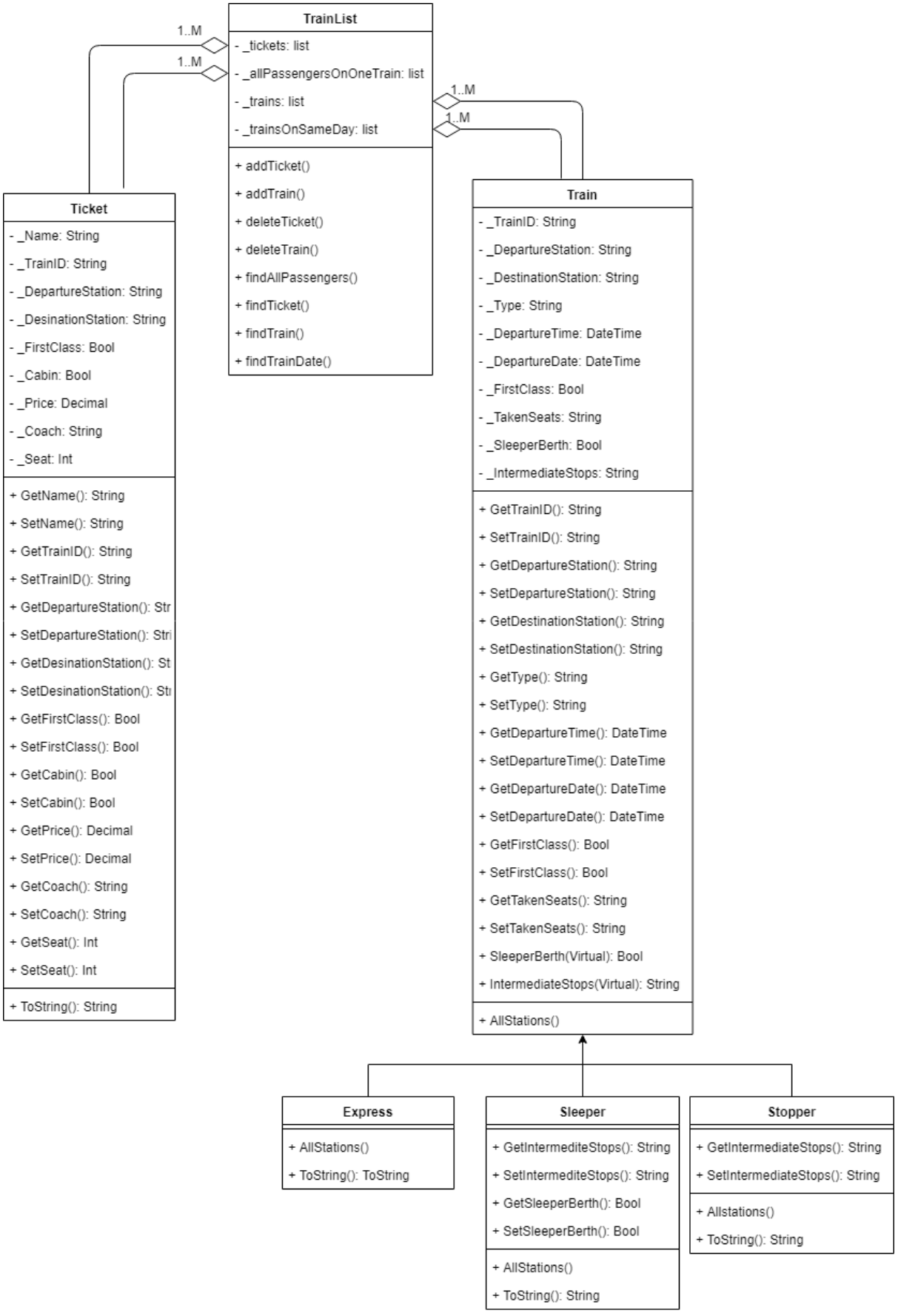
Class Diagram



Used Design Patterns

Singleton: my trainlist class shows this design pattern as it is only instantiated once in the code.

Decorator: my express, stopper and sleeper classes show this design pattern as they all add functionality to the train class they all inherit from

Listing of all Non-GUI classes

Trainlist.cs

```
public class TrainList
    {
        //creates the private lists that will be used throughout the methods
        //creates a list to store all trains
        private List<Train> _trains = new List<Train>();
        //creates a list to store all the trains that depart on the same day
        private List<Train> _trainsOnSameDay = new List<Train>();
        //creates a list to store all tickets
        private List<Ticket> _tickets = new List<Ticket>();
        //creates a list to store all tickets on one train
        private List<Ticket> _allPassengersOnOneTrain = new List<Ticket>();

        //method to add a train to the train list
        public void addTrain(Train newTrain)
        {
            //adds the train to the list
            _trains.Add(newTrain);
        }

        //method to find a single train by its ID
        public Train findTrain(string searchTrainID)
        {
            //looks through each train in the list of all trains and returns the one
with the id that matches the id to be searched
            foreach (Train t in _trains)
            {
                if (searchTrainID == t.TrainID)
                {
                    return t;
                }
            }

            return null;
        }

        //method to find all the trains that depart on the same date
        public List<Train> findTrainDate(DateTime searchTrainDate)
        {
            //looks through each train in the list of all trains and if the trains
date matches the date being searched the train is added to a second list
            foreach (Train t in _trains)
            {
                if (searchTrainDate == t.DepartureDate)
                {
                    _trainsOnSameDay.Add(t);
                }
            }
            //returns the list
            return _trainsOnSameDay;
```

```
        }

        //method to delete a train from the list
        public void deleteTrain(string deleteTrainId)
        {
            //searches the train to be deleted and if the search doesnt return null
removes the train from the list
            Train t = this.findTrain(deleteTrainId);
            if (t != null)
            {
                _trains.Remove(t);
            }

        }

        //method to add a ticket to the list
        public void addTicket(Ticket newTicket)
        {
            //adds the ticket to the list of all tickets
            _tickets.Add(newTicket);
        }

        //method to find a ticket by its name
        public Ticket findTicket(string searchName)
        {
            //looks through each ticket and if the name to be searched is the same as
the name on the ticket the ticket is returned
            foreach (Ticket t in _tickets)
            {
                if (searchName == t.Name)
                {
                    return t;
                }
            }

            return null;
        }

        //method to delete a ticket from the list
        public void deleteTicket(string deleteName)
        {
            //finds the ticket by the name
            Ticket t = this.findTicket(deleteName);
            //if the ticket returned doesnt equal null, the ticket is removed from the
list
            if (t != null)
            {
                _tickets.Remove(t);
            }
        }

        //method to find all passengers on one train
        public List<Ticket> findAllPassengers(string SearchTrainID)
        {
            //looks through each ticket in the list of all tickets and if the train id
of that ticket is the same as the train id to be searched it adds the ticket to
another list
            foreach (Ticket t in _tickets)
            {
                if (SearchTrainID == t.TrainID)
                {
                    _allPassengersOnOneTrain.Add(t);
```

```
            }
        }
        //returns the list
        return _allPassengersOnOneTrain;
    }

}
```

Ticket.cs

```
public class Ticket
  {
      //creates the private attributes of the class
      private string _Name;
      private string _TrainID;
      private string _DepartureStation;
      private string _DestinationStation;
      private bool _FirstClass;
      private bool _Cabin;
      private decimal _Price;
      private string _Coach;
      private int _Seat;

      //get and set methods for the tickets name
      public string Name
      {
          get
          {
              //returns the private variable name
              return _Name;
          }

          set
          {
              //checks if the string is empty
              if (value != string.Empty)
              {
                  //sets the name if the string isnt blank
                  _Name = value;
              }
              else
              {
                  //throws argument if the string is empty
                  throw new ArgumentException("Passenger name cannot be empty");
              }
          }
      }
      //contains the get and set methods for the train id the ticket is assigned to
      //the validation for this is done in the MainWindow.xaml.cs
      public string TrainID
      {
          get
          {
              //returns the train id
              return _TrainID;
          }

          set
          {
              //sets the train id
              _TrainID = value;
          }
```

```
        }
        //contains the get/set methods for the departure station of the ticket
        public string DepartureStation
        {
            get
            {
                //returns the Departure Station
                return _DepartureStation;
            }

            set
            {
                //validation to make sure the departure station is only one of the
possible stations
                if (value.Equals("Edinburgh Waverley"))
                {
                    _DepartureStation = value;
                }
                else if (value.Equals("London Kings Cross"))
                {
                    _DepartureStation = value;
                }
                else if (value.Equals("Peterborough"))
                {
                    _DepartureStation = value;
                }
                else if (value.Equals("Darlington"))
                {
                    _DepartureStation = value;
                }
                else if (value.Equals("York"))
                {
                    _DepartureStation = value;
                }
                else if (value.Equals("Newcastle"))
                {
                    _DepartureStation = value;
                }
                else
                {
                    //throws exception if the value isnt one of the possible stations
                    throw new ArgumentException("Please enter a vaid station from the
list; Edinburgh Waverley, London Kings Cross, Peterborough, Darlington, York or
Newcastle");
                }
            }
        }
        //methods to get/set the desination station of the ticket
        public string DestinationStation
        {
            get
            {
                //returns the desination station
                return _DestinationStation;
            }
            //validation to allow you to only set the desination station to one of the
possible ones
            set
            {
                if (value.Equals("Edinburgh Waverley"))
                {
                    _DestinationStation = value;
```

```
                }
                else if (value.Equals("London Kings Cross"))
                {
                    _DestinationStation = value;
                }
                else if (value.Equals("Peterborough"))
                {
                    _DestinationStation = value;
                }
                else if (value.Equals("Darlington"))
                {
                    _DestinationStation = value;
                }
                else if (value.Equals("York"))
                {
                    _DestinationStation = value;
                }
                else if (value.Equals("Newcastle"))
                {
                    _DestinationStation = value;
                }
                else
                {
                    //throws exception if the value isnt one of the possible stations
                    throw new ArgumentException("Please enter a vaid station from the
list; Edinburgh Waverley, London Kings Cross, Peterborough, Darlington, York or
Newcastle");
                }
            }
        }
        //gets/setas weather the ticket is first class or not
        public bool FirstClass
        {
            get
            {
                //returns the first class attribute
                return _FirstClass;
            }

            set
            {
                //sets the first class variable
                _FirstClass = value;
            }
        }
        //gets/sets the cabin for the ticket
        public bool Cabin
        {
            get
            {
                //returns the cabin
                return _Cabin;
            }

            set
            {
                //sets the cabin
                _Cabin = value;
            }
        }
        //gets/sets the tickets price
        public decimal Price
```

```
        {
            get
            {
                //returns the price
                return _Price;
            }

            set
            {
                //validation to make sure the price is more than 0
                if (value == 0)
                {
                    //throws error if the price is 0
                    throw new ArgumentException("Have you remebered to calculate the
price?");
                }
                else
                {
                    //sets the price
                    _Price = value;
                }
            }
        }
        //gets/sets what coach the ticket is for
        public string Coach
        {
            get
            {
                //returns the coach for the ticket
                return _Coach;
            }

            set
            {
                //validation to make sure the coach is a valid letter
                if (value.Any(char.IsLetter))
                {
                    _Coach = value;
                }
                else
                {
                    //throws argument if the coach isnt a valid letter
                    throw new ArgumentException("Please enter a valid letter");
                }
            }
        }
        //gets/sets the seat of the ticket
        public int Seat
        {
            get
            {
                //returns the seat number
                return _Seat;
            }

            set
            {
                //sets the seat number
                _Seat = value;
            }
        }
```

```
        //overrides ToString() to display all the attributes
        public override string ToString()
        {
            return "Name: " + _Name + ", " + "Train ID: " + _TrainID + ", " +
"Departure Station: " + _DepartureStation + ", " + "Destination Station: " +
_DestinationStation + ", " + " First Class: " + _FirstClass + ", " + "Cabin: " +
_Cabin + ", " + "Price: £" + _Price + ", " + "Seat: " +  _Coach + _Seat +
System.Environment.NewLine + System.Environment.NewLine;
        }

    }
```

Train.cs

```
public class Train
    {
        //creates the private attributes of the class
        private string _TrainID;
        private string _DepartureStation;
        private string _DestinationStation;
        private string _Type;
        private DateTime _DepartureTime;
        private DateTime _DepartureDate;
        private bool _FirstClass;
        private string _TakenSeats;
        private bool _SleeperBerth;
        private string _IntermediateStops;

        //get/set methods for the trains id
        public string TrainID
        {
            get
            {
                //returns the train id
                return _TrainID;
            }

            set
            {
                //validation to make sure the trains id has to be 4 cahracters
                if (value.Length == 4)
                {
                    //sets the train id if its 4 characters long
                    _TrainID = value;
                }
                else
                {
                    //throws argument if it isnt exactly 4 characters
                    throw new ArgumentException("The train ID must be 4 characters");
                }
            }
        }

        //get/set methods for the trains departure station
        public string DepartureStation
        {
            get
            {
                //returns the departure station
                return _DepartureStation;
            }
```

```
            set
            {
                //validation to make sure the trains departure station can only be one
of the valid stations
                if (value.Equals("Edinburgh Waverley"))
                {
                    _DepartureStation = value;
                }
                else if (value.Equals("London Kings Cross"))
                {
                    _DepartureStation = value;
                }
                else
                {
                    //throws an exception if the departure station to be set isnt one
of the valid options
                    throw new ArgumentException("Departure station can only be either
'Edinburgh Waverley' or 'London Kings Cross'");
                }
            }
        }

        //get/set methods for the trains desination station
        public string DestinationStation
        {
            get
            {
                //returns the trains desination station
                return _DestinationStation;
            }

            set
            {
                //validation to make sure that the desination station can only be one
of the valid stations
                if (value.Equals("Edinburgh Waverley"))
                {
                    _DestinationStation = value;
                }
                else if (value.Equals("London Kings Cross"))
                {
                    _DestinationStation = value;
                }
                else
                {
                    //throws excetion if the desination station isnt valid
                    throw new ArgumentException("Destination station can only be
either 'Edinburgh Waverley' or 'London Kings Cross'");
                }
            }
        }

        //get/set methods for the trains type
        public string Type
        {
            get
            {
                //returns the type
                return _Type;
            }

            set
```

```
    {
        //sets the type
        _Type = value;
    }
}

//get/set methods for the trains departure time
public DateTime DepartureTime
{
    get
    {
        //returns the time
        return _DepartureTime;
    }

    set
    {
        //sets the time
        _DepartureTime = value;
    }
}

//get/set methods for the trains departure date
public DateTime DepartureDate
{
    get
    {
        //returns the departure date
        return _DepartureDate;
    }

    set
    {
        //sets the departure date
        _DepartureDate = value;
    }
}

//get/set methods for weather the train allows a first class carrage
public bool FirstClass
{
    get
    {
        //returns the first class attribute
        return _FirstClass;
    }

    set
    {
        //sets the first class attribute
        _FirstClass = value;
    }
}

//get/set methods for all of the seats booked on the train
public string TakenSeats
{
    get
    {
        //returns all of the seats booked on the train
        return _TakenSeats;
    }
```

```
            set
            {
                //sets the seats booked on the train
                _TakenSeats = value;
            }
        }

        //virtural method for getting/setting the sleeper berth attribute, to be
overridden by the child classes
        public virtual bool SleeperBerth
        {
            get
            {
                //returns the sleeper berth
                return _SleeperBerth;
            }
            set
            {
                //sets the sleeper berth
                _SleeperBerth = value;
            }
        }

        //virtual method for getting/setting the trains intermediate stops, to be
overridden by the child classes
        public virtual string IntermediateStops
        {
            get
            {
                //returns the intermediate stops
                return _IntermediateStops;
            }

            set
            {
                //sets the intermediate stops
                _IntermediateStops = value;
            }
        }

        //virtual string for displaying all stations, to be overridden by the child
classes
        public virtual string AllStations()
        {
            return null;
        }
    }
```

Stopper.cs

```
public class Stopper : Train
    {
        //overrides the intermediate stops string
        public override string IntermediateStops
        {
            get => base.IntermediateStops;
            set => base.IntermediateStops = value;
        }
        //overrides the all stations string to show the departure, intermediate and
desination stations
        public override string AllStations()
        {
```

```
            return DepartureStation + ", " + IntermediateStops + DestinationStation;
        }
        //overrides ToString to show all relevant attributes
        public override string ToString()
        {
            return "Train ID: " + TrainID + ", " + "Type: " + Type + ", " + "Departure
Station: " + DepartureStation + ", " + "Intermediate Stations: " + IntermediateStops +
"Destination Station:" + DestinationStation + ", " + "Departure Time:" +
DepartureTime.ToString("HH:mm") + ", " + "Departure Date: " +
DepartureDate.ToString("dd/MM/yyyy") + ", " + "First Class: " + FirstClass + ", " +
"Taken Seats: " + TakenSeats + System.Environment.NewLine +
System.Environment.NewLine;
        }
    }
```

Sleeper.cs

```
public class Sleeper : Train
    {
        //overrides the sleeper berth bool
        public override bool SleeperBerth
        {
            get => base.SleeperBerth;
            set => base.SleeperBerth = value;
        }
        //overrides the intermediate stops string
        public override string IntermediateStops
        {
            get => base.IntermediateStops;
            set => base.IntermediateStops = value;
        }
        //overrides AllStations to show the desination, departure and intermediate
stops
        public override string AllStations()
        {
            return DepartureStation + ", " + IntermediateStops + DestinationStation;
        }
        //overrides to string to display the relevent attrubutes
        public override string ToString()
        {
            return "Train ID: " + TrainID + ", " + "Type: " + Type + ", " + "Departure
Station: " + DepartureStation + ", " + "Intermediate Stations: " + IntermediateStops +
"Destination Station:" + DestinationStation + ", " + "Departure Time:" +
DepartureTime.ToString("HH:mm") + ", " + "Departure Date: " +
DepartureDate.ToString("dd/MM/yyyy") + ", " + "First Class: " + FirstClass + ", " +
"Sleeper Bearth: " + SleeperBerth + "Taken Seats: " + TakenSeats +
System.Environment.NewLine + System.Environment.NewLine;
        }
    }
```

Express.cs

```
public class Express : Train
    {
        //overrides the method for AllStations, only displays the departure and
desination stations, as they are the only ones that are valid for express trains
        public override string AllStations()
        {
            return DepartureStation + ", " + DestinationStation;
        }
        //overrides tostring to show all the appropriate attrubutes for the express
class
        public override string ToString()
```

```
        {
            return "Train ID: " + TrainID + ", " + "Type: " + Type + ", " + "Departure
Station: " + DepartureStation + ", " + "Destination Station:" + DestinationStation +
", " + "Departure Time:" +  DepartureTime.ToString("HH:mm") + ", " + "Departure Date:
" + DepartureDate.ToString("dd/MM/yyyy") + ", " +"First Class: "+ FirstClass + ", "
+"Taken Seats: " + TakenSeats + System.Environment.NewLine +
System.Environment.NewLine;
        }
    }
```

a. What advantages are of the 3 layered approach to building applications.

The three tiered approach to building applications is relatively simple. There are three layers, somewhat like the layers of a cake, that each take a different job in the application. The first layer is responsible for the U.I. of the application and is known as the presentation layer (IBM. 2018). The second tier is known as the business or object layer, this is needed to manage the objects or business logic of the application and can access the third tier. The presentation layer only has access to the methods and attributes declared as public in this layer. Finally, the third layer is the data layer. This layer accesses the data stores by the application, for example a database. It allows the other layers to be configured without directly having access to the database libraries (Microsoft. 2012). The main advantage of using the three layered architecture is maintainability, this means that changes in one layer have no effect on other layers. This means that if the presentation layer has a problem it can be fixed without having to re-code any of the other layers. A second advantage is reliability. For example, if the database supporting the application is split over multiple servers, multiple levels of redundancy can be added to the data layer in case one of the servers fails, allowing the application to keep running with minimal interruption. This backs up the first point as if that were to happen only the data layer would need to be dealt with as none of the other layers can access the database (Tony Marston. 2012). A final advantage is that a three tier architecture makes collaboration very simple as each layer can only access other layers by the methods and properties defined in that layer. This means that they can be worked on by different people entirely as they don't need to know the inner workings of each layer. Overall the main advantage to using this architecture is its reliability, which stems from the fact that all the layers are isolated from each other so if there is a problem with one it doesn't affect any other layers and can be fixed without having to look at other layers.

References

IBM. (2018). Three-tier architectures. Retrieved From: https://www.ibm.com/support/knowledgecenter/en/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/covr_3-tier.html (Accessed on 07/12/2018)

Microsoft. (2018). Using a Three-Tier Architecture Model. Retrieved From: https://docs.microsoft.com/en-us/windows/desktop/cossdk/using-a-three-tier-architecture-model (Accessed on 07/12/2018)

Tony Marston. (2012). What is the 3-Tier Architecture?. Tony Marston's Blog. Retrieved From:

https://www.tonymarston.net/php-mysql/3-tier-architecture.html (Accessed on 07/12/2018)

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

A design pattern is a framework to help programmers design how an applications classes interact with each other. They can be categorized in three groups, creational, structural and behavioral (Dofactory. 2018). One example of a design pattern is singleton, which is an example of a creational pattern. In order to adhere to this pattern a class can only have one instance and have a global point of access to it (Source Making. 2012). Singleton makes an object oriented system easy to understand by only instantiating the class once. This means that code can be followed easily as all the references to the class will be referencing the same object. This also means that the object can only be accessed by its pre-defined methods meaning that it is simple to use. This helps the system overall to be easily understood by reducing the number of objects created allowing a programmer to follow each one through the code without getting lost or being confused as to which object is being dealt at a time. In the example the class is instantiated twice (Figure 2), by using the pre-defined method to instantiate the class (Figure 1). However, both classes are identical, showing that the class follows the pattern as only one version of the class can be created. Overall singleton makes an object oriented system easy to understand by making sure that the class can only be instantiated once meaning that all references to that object can be followed easily.

Diagram

```
/// <summary>
/// The 'Singleton' class
/// </summary>
class Singleton
{
  private static Singleton _instance;

  // Constructor is 'protected'
  protected Singleton()
  {
  }

  public static Singleton Instance()
  {
    // Uses lazy initialization.
    // Note: this is not thread safe.
    if (_instance == null)
    {
      _instance = new Singleton();
    }

    return _instance;
  }
 }
}
```

Figure 1, an example singleton class (Dofactory. 2018)

```
class MainApp
{
  /// <summary>
  /// Entry point into console application.
  /// </summary>
  static void Main()
  {
    // Constructor is protected -- cannot use new
    Singleton s1 = Singleton.Instance();
    Singleton s2 = Singleton.Instance();

    // Test for same instance
    if (s1 == s2)
    {
      Console.WriteLine("Objects are the same instance");
    }

    // Wait for user
    Console.ReadKey();
  }
}
```

Figure 2, the singleton class being put to use in a main method. The output will say ‘Objects are the same instance’. (Dofactory. 2018)

References

Dofactory. (2018). .Net Design Patterns. Retrieved From:

https://www.dofactory.com/net/design-patterns (Accessed on 07/12/2018).

Source Making. (2018). Singleton Design Pattern. Retrieved From:

https://sourcemaking.com/design_patterns/singleton (Accessed on 07/12/2018).

DoFactory. (2018). Singleton. (Used for the example). Retrieved From:

https://www.dofactory.com/net/singleton-design-pattern (Accessed on 07/12/2018).