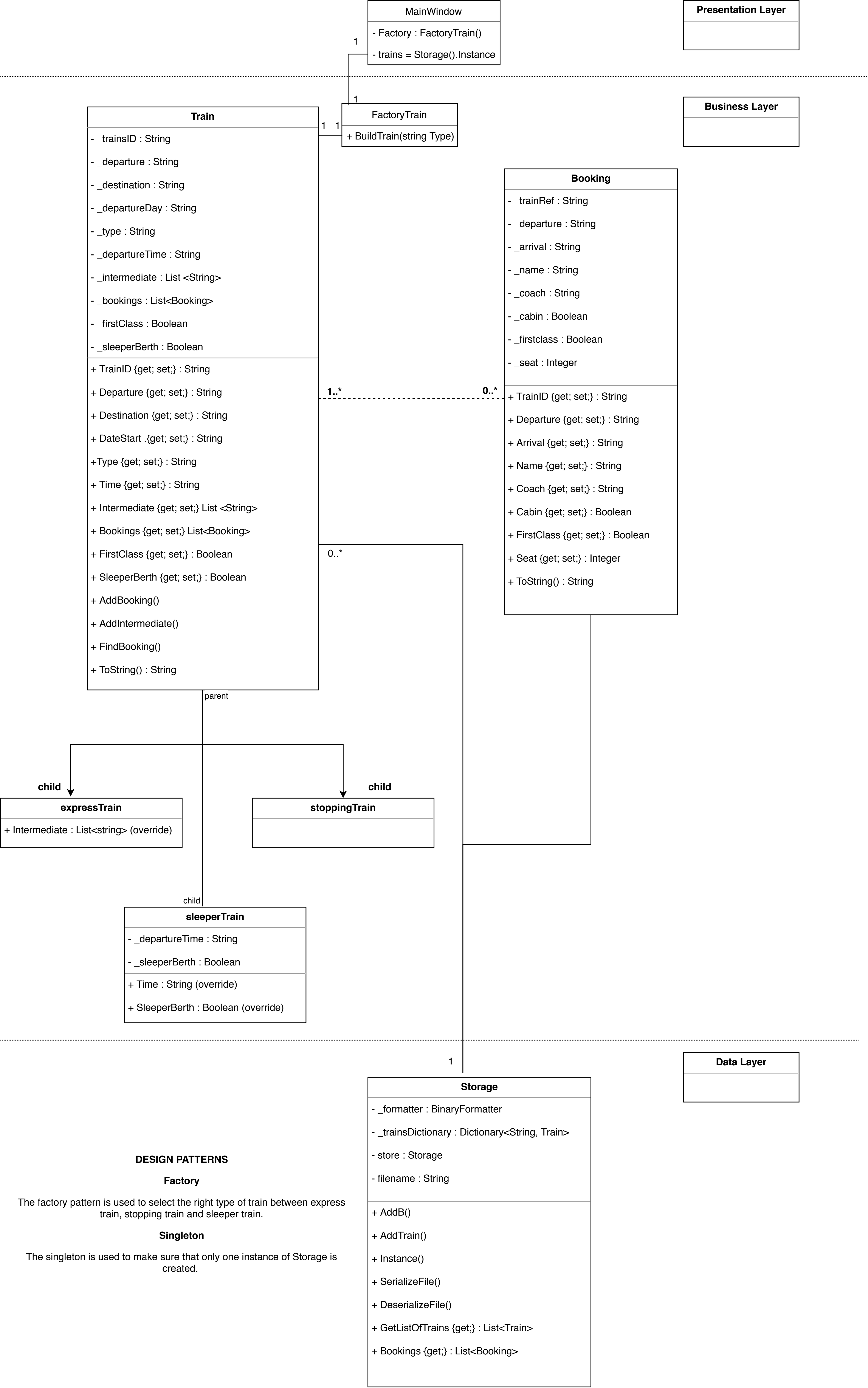

MainWindow
- Factory : FactoryTrain()
- trains = Storage().Instance
1
1
Presentation Layer
Business Layer
Train
- _trainsID : String
- _departure : String
- _destination : String
- _departureDay : String
- _type : String
- _departureTime : String
- _intermediate : List <String>
- _bookings : List<Booking>
- _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
1
1
FactoryTrain
+ BuildTrain(string Type)
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
child
expressTrain
+ Intermediate : List<string> (override)
stoppingTrain
child
sleeperTrain
- _departureTime : String
- _sleeperBerth : Boolean
+ Time : String (override)
+ SleeperBerth : Boolean (override)
Data Layer
1
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.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;
using System.IO;


namespace Business
{
    [Serializable]
    public class Train
    {
        /*
         * Author:                 40326941 Ismael Souf
         * Description:            This is the trains class for the booking which
         *                         defines the trains attributes
         * Date last modified:     06/12/2018
        */

        // Declare private variables
        private string _trainsID;
        private string _departure;
        private string _destination;
        private string _type;
        private string _departureTime;
        private string _departureDay;
        private bool _firstClass;
        private bool _sleeperBerth;
        private List<Booking> _bookings = new List<Booking>();
        private List<string> _intermediate = new List<string>();

        public Train()
        {

        }

        //Train ID must be a code.
        public string TrainID
        {

            get
            {
                return _trainsID;
            }
            set
            {
                if(value.Length != 4)
                {
                    throw new ArgumentException("Please enter a Train ID");
                }
                else
                {
                    _trainsID = value;
```

```csharp
                }
            }

        }


        //Departure of train
        public string Departure
        {
            get
            {
                return _departure;
            }
            set
            {
                if (value != null)
                {
                    _departure = value;
                }

                else
                {
                    throw new ArgumentException("Please verify the departure and destination.");
                }

            }
        }

        //Destination of train
        public string Destination
        {
            get
            {
                return _destination;
            }
            set
            {
                if (value != null)
                {
                    _destination = value;
                }

                else
                {
                    throw new ArgumentException("Please verify the departure and destination.");
                }
            }
        }


        //First class of train
        public bool FirstClass
        {
            get
            {
                return _firstClass;
            }
            set
            {
              _firstClass = value;
            }
        }

        //Departure date of train
        public string DateStart
        {
            get
            {
                return _departureDay;
            }
            set
            {

                if (value != null)
                {

                    _departureDay = value;
                }
                else
                {

                    throw new ArgumentException("Trains start date is not 
                    valid DD/MM/YYYY");
                }
            }
        }

        //Departure time of train
        public virtual string Time
        {
            get
            {
                return _departureTime;
            }
            set
            {

                if (value != null)
                {
                    _departureTime = value;
                }
                else
                {

                    throw new ArgumentException("Departure time is not 
                    valid");
                }
            }
        }
        //Type of train
        public string Type
        {
            get
            {
                return _type;
            }
            set
            {

                if (value != null)
                {
                    _type = value;
                }
                else
                {

                    throw new ArgumentException("Type of trains is not 
                    valid");
                }
            }
        }
        //Sleeper berth of train
        public virtual bool SleeperBerth
        {
            get
            {
                return _sleeperBerth;
            }
            set
            {
                _sleeperBerth = value;
            }
        }


        //Method to get intermediates of train
        public virtual string getIntermediate
        {
            get
            {
                var intermediateStop = String.Join(",", _intermediate);
                return intermediateStop;
            }

        }
        //List of string for intermediates
        public virtual List<string> Intermediate
        {
            get
            {
                {
                    List<string> res = new List<string>();
                    foreach (var v in _intermediate)
                    {
                        res.Add(v);
                    }
                    return res;
                }
            }

        }
        //Method to add intermediates in the List
        public virtual void AddIntermediate(string intermediates)
        {
            _intermediate.Add(intermediates);
        }

        //List of bookings for train
        public List<Booking> Bookings
        {
            get
            {
                return _bookings;
            }
            set
            {

                if (value == null)
                {

                    throw new ArgumentException("Passenger does not have any 
                   bookings");
                }
                else
                {
                    _bookings = value;
                }
            }
        }
        //Method bool to find if a booking is free
        public bool FindBooking(string coach, int seat)
        {

            foreach (Booking p in _bookings)
            {
                if (coach.Equals(p.Coach) && seat == p.Seat)
                {
                    return true;
                }
            }

            return false;
        }

        //Method to add a booking
        public void AddBooking(Booking booking)
        {
            _bookings.Add(booking);

        }

        public override string ToString()
        {
            return TrainID + " " + Departure + "-" + Destination + " " + 
              DateStart + " " + Time + " " + Type + " " + FirstClass + " " + 
              SleeperBerth + " " + getIntermediate;
        }


    }
}

```

```
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;
            }
        }
    }
}

```

```
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 a
           stopping Train
         * Date last modified:    06/12/2018
        */

    [Serializable]
    public class stoppingTrain : Train
    {

    }

}

```

```
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
           sleeperTrain and override methods
         * Date last modified:   06/12/2018
        */

    [Serializable]
    public class sleeperTrain : Train
    {
        private bool _sleeperBerth;
        private string _departureTime;

        public override bool SleeperBerth
        {
            get
            {
                return _sleeperBerth;
            }
            set
            {
                _sleeperBerth = value;
            }
        }

        public override string Time
        {
            get
            {
                return _departureTime;
            }
            set
            {
                string strTime_Start = value;
                DateTime dateTime_Start = Convert.ToDateTime(strTime_Start);
                if (dateTime_Start.Hour >= 21)
                {
                    _departureTime = value;
                }
                else
                {
                    throw new ArgumentException("Sleeper departs after 21:00");
                }
            }
        }


    }
}

```

```
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 the FactoryTrain class which build the ⮐
           type of train
         * Date last modified:    06/12/2018
        */
    [Serializable]
    public class FactoryTrain
    {
        public Train BuildTrain(string type)
        {
            if (type.Contains("Express"))
            {
                return new expressTrain();
            }
            else if (type.Contains("Stopping"))
            {
                return new stoppingTrain();
            }
            else if (type.Contains("Sleeper"))
            {
                return new sleeperTrain();
            }
            else
            {
                throw new ArgumentException("Type does not exist");
            }

        }
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using Business;

namespace Data
{
    public class Storage
    {
        /*
         * Author:                 40326941 Ismael Souf
         * Description:            This is my storage class which add trains to 
           dictionary.
         *                         This class also save and load to a bin file  
           using singleton and BinaryFormatter
         * Date last modified:    06/12/2018
        */


        private static Storage store;

        private Dictionary<string, Train> _trainDictionary;
        private BinaryFormatter _formatter;

        private const string filename = "trainsList.bin";

        private static readonly object padlock = new object();

        //Initializing instance of the class if there isn't one already
        public static Storage Instance()
        {
            if (store == null)
            {
                lock (padlock)
                {
                    if (store == null)
                    {
                        store = new Storage();
                    }
                }
            }

            return store;
        }

        //private constructor
        private Storage()
        {
            _trainDictionary = new Dictionary<string, Train>();
            _formatter = new BinaryFormatter();
        }

        //Method to add train into Dictionary
        public void AddTrain(string trainID, Train train)
        {
            // Check if train with that ID already exists in the trainDictionary
            if (_trainDictionary.ContainsKey(trainID))
            {
                throw new ArgumentException("Train with that ID already exists");


            }
            else
            {
                // Add train in the dictionary
                _trainDictionary.Add(trainID, train);
            }
        }

        //Method to add booking train
        public void AddBooking(string trainID, Booking booking)
        {
            // Check if train with that ID already exists in the trainDictionary
            if (_trainDictionary.ContainsKey(trainID))
            {
                //Check if the Seat and Coach are free in order to book, if yes add booking
                if (!_trainDictionary[trainID].FindBooking(booking.Coach, booking.Seat))
                {

                    _trainDictionary[trainID].AddBooking(booking);

                }
                else
                {
                    throw new ArgumentException("Booking with that ID already exists");

                }
            }
            else
            {
                throw new ArgumentException("Train with that ID not found");
            }
        }


        public void SerializeFile()
        {
            try
            {
                // Create a FileStream that will write data to file.
                FileStream writer = new FileStream(filename, FileMode.Create,
                    FileAccess.Write);
                // Save our dictionary of trains to file
                this._formatter.Serialize(writer, _trainDictionary);
                // Close the writer FileStream when we are done.
                writer.Close();
            }
            catch (Exception)
            {
                throw new ArgumentException("Unable to save our trains
                  informations.");
            }
        }

        public void DeserializeFile()
        {

            // Check if we had previously save informations of our trains
            if (File.Exists(filename))
            {

                try
                {
                    // Create a FileStream that will read access to the data
                    file.
                    FileStream reader = new FileStream(filename,
                    FileMode.Open, FileAccess.Read);
                    // Reconstruct from file.
                    _trainDictionary = (Dictionary<String, Train>)
                    _formatter.Deserialize(reader);
                    // Close the reader FileStream when we are done
                    reader.Close();

                }
                catch (Exception)
                {
                    throw new ArgumentException("Problem occurred with the
                    file.");
                }

            }

        }

        //Get our list of trains
        public List<Train> GetListOfTrains
        {
            get
            {
                // Create a new list of type train
                List<Train> trains = new List<Train>();

                if (_trainDictionary.Count > 0)
                {
                    // Loop through each trains in the dictionary
                    foreach (Train train in _trainDictionary.Values)
                    {
                        trains.Add(train);
                    }
                }
                return trains;

            }
        }

        //Get our list of bookings
        public List<Booking> Bookings
        {
            get
            {
                // Create a new list of type booking
                List<Booking> bookings = new List<Booking>();

                if (_trainDictionary.Count > 0)
                {
                    // Loop through each trains in the dictionary
                    foreach (Train c in _trainDictionary.Values)
                    {
                        // Loop through each booking in the current trains ↩
                       bookings list
                        foreach (Booking b in c.Bookings)
                        {
                            // Add the current booking to the list of bookings
                            bookings.Add(b);
                        }
                    }
                }
                return bookings;
            }

        }
    }
}


```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Xml.Serialization;
using Business;
using Data;


namespace Presentation
{

    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        FactoryTrain Factory = new FactoryTrain();
        Storage trains = Storage.Instance();

        public MainWindow()
        {

            InitializeComponent();
            trains.DeserializeFile();

            foreach (var s in trains.GetListOfTrains)
            {
                lbxTrain.Items.Add(s);
            }

            foreach (var u in trains.Bookings)
            {
                lbxBooking.Items.Add(u);
            }

            DateTime time = DateTime.Today;

            //Loop through time and populate to the appropriate combobox
            for (DateTime _time = time.AddHours(0); _time < time.AddHours(24); ↩
              _time = _time.AddMinutes(30))
            {
                timeBox.Items.Add(_time.ToShortTimeString());
            }

            //Hide visibility of intermediates until type is selected
            chkPeterborough.Visibility = Visibility.Hidden;
            chkDarlington.Visibility = Visibility.Hidden;
            chkYork.Visibility = Visibility.Hidden;
            chkNewcastle.Visibility = Visibility.Hidden;
            lblintermediate.Visibility = Visibility.Hidden;

            // Populate the seat combo box 1-60
            populateSeatList();

        }

        private void Window_Closed(object sender, CancelEventArgs e)
        {
            // Serialize the trains list including booking
            trains.SerializeFile();
        }



        private void AddTrains_Click(object sender, RoutedEventArgs e)
        {
            //Runs add train method
            AddTrain();
        }


        public void AddTrain()
        {


            try
            {
                //If the type of train is not selected
                if (typeBox.SelectedIndex == -1)
                {
                    throw new ArgumentException("Type undefined");
                }

                //Use type selected to get the right type of train in the
                  factory
                var train = Factory.BuildTrain(typeBox.SelectedItem.ToString
                  ());

                if (typeBox.SelectedItem != null)
                {
                    train.Type = typeBox.SelectionBoxItem.ToString();
                }

                if (txtTrainsID.Text.Length != 4)
                {
                    throw new ArgumentException("Invalid Trains ID.");
                }
                else
```

```csharp
            {
                train.TrainID = txtTrainsID.Text;
            }

            if (departureBox.SelectedIndex == 0)
            {
                train.Departure = "Edinburgh(Waverly)";
            }
            else if (departureBox.SelectedIndex == 1)
            {
                train.Departure = "London(Kings Cross)";
            }
            else
            {
                throw new ArgumentException("Invalid data for the departure station!");
            }

            if (destinationBox.SelectedIndex == 0)
            {
                train.Destination = "Edinburgh(Waverly)";
            }
            else if (destinationBox.SelectedIndex == 1)
            {
                train.Destination = "London(Kings Cross)";
            }
            else
            {
                throw new ArgumentException("Invalid data for the destination station!");
            }

            if (train.Departure == train.Destination)
            {
                throw new ArgumentException("Departure and Destination can not be the same!");
            }

            //If a date is selected in the departure picker, give the date selected
            if (departurePicker.SelectedDate != null)
            {
                train.DateStart = departurePicker.SelectedDate.Value.ToShortDateString();
            }
            else
            {
                throw new ArgumentException("Please select a departure day.");
            }

            //If first class is checked the train has first class
            if (chkFirstClass.IsChecked.Value)
            {
                train.FirstClass = true;
            }
```

```
                    else
                    {
                        train.FirstClass = false;
                    }

                    //If the type of train is a sleeper and sleeper berth is checked then the train has sleeperberth
                    if (chkSleeperBerth.IsChecked.Value && typeBox.SelectedIndex == 2)
                    {
                        train.SleeperBerth = true;
                    }
                    else
                    {
                        train.SleeperBerth = false;
                    }

                    //Convert string time to Datetime
                    if (timeBox.SelectedIndex >= 0)
                    {
                        string strTime_Start = timeBox.SelectedItem.ToString();
                        DateTime dateTime_Start = Convert.ToDateTime(strTime_Start);
                        train.Time = dateTime_Start.ToShortTimeString();

                    }
                    else
                    {
                        throw new ArgumentException("Please select a departure time");
                    }

                    //If station is checked, add it to the train.
                    if (chkPeterborough.IsChecked == true)
                    {
                        train.AddIntermediate("Peteborough");
                    }

                    if (chkDarlington.IsChecked == true)
                    {
                        train.AddIntermediate("Darlington");
                    }

                    if (chkYork.IsChecked == true)
                    {
                        train.AddIntermediate("York");
                    }

                    if (chkNewcastle.IsChecked == true)
                    {
                        train.AddIntermediate("Newcastle");
                    }

                    //Add trains
                    trains.AddTrain(train.TrainID, train);
                    //Display train properties
```

```
                lbxTrain.Items.Add(train);

                MessageBox.Show("Train added !");

            }
            catch (ArgumentException excep)
            {
                MessageBoxButton btnMessageBox = MessageBoxButton.OK;
                string caption = "Trains Error";
                MessageBox.Show(excep.Message, caption, btnMessageBox);
            }

        }



        public void populateSeatList()
        {
            int maxSeat = 60;
            // Loop 60 times
            for (int i = 1; i <= maxSeat; i++)
            {
                seatBox.Items.Add(i.ToString());
            }
        }

        private void CheckBox_Checked(object sender, RoutedEventArgs e)
        {

        }

        private void timeBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }

        private void ListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {


        }

        private void typeBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            //If the type of train is Stopping or Sleeper, show intermediates
            if (typeBox.SelectedIndex >= 1)
            {
                chkPeterborough.Visibility = Visibility.Visible;
                chkDarlington.Visibility = Visibility.Visible;
                chkYork.Visibility = Visibility.Visible;
                chkNewcastle.Visibility = Visibility.Visible;
                lblintermediate.Visibility = Visibility.Visible;

            }
            else
            {
                chkPeterborough.Visibility = Visibility.Hidden;
                chkDarlington.Visibility = Visibility.Hidden;
                chkYork.Visibility = Visibility.Hidden;
                chkNewcastle.Visibility = Visibility.Hidden;
                lblintermediate.Visibility = Visibility.Hidden;

            }

        }



        private void intermediateBox_SelectionChanged(object sender,
          SelectionChangedEventArgs e)
        {

        }
        //Add booking when button is pressed
        private void btnAddBooking_Click(object sender, RoutedEventArgs e)
        {

            Booking passenger = new Booking();
            Train train = new Train();
            int _interCost = 25;
            int _mainCost = 50;

            try
            {

               //If txtTrainBook is null or empty train ID is invalid
                if(String.IsNullOrEmpty(txtTrainBook.Text))
                {
                    throw new ArgumentException("Train ID is invalid");
                }
                else
                {
                    passenger.TrainID = txtTrainBook.Text;
                }

                //If txtName is null or blank name is invalid
                if (String.IsNullOrWhiteSpace(txtName.Text))
                {
                    throw new ArgumentException("Please enter a name for
                   booking");
                }
                else
                {
                    passenger.Name = txtName.Text;
                }

                if (coachBox.SelectedIndex == -1)
                {
                    throw new ArgumentException("Please select a Coach for booking");
                }
                else
                {
                    passenger.Coach = coachBox.SelectionBoxItem.ToString();
                }


                if (seatBox.SelectedIndex == -1)
                {
                    throw new ArgumentException("Please select a Seat for booking");
                }
                else
                {
                    passenger.Seat = Convert.ToInt32(seatBox.SelectionBoxItem.ToString());
                }

                if (departBox.SelectedIndex == -1 || arrivalBox.SelectedIndex == -1)
                {
                    throw new ArgumentException("Please select a Departure and Arrival station");
                }
                else if (departBox.SelectionBoxItem.ToString() == arrivalBox.SelectionBoxItem.ToString())
                {
                    throw new ArgumentException("Departure and Arrival must be different");
                }
                else
                {
                    passenger.Departure = departBox.SelectionBoxItem.ToString();
                    passenger.Arrival = arrivalBox.SelectionBoxItem.ToString();
                }

                if (train.SleeperBerth == true && chkCabin.IsChecked == true)
                {
                    passenger.Cabin = true;
                }
                else if (train.SleeperBerth == true && chkCabin.IsChecked == false)
                {
                    passenger.Cabin = false;
                }
                else if (train.SleeperBerth == false)
                {
                    passenger.Cabin = false;
                }
                else
                {
                    throw new ArgumentException("Train or coach does not offer sleeper berth");
                }

                if (train.FirstClass == true && chkFirstClass2.IsChecked == true)
                {
                    passenger.FirstClass = true;
                }
                else if (train.FirstClass == true && chkFirstClass2.IsChecked == false)
                {
                    passenger.FirstClass = false;
                }
                else if (train.FirstClass == false)
                {
                    passenger.FirstClass = false;
                }
                else
                {
                    throw new ArgumentException("Train or coach does not offer first class");
                }


                //Set booking fare for trains
                if (departBox.SelectionBoxItem.ToString() == "Edinburgh (Waverly)" && arrivalBox.SelectionBoxItem.ToString() == "London(Kings Cross)")
                {
                    if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked == true)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 90, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }

                    if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked == false)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 80, "Booking Fare", MessageBoxButton.YesNoCancel,
                    MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }
                    if (chkFirstClass2.IsChecked == true && chkCabin.IsChecked == false)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 60, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }

                    if(chkFirstClass2.IsChecked == false && chkCabin.IsChecked == false)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 50, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }
```

```


                }
                else if (departBox.SelectionBoxItem.ToString() == "London (Kings Cross)" && arrivalBox.SelectionBoxItem.ToString() == "Edinburgh(Waverly)")
                {
                    if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked == true)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 90, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }
                    if (chkFirstClass2.IsChecked == true)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 60, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }
                    else
                    {
                        switch (MessageBox.Show("Booking will cost £" + 50, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                MessageBox.Show("Booking Cancelled");
                                return;

                        }
                    }
                }


                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"))
                {
                    if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked == true)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 65, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }

                    if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked == false)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 55, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;
```

```csharp

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }

                    if (chkFirstClass2.IsChecked == true && chkCabin.IsChecked == false)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 35, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }

                    if(chkFirstClass2.IsChecked == false && chkCabin.IsChecked == false)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 25, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                MessageBox.Show("Booking Cancelled");
                                return;

                        }
                    }
                }

                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)"))
                {
                    if (chkCabin.IsChecked == true && chkFirstClass2.IsChecked
                    == true)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 65,
                  "Booking Fare", MessageBoxButton.YesNoCancel,
                  MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }

                    if (chkCabin.IsChecked == true)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 55,
                  "Booking Fare", MessageBoxButton.YesNoCancel,
                  MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }

                    if (chkFirstClass2.IsChecked == true)
                    {
                        switch (MessageBox.Show("Booking will cost £" + 35,
                  "Booking Fare", MessageBoxButton.YesNoCancel,
                    MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                return;

                        }
                    }
                    else
                    {

                        switch (MessageBox.Show("Booking will cost £" + 25, "Booking Fare", MessageBoxButton.YesNoCancel, MessageBoxImage.Question))
                        {
                            case MessageBoxResult.Yes:
                                MessageBox.Show("Booking purchased");
                                break;

                            case MessageBoxResult.No:
                                return;


                            case MessageBoxResult.Cancel:
                                MessageBox.Show("Booking Cancelled");
                                return;

                        }
                    }

                 }

               trains.AddBooking(passenger.TrainID, passenger);
               lbxBooking.Items.Add(passenger);

            }
            catch (Exception exception)
            {
                MessageBox.Show(exception.Message);
            }


        }


        private void listbox2_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {


        }

        private void FindTrain_Click(object sender, RoutedEventArgs e)
        {

            lbxDate.Items.Clear();

            //For each trains if a train has a selected date, display the trainID, the date and time of departure
            foreach (var res in trains.GetListOfTrains)
            {

                if (res.DateStart.Equals(departurePicker.Text))
                {
                    lbxDate.Items.Add(res.TrainID + " " + res.DateStart + " " + res.Time);
                }
            }


        }

        private void departureBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }

        private void timeBox_SelectionChanged_1(object sender, SelectionChangedEventArgs e)
        {

        }

        private void destinationBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }

        private void listbox3_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }

        private void departBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }

        private void arrivalBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }

        private void coachBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }

        private void seatBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }

        private void lbxDate_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }

    }
}

```

```csharp

using System.Collections.Generic;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Business;


namespace UnitTestProject1
{
    [TestClass]
    public class TrainTest
    {

        private Dictionary<string, Train> _trainsDictionary = new Dictionary<string, Train>();
        Train example1 = new Train();
        Train example2 = new Train();


        [TestMethod]
        public void DepartureTest()
        {

            _trainsDictionary.Add("1H34", example1);
            _trainsDictionary.Add("1E32", example2);

            string departure1 = "Edinburgh(Waverly)";
            string departure2 = "London(Kings Cross)";

            example1.Departure = departure1;
            example2.Departure = departure2;

            Assert.AreEqual(departure1, example1.Departure, "Departure Test");
            Assert.AreEqual(departure2, example2.Departure, "Departure Test 2");

        }

        [TestMethod()]
        public void DestinationTest()
        {
            _trainsDictionary.Add("1H34", example1);
            _trainsDictionary.Add("1E32", example2);
            string destination1 = "Edinburgh(Waverly)";
            string destination2 = "London(Kings Cross)";
            example1.Destination = destination1;
            example2.Destination = destination2;

            Assert.AreEqual(destination1, example1.Destination, "Destination Test");
            Assert.AreEqual(destination2, example2.Destination, "Destination Test 2");
        }

        [TestMethod()]
        public void DepartureDayTest()
        {
            _trainsDictionary.Add("1H34", example1);
            _trainsDictionary.Add("1E32", example2);

            example1.DateStart = "01/11/2018";
            example2.DateStart = "20/12/2018";

            Assert.AreEqual("01/11/2018", example1.DateStart, "DepartureDay 
              Test");
            Assert.AreEqual("20/12/2018", example2.DateStart, "DepartureDay 
              Test 2");
        }

        [TestMethod()]
        public void DepartureTimeTest()
        {
            _trainsDictionary.Add("1H34", example1);
            _trainsDictionary.Add("1E32", example2);

            string time1 = "11:00";
            string time2 = "21:00";
            example1.Time = time1;
            example2.Time = time2;

            Assert.AreEqual(time1, example1.Time, "DepartureTime Test");
            Assert.AreEqual(time2, example2.Time, "DepartureTime Test 2");
        }

        [TestMethod()]
        public void TypeTest()
        {
            _trainsDictionary.Add("1H34", example1);
            _trainsDictionary.Add("1E32", example2);
            string type1 = "Stopping";
            string type2 = "Sleeper";
            example1.Type = "Stopping";
            example2.Type = "Sleeper";

            Assert.AreEqual(type1, example1.Type, "Type Test");
            Assert.AreEqual(type2, example2.Type, "Type Test 2");
        }

        [TestMethod()]
        public void FirstClassTest()
        {
            _trainsDictionary.Add("1H34", example1);
            _trainsDictionary.Add("1E32", example2);

            example1.FirstClass = true;
            example2.FirstClass = false;

            Assert.IsTrue(example1.FirstClass, "FirstClass Test");
            Assert.IsFalse(example2.FirstClass, "FirstClass Test 2");
        }

        [TestMethod()]
        public void SleeperBerthTest()
        {
            _trainsDictionary.Add("1H34", example1);
            _trainsDictionary.Add("1E32", example2);

            example1.SleeperBerth = false;
            example2.SleeperBerth = true;

            Assert.IsFalse(example1.SleeperBerth, "SleeperBerth Test");
            Assert.IsTrue(example2.SleeperBerth, "SleeperBerth Test 2");

        }

        //This test should fail...
        [TestMethod()]
        public void IntermediateTest()
        {
            _trainsDictionary.Add("1H34", example1);
            _trainsDictionary.Add("1E32", example2);


            List<string> _intermediate = new List<string>();
            List<string> _intermediate1 = new List<string>();



            _intermediate.Add("Peterbourgh");
            _intermediate = example1.Intermediate;
            _intermediate1.Add("Darlington"+ "York");
            _intermediate1 = example2.Intermediate;

            CollectionAssert.AreEqual(_intermediate, example1.Intermediate);
            CollectionAssert.AreEqual(_intermediate1, example2.Intermediate);

        }

    }
}
```

Coursework 2 Assessment- SET08119

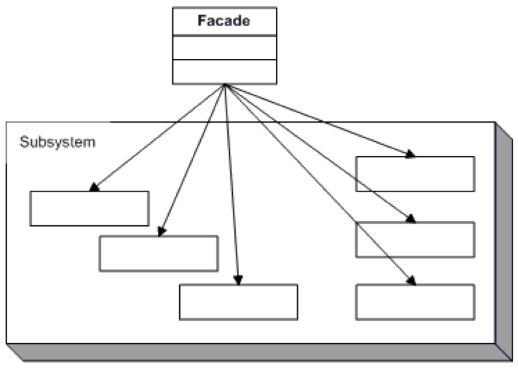
What advantages are of the 3-layered approach to building applications?

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