Create a new list of type train
149                 List<Train> trains = new List<Train>();
150
151                 if (_trainDictionary.Count > 0)
152                 {
153                     // Loop through each trains in the dictionary
154                     foreach (Train train in _trainDictionary.Values)
155                     {
```

```
156                         trains.Add(train);
157                     }
158                 }
159                 return trains;
160 
161             }
162         }
163 
164         //Get our list of bookings
165         public List<Booking> Bookings
166         {
167             get
168             {
169                 // Create a new list of type booking
170                 List<Booking> bookings = new List<Booking>();
171 
172                 if (_trainDictionary.Count > 0)
173                 {
174                     // Loop through each trains in the dictionary
175                     foreach (Train c in _trainDictionary.Values)
176                     {
177                         // Loop through each booking in the current trains
                        bookings list
178                         foreach (Booking b in c.Bookings)
179                         {
180                             // Add the current booking to the list of bookings
181                             bookings.Add(b);
182                         }
183                     }
184                 }
185                 return bookings;
186             }
187 
188         }
189     }
190 }
191 
192 
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.ComponentModel;
 4  using System.IO;
 5  using System.Linq;
 6  using System.Text;
 7  using System.Threading.Tasks;
 8  using System.Windows;
 9  using System.Windows.Controls;
10  using System.Windows.Data;
11  using System.Windows.Documents;
12  using System.Windows.Input;
13  using System.Windows.Media;
14  using System.Windows.Media.Imaging;
15  using System.Windows.Navigation;
16  using System.Windows.Shapes;
17  using System.Xml.Serialization;
18  using Business;
19  using Data;
20
21
22  namespace Presentation
23  {
24
25      /// <summary>
26      /// Interaction logic for MainWindow.xaml
27      /// </summary>
28      public partial class MainWindow : Window
29      {
30
31          FactoryTrain Factory = new FactoryTrain();
32          Storage trains = Storage.Instance();
33
34          public MainWindow()
35          {
36
37              InitializeComponent();
38              trains.DeserializeFile();
39
40              foreach (var s in trains.GetListOfTrains)
41              {
42                  lbxTrain.Items.Add(s);
43              }
44
45              foreach (var u in trains.Bookings)
46              {
47                  lbxBooking.Items.Add(u);
48              }
49
50              DateTime time = DateTime.Today;
51
52              //Loop through time and populate to the appropriate combobox
53              for (DateTime _time = time.AddHours(0); _time < time.AddHours(24); ↵
                    _time = _time.AddMinutes(30))
54              {
55                  timeBox.Items.Add(_time.ToShortTimeString());
```

```
56              }
57
58              //Hide visibility of intermediates until type is selected
59              chkPeterborough.Visibility = Visibility.Hidden;
60              chkDarlington.Visibility = Visibility.Hidden;
61              chkYork.Visibility = Visibility.Hidden;
62              chkNewcastle.Visibility = Visibility.Hidden;
63              lblintermediate.Visibility = Visibility.Hidden;
64
65              // Populate the seat combo box 1-60
66              populateSeatList();
67
68          }
69
70          private void Window_Closed(object sender, CancelEventArgs e)
71          {
72              // Serialize the trains list including booking
73              trains.SerializeFile();
74          }
75
76
77
78          private void AddTrains_Click(object sender, RoutedEventArgs e)
79          {
80              //Runs add train method
81              AddTrain();
82          }
83
84
85          public void AddTrain()
86          {
87
88
89              try
90              {
91                  //If the type of train is not selected
92                  if (typeBox.SelectedIndex == -1)
93                  {
94                      throw new ArgumentException("Type undefined");
95                  }
96
97                  //Use type selected to get the right type of train in the
                      factory
98                  var train = Factory.BuildTrain(typeBox.SelectedItem.ToString
                      ());
99
100                 if (typeBox.SelectedItem != null)
101                 {
102                     train.Type = typeBox.SelectionBoxItem.ToString();
103                 }
104
105                 if (txtTrainsID.Text.Length != 4)
106                 {
107                     throw new ArgumentException("Invalid Trains ID.");
108                 }
109                 else
```

```
110                 {
111                     train.TrainID = txtTrainsID.Text;
112                 }
113 
114                 if (departureBox.SelectedIndex == 0)
115                 {
116                     train.Departure = "Edinburgh(Waverly)";
117                 }
118                 else if (departureBox.SelectedIndex == 1)
119                 {
120                     train.Departure = "London(Kings Cross)";
121                 }
122                 else
123                 {
124                     throw new ArgumentException("Invalid data for the
                    departure station!");
125                 }
126 
127                 if (destinationBox.SelectedIndex == 0)
128                 {
129                     train.Destination = "Edinburgh(Waverly)";
130                 }
131                 else if (destinationBox.SelectedIndex == 1)
132                 {
133                     train.Destination = "London(Kings Cross)";
134                 }
135                 else
136                 {
137                     throw new ArgumentException("Invalid data for the
                    destination station!");
138                 }
139 
140                 if (train.Departure == train.Destination)
141                 {
142                     throw new ArgumentException("Departure and Destination can
                     not be the same!");
143                 }
144 
145                 //If a date is selected in the departure picker, give the date
                   selected
146                 if (departurePicker.SelectedDate != null)
147                 {
148                     train.DateStart =
                    departurePicker.SelectedDate.Value.ToShortDateString();
149                 }
150                 else
151                 {
152                     throw new ArgumentException("Please select a departure
                    day.");
153                 }
154 
155                 //If first class is checked the train has first class
156                 if (chkFirstClass.IsChecked.Value)
157                 {
158                     train.FirstClass = true;
159                 }
```

```
160                 else
161                 {
162                     train.FirstClass = false;
163                 }
164 
165                 //If the type of train is a sleeper and sleeper berth is
                      checked then the train has sleeperberth
166                 if (chkSleeperBerth.IsChecked.Value && typeBox.SelectedIndex
                      == 2)
167                 {
168                     train.SleeperBerth = true;
169                 }
170                 else
171                 {
172                     train.SleeperBerth = false;
173                 }
174 
175                 //Convert string time to Datetime
176                 if (timeBox.SelectedIndex >= 0)
177                 {
178                     string strTime_Start = timeBox.SelectedItem.ToString();
179                     DateTime dateTime_Start = Convert.ToDateTime
                        (strTime_Start);
180                     train.Time = dateTime_Start.ToShortTimeString();
181 
182                 }
183                 else
184                 {
185                     throw new ArgumentException("Please select a departure
                        time");
186                 }
187 
188                 //If station is checked, add it to the train.
189                 if (chkPeterborough.IsChecked == true)
190                 {
191                     train.AddIntermediate("Peteborough");
192                 }
193 
194                 if (chkDarlington.IsChecked == true)
195                 {
196                     train.AddIntermediate("Darlington");
197                 }
198 
199                 if (chkYork.IsChecked == true)
200                 {
201                     train.AddIntermediate("York");
202                 }
203 
204                 if (chkNewcastle.IsChecked == true)
205                 {
206                     train.AddIntermediate("Newcastle");
207                 }
208 
209                 //Add trains
210                 trains.AddTrain(train.TrainID, train);
211                 //Display train properties
```

```
212                 lbxTrain.Items.Add(train);
213
214                 MessageBox.Show("Train added !");
215
216             }
217             catch (ArgumentException excep)
218             {
219                 MessageBoxButton btnMessageBox = MessageBoxButton.OK;
220                 string caption = "Trains Error";
221                 MessageBox.Show(excep.Message, caption, btnMessageBox);
222             }
223
224         }
225
226
227
228
229         public void populateSeatList()
230         {
231             int maxSeat = 60;
232             // Loop 60 times
233             for (int i = 1; i <= maxSeat; i++)
234             {
235                 seatBox.Items.Add(i.ToString());
236             }
237         }
238
239         private void CheckBox_Checked(object sender, RoutedEventArgs e)
240         {
241
242         }
243
244         private void timeBox_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
245         {
246
247         }
248
249         private void ListBox_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
250         {
251
252
253         }
254
255         private void typeBox_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
256         {
257             //If the type of train is Stopping or Sleeper, show intermediates
258             if (typeBox.SelectedIndex >= 1)
259             {
260                 chkPeterborough.Visibility = Visibility.Visible;
261                 chkDarlington.Visibility = Visibility.Visible;
262                 chkYork.Visibility = Visibility.Visible;
263                 chkNewcastle.Visibility = Visibility.Visible;
```

```
264                 lblintermediate.Visibility = Visibility.Visible;
265
266             }
267             else
268             {
269                 chkPeterborough.Visibility = Visibility.Hidden;
270                 chkDarlington.Visibility = Visibility.Hidden;
271                 chkYork.Visibility = Visibility.Hidden;
272                 chkNewcastle.Visibility = Visibility.Hidden;
273                 lblintermediate.Visibility = Visibility.Hidden;
274
275             }
276
277         }
278
279
280
281         private void intermediateBox_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
282         {
283
284         }
285         //Add booking when button is pressed
286         private void btnAddBooking_Click(object sender, RoutedEventArgs e)
287         {
288
289             Booking passenger = new Booking();
290             Train train = new Train();
291             int _interCost = 25;
292             int _mainCost = 50;
293
294             try
295             {
296
297                //If txtTrainBook is null or empty train ID is invalid
298                 if(String.IsNullOrEmpty(txtTrainBook.Text))
299                 {
300                     throw new ArgumentException("Train ID is invalid");
301                 }
302                 else
303                 {
304                     passenger.TrainID = txtTrainBook.Text;
305                 }
306
307                 //If txtName is null or blank name is invalid
308                 if (String.IsNullOrWhiteSpace(txtName.Text))
309                 {
310                     throw new ArgumentException("Please enter a name for
                        booking");
311                 }
312                 else
313                 {
314                     passenger.Name = txtName.Text;
315                 }
316
317                 if (coachBox.SelectedIndex == -1)
```

```
318                 {
319                     throw new ArgumentException("Please select a Coach for
                    booking");
320                 }
321                 else
322                 {
323                     passenger.Coach = coachBox.SelectionBoxItem.ToString();
324                 }
325
326
327                 if (seatBox.SelectedIndex == -1)
328                 {
329                     throw new ArgumentException("Please select a Seat for
                    booking");
330                 }
331                 else
332                 {
333                     passenger.Seat = Convert.ToInt32
                    (seatBox.SelectionBoxItem.ToString());
334                 }
335
336                 if (departBox.SelectedIndex == -1 || arrivalBox.SelectedIndex
                  == -1)
337                 {
338                     throw new ArgumentException("Please select a Departure and
                     Arrival station");
339                 }
340                 else if (departBox.SelectionBoxItem.ToString() ==
                  arrivalBox.SelectionBoxItem.ToString())
341                 {
342                     throw new ArgumentException("Departure and Arrival must be
                     different");
343                 }
344                 else
345                 {
346                     passenger.Departure = departBox.SelectionBoxItem.ToString
                    ();
347                     passenger.Arrival = arrivalBox.SelectionBoxItem.ToString
                    ();
348                 }
349
350                 if (train.SleeperBerth == true && chkCabin.IsChecked == true)
351                 {
352                     passenger.Cabin = true;
353                 }
354                 else if (train.SleeperBerth == true && chkCabin.IsChecked ==
                  false)
355                 {
356                     passenger.Cabin = false;
357                 }
358                 else if (train.SleeperBerth == false)
359                 {
360                     passenger.Cabin = false;
361                 }
362                 else
363                 {
```

```
364                     throw new ArgumentException("Train or coach does not offer
                         sleeper berth");
365                 }
366
367                 if (train.FirstClass == true && chkFirstClass2.IsChecked ==
                      true)
368                 {
369                     passenger.FirstClass = true;
370                 }
371                 else if (train.FirstClass == true && chkFirstClass2.IsChecked
                      == false)
372                 {
373                     passenger.FirstClass = false;
374                 }
375                 else if (train.FirstClass == false)
376                 {
377                     passenger.FirstClass = false;
378                 }
379                 else
380                 {
381                     throw new ArgumentException("Train or coach does not offer
                         first class");
382                 }
383
384
385                 //Set booking fare for trains
386                 if (departBox.SelectionBoxItem.ToString() == "Edinburgh
                      (Waverly)" && arrivalBox.SelectionBoxItem.ToString() ==
                      "London(Kings Cross)")
387                 {
388                     if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked
                         == true)
389                     {
390                         switch (MessageBox.Show("Booking will cost £" + 90,
                        "Booking Fare", MessageBoxButton.YesNoCancel,
                        MessageBoxImage.Question))
391                         {
392                             case MessageBoxResult.Yes:
393                                 MessageBox.Show("Booking purchased");
394                                 break;
395
396                             case MessageBoxResult.No:
397                                 return;
398
399
400                             case MessageBoxResult.Cancel:
401                                 return;
402
403                         }
404                     }
405
406                     if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked
                         == false)
407                     {
408                         switch (MessageBox.Show("Booking will cost £" + 80,
                        "Booking Fare", MessageBoxButton.YesNoCancel,
```

```
                    MessageBoxImage.Question))
409                     {
410                         case MessageBoxResult.Yes:
411                             MessageBox.Show("Booking purchased");
412                             break;
413 
414                         case MessageBoxResult.No:
415                             return;
416 
417 
418                         case MessageBoxResult.Cancel:
419                             return;
420 
421                     }
422                 }
423                 if (chkFirstClass2.IsChecked == true && chkCabin.IsChecked
                     == false)
424                 {
425                     switch (MessageBox.Show("Booking will cost £" + 60,
                    "Booking Fare", MessageBoxButton.YesNoCancel,
                    MessageBoxImage.Question))
426                     {
427                         case MessageBoxResult.Yes:
428                             MessageBox.Show("Booking purchased");
429                             break;
430 
431                         case MessageBoxResult.No:
432                             return;
433 
434 
435                         case MessageBoxResult.Cancel:
436                             return;
437 
438                     }
439                 }
440 
441                 if(chkFirstClass2.IsChecked == false && chkCabin.IsChecked
                     == false)
442                 {
443                     switch (MessageBox.Show("Booking will cost £" + 50,
                    "Booking Fare", MessageBoxButton.YesNoCancel,
                    MessageBoxImage.Question))
444                     {
445                         case MessageBoxResult.Yes:
446                             MessageBox.Show("Booking purchased");
447                             break;
448 
449                         case MessageBoxResult.No:
450                             return;
451 
452 
453                         case MessageBoxResult.Cancel:
454                             return;
455 
456                     }
457                 }
```

```
458
459
460                 }
461                 else if (departBox.SelectionBoxItem.ToString() == "London
                      (Kings Cross)" && arrivalBox.SelectionBoxItem.ToString() ==
                      "Edinburgh(Waverly)")
462                 {
463                     if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked
                       == true)
464                     {
465                         switch (MessageBox.Show("Booking will cost £" + 90,
                        "Booking Fare", MessageBoxButton.YesNoCancel,
                        MessageBoxImage.Question))
466                         {
467                             case MessageBoxResult.Yes:
468                                 MessageBox.Show("Booking purchased");
469                                 break;
470
471                             case MessageBoxResult.No:
472                                 return;
473
474
475                             case MessageBoxResult.Cancel:
476                                 return;
477
478                         }
479                     }
480                     if (chkFirstClass2.IsChecked == true)
481                     {
482                         switch (MessageBox.Show("Booking will cost £" + 60,
                        "Booking Fare", MessageBoxButton.YesNoCancel,
                        MessageBoxImage.Question))
483                         {
484                             case MessageBoxResult.Yes:
485                                 MessageBox.Show("Booking purchased");
486                                 break;
487
488                             case MessageBoxResult.No:
489                                 return;
490
491
492                             case MessageBoxResult.Cancel:
493                                 return;
494
495                         }
496                     }
497                     else
498                     {
499                         switch (MessageBox.Show("Booking will cost £" + 50,
                        "Booking Fare", MessageBoxButton.YesNoCancel,
                        MessageBoxImage.Question))
500                         {
501                             case MessageBoxResult.Yes:
502                                 MessageBox.Show("Booking purchased");
503                                 break;
504
```

```
505                             case MessageBoxResult.No:
506                                 return;
507
508
509                             case MessageBoxResult.Cancel:
510                                 MessageBox.Show("Booking Cancelled");
511                                 return;
512
513                         }
514                     }
515                 }
516
517
518                 if ((departBox.SelectionBoxItem.ToString() == "Peterborough"
                      || departBox.SelectionBoxItem.ToString() == "Darlington" ||
                      departBox.SelectionBoxItem.ToString() == "York" ||
                      departBox.SelectionBoxItem.ToString() == "NewCastle" ||
                      departBox.SelectionBoxItem.ToString() == "Edinburgh
                      (Waverly)" || departBox.SelectionBoxItem.ToString() ==
                      "London(Kings Cross)") &&
                      (arrivalBox.SelectionBoxItem.ToString() == "Peterborough" ||
                       arrivalBox.SelectionBoxItem.ToString() == "Darlington" ||
                      arrivalBox.SelectionBoxItem.ToString() == "York" ||
                      arrivalBox.SelectionBoxItem.ToString() == "Newcastle"))
519                 {
520                     if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked
                        == true)
521                     {
522                         switch (MessageBox.Show("Booking will cost £" + 65,
                        "Booking Fare", MessageBoxButton.YesNoCancel,
                        MessageBoxImage.Question))
523                         {
524                             case MessageBoxResult.Yes:
525                                 MessageBox.Show("Booking purchased");
526                                 break;
527
528                             case MessageBoxResult.No:
529                                 return;
530
531
532                             case MessageBoxResult.Cancel:
533                                 return;
534
535                         }
536                     }
537
538                     if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked
                        == false)
539                     {
540                         switch (MessageBox.Show("Booking will cost £" + 55,
                        "Booking Fare", MessageBoxButton.YesNoCancel,
                        MessageBoxImage.Question))
541                         {
542                             case MessageBoxResult.Yes:
543                                 MessageBox.Show("Booking purchased");
544                                 break;
```

```
545
546                             case MessageBoxResult.No:
547                                 return;
548
549
550                             case MessageBoxResult.Cancel:
551                                 return;
552
553                         }
554                     }
555
556                     if (chkFirstClass2.IsChecked == true && chkCabin.IsChecked
                         == false)
557                     {
558                         switch (MessageBox.Show("Booking will cost £" + 35,
                        "Booking Fare", MessageBoxButton.YesNoCancel,
                        MessageBoxImage.Question))
559                         {
560                             case MessageBoxResult.Yes:
561                                 MessageBox.Show("Booking purchased");
562                                 break;
563
564                             case MessageBoxResult.No:
565                                 return;
566
567
568                             case MessageBoxResult.Cancel:
569                                 return;
570
571                         }
572                     }
573
574                     if(chkFirstClass2.IsChecked == false && chkCabin.IsChecked
                         == false)
575                     {
576                         switch (MessageBox.Show("Booking will cost £" + 25,
                        "Booking Fare", MessageBoxButton.YesNoCancel,
                        MessageBoxImage.Question))
577                         {
578                             case MessageBoxResult.Yes:
579                                 MessageBox.Show("Booking purchased");
580                                 break;
581
582                             case MessageBoxResult.No:
583                                 return;
584
585
586                             case MessageBoxResult.Cancel:
587                                 MessageBox.Show("Booking Cancelled");
588                                 return;
589
590                         }
591                     }
592                 }
593
594                 else if ((departBox.SelectionBoxItem.ToString() ==
```

```
                    "Peterborough" || departBox.SelectionBoxItem.ToString() ==
                    "Darlington" || departBox.SelectionBoxItem.ToString() ==
                    "York" || departBox.SelectionBoxItem.ToString() ==
                    "Newcastle") && (arrivalBox.SelectionBoxItem.ToString() ==
                    "Peterborough" || arrivalBox.SelectionBoxItem.ToString() ==
                    "Darlington" || arrivalBox.SelectionBoxItem.ToString() ==
                    "York" || arrivalBox.SelectionBoxItem.ToString() ==
                    "Newcastle" || arrivalBox.SelectionBoxItem.ToString() ==
                    "Edinburgh(Waverly)" || arrivalBox.SelectionBoxItem.ToString
                    () == "London(Kings Cross)"))
595                 {
596                     if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked
                     == true)
597                     {
598                         switch (MessageBox.Show("Booking will cost £" + 65,
                    "Booking Fare", MessageBoxButton.YesNoCancel,
                    MessageBoxImage.Question))
599                         {
600                             case MessageBoxResult.Yes:
601                                 MessageBox.Show("Booking purchased");
602                                 break;
603 
604                             case MessageBoxResult.No:
605                                 return;
606 
607 
608                             case MessageBoxResult.Cancel:
609                                 return;
610 
611                         }
612                     }
613 
614                     if (chkCabin.IsChecked == true)
615                     {
616                         switch (MessageBox.Show("Booking will cost £" + 55,
                    "Booking Fare", MessageBoxButton.YesNoCancel,
                    MessageBoxImage.Question))
617                         {
618                             case MessageBoxResult.Yes:
619                                 MessageBox.Show("Booking purchased");
620                                 break;
621 
622                             case MessageBoxResult.No:
623                                 return;
624 
625 
626                             case MessageBoxResult.Cancel:
627                                 return;
628 
629                         }
630                     }
631 
632                     if (chkFirstClass2.IsChecked == true)
633                     {
634                         switch (MessageBox.Show("Booking will cost £" + 35,
                    "Booking Fare", MessageBoxButton.YesNoCancel,
```

```
                    MessageBoxImage.Question))
635                     {
636                         case MessageBoxResult.Yes:
637                             MessageBox.Show("Booking purchased");
638                             break;
639 
640                         case MessageBoxResult.No:
641                             return;
642 
643 
644                         case MessageBoxResult.Cancel:
645                             return;
646 
647                     }
648                 }
649                 else
650                 {
651 
652                     switch (MessageBox.Show("Booking will cost £" + 25,    ↵
                    "Booking Fare", MessageBoxButton.YesNoCancel,              ↵
                    MessageBoxImage.Question))
653                     {
654                         case MessageBoxResult.Yes:
655                             MessageBox.Show("Booking purchased");
656                             break;
657 
658                         case MessageBoxResult.No:
659                             return;
660 
661 
662                         case MessageBoxResult.Cancel:
663                             MessageBox.Show("Booking Cancelled");
664                             return;
665 
666                     }
667                 }
668 
669               }
670 
671             trains.AddBooking(passenger.TrainID, passenger);
672             lbxBooking.Items.Add(passenger);
673 
674         }
675         catch (Exception exception)
676         {
677             MessageBox.Show(exception.Message);
678         }
679 
680 
681     }
682 
683 
684     private void listbox2_SelectionChanged(object sender,                  ↵
          SelectionChangedEventArgs e)
685     {
686 
```

```
687
688         }
689
690         private void FindTrain_Click(object sender, RoutedEventArgs e)
691         {
692
693             lbxDate.Items.Clear();
694
695             //For each trains if a train has a selected date, display the
                  trainID, the date and time of departure
696             foreach (var res in trains.GetListOfTrains)
697             {
698
699                 if (res.DateStart.Equals(departurePicker.Text))
700                 {
701                     lbxDate.Items.Add(res.TrainID + " " + res.DateStart + " "
                    + res.Time);
702                 }
703             }
704
705
706         }
707
708         private void departureBox_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
709         {
710
711         }
712
713         private void timeBox_SelectionChanged_1(object sender,
              SelectionChangedEventArgs e)
714         {
715
716         }
717
718         private void destinationBox_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
719         {
720
721         }
722
723         private void listbox3_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
724         {
725
726         }
727
728         private void departBox_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
729         {
730
731         }
732
733         private void arrivalBox_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
734         {
```

```
735
736         }
737
738         private void coachBox_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
739         {
740
741         }
742
743         private void seatBox_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
744         {
745
746         }
747
748         private void lbxDate_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
749         {
750
751         }
752
753     }
754 }
755
```

```
 1
 2  using System.Collections.Generic;
 3  using Microsoft.VisualStudio.TestTools.UnitTesting;
 4  using Business;
 5
 6
 7  namespace UnitTestProject1
 8  {
 9      [TestClass]
10      public class TrainTest
11      {
12
13          private Dictionary<string, Train> _trainsDictionary = new
              Dictionary<string, Train>();
14          Train example1 = new Train();
15          Train example2 = new Train();
16
17
18          [TestMethod]
19          public void DepartureTest()
20          {
21
22              _trainsDictionary.Add("1H34", example1);
23              _trainsDictionary.Add("1E32", example2);
24
25              string departure1 = "Edinburgh(Waverly)";
26              string departure2 = "London(Kings Cross)";
27
28              example1.Departure = departure1;
29              example2.Departure = departure2;
30
31              Assert.AreEqual(departure1, example1.Departure, "Departure Test");
32              Assert.AreEqual(departure2, example2.Departure, "Departure Test
                  2");
33
34          }
35
36          [TestMethod()]
37          public void DestinationTest()
38          {
39              _trainsDictionary.Add("1H34", example1);
40              _trainsDictionary.Add("1E32", example2);
41              string destination1 = "Edinburgh(Waverly)";
42              string destination2 = "London(Kings Cross)";
43              example1.Destination = destination1;
44              example2.Destination = destination2;
45
46              Assert.AreEqual(destination1, example1.Destination, "Destination
                  Test");
47              Assert.AreEqual(destination2, example2.Destination, "Destination
                  Test 2");
48          }
49
50          [TestMethod()]
51          public void DepartureDayTest()
52          {
```

```
53              _trainsDictionary.Add("1H34", example1);
54              _trainsDictionary.Add("1E32", example2);
55
56              example1.DateStart = "01/11/2018";
57              example2.DateStart = "20/12/2018";
58
59              Assert.AreEqual("01/11/2018", example1.DateStart, "DepartureDay
                  Test");
60              Assert.AreEqual("20/12/2018", example2.DateStart, "DepartureDay
                  Test 2");
61          }
62
63          [TestMethod()]
64          public void DepartureTimeTest()
65          {
66              _trainsDictionary.Add("1H34", example1);
67              _trainsDictionary.Add("1E32", example2);
68
69              string time1 = "11:00";
70              string time2 = "21:00";
71              example1.Time = time1;
72              example2.Time = time2;
73
74              Assert.AreEqual(time1, example1.Time, "DepartureTime Test");
75              Assert.AreEqual(time2, example2.Time, "DepartureTime Test 2");
76          }
77
78          [TestMethod()]
79          public void TypeTest()
80          {
81              _trainsDictionary.Add("1H34", example1);
82              _trainsDictionary.Add("1E32", example2);
83              string type1 = "Stopping";
84              string type2 = "Sleeper";
85              example1.Type = "Stopping";
86              example2.Type = "Sleeper";
87
88              Assert.AreEqual(type1, example1.Type, "Type Test");
89              Assert.AreEqual(type2, example2.Type, "Type Test 2");
90          }
91
92          [TestMethod()]
93          public void FirstClassTest()
94          {
95              _trainsDictionary.Add("1H34", example1);
96              _trainsDictionary.Add("1E32", example2);
97
98              example1.FirstClass = true;
99              example2.FirstClass = false;
100
101             Assert.IsTrue(example1.FirstClass, "FirstClass Test");
102             Assert.IsFalse(example2.FirstClass, "FirstClass Test 2");
103         }
104
105         [TestMethod()]
106         public void SleeperBerthTest()
```

```
107         {
108             _trainsDictionary.Add("1H34", example1);
109             _trainsDictionary.Add("1E32", example2);
110 
111             example1.SleeperBerth = false;
112             example2.SleeperBerth = true;
113 
114             Assert.IsFalse(example1.SleeperBerth, "SleeperBerth Test");
115             Assert.IsTrue(example2.SleeperBerth, "SleeperBerth Test 2");
116 
117         }
118 
119         //This test should fail...
120         [TestMethod()]
121         public void IntermediateTest()
122         {
123             _trainsDictionary.Add("1H34", example1);
124             _trainsDictionary.Add("1E32", example2);
125 
126 
127             List<string> _intermediate = new List<string>();
128             List<string> _intermediate1 = new List<string>();
129 
130 
131 
132             _intermediate.Add("Peterbourgh");
133             _intermediate = example1.Intermediate;
134             _intermediate1.Add("Darlington"+ "York");
135             _intermediate1 = example2.Intermediate;
136 
137             CollectionAssert.AreEqual(_intermediate, example1.Intermediate);
138             CollectionAssert.AreEqual(_intermediate1, example2.Intermediate);
139 
140         }
141 
142     }
143 }
144 
```

Coursework 2 Assessment- SET08119

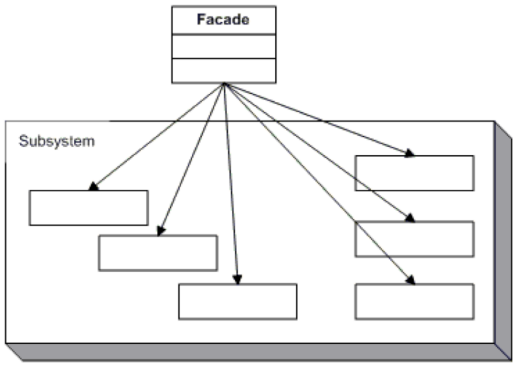
What advantages are of the 3-layered approach to building applications?

The architecture of 3 layer give us the ability to update the technology stack of one tier without impacting others areas of the application.
In fact we are able to scale the application up and out. For example in our coursework in the presentation layer we have a GUI, which contains all things that are visible (trains and booking) to the user such as screen layout. The business layer is the core of the system this is the link between other layers and contains runtime values like train ID in our example. And finally we have the data layer, which takes care of persistency, indeed in our coursework I implemented a binary formatter and serialized my classes in order to save every train in a bin file and load it when the project is started.
To conclude the 3 layered approach saves development manpower. It provides scalability, performance and availability.

With an example, explain why using design patterns can make the design of an OO system easier to understand.

A design pattern provides a general reusable solution to a common design problem. Design patterns are very useful as they solve recurring problems and in general simplify code. For example a Facade pattern allow us to simplify how to use an existing system. This design patterns make the design of an OO system easier. First of all, it enables us to use a complex system more easily, indeed we have a complicated system of which we need to use only a part so we use that design and we end up with a simpler, easier-to-use system. Secondly, it is easy to implement (eg. Define a new class that has the required interface). Finally, even though the facade simplifies the use of the required sub-system, the facade is not complete; certain functionality may be unavailable to the client.

Architecture



http://www.dofactory.com/net/facade-design-pattern