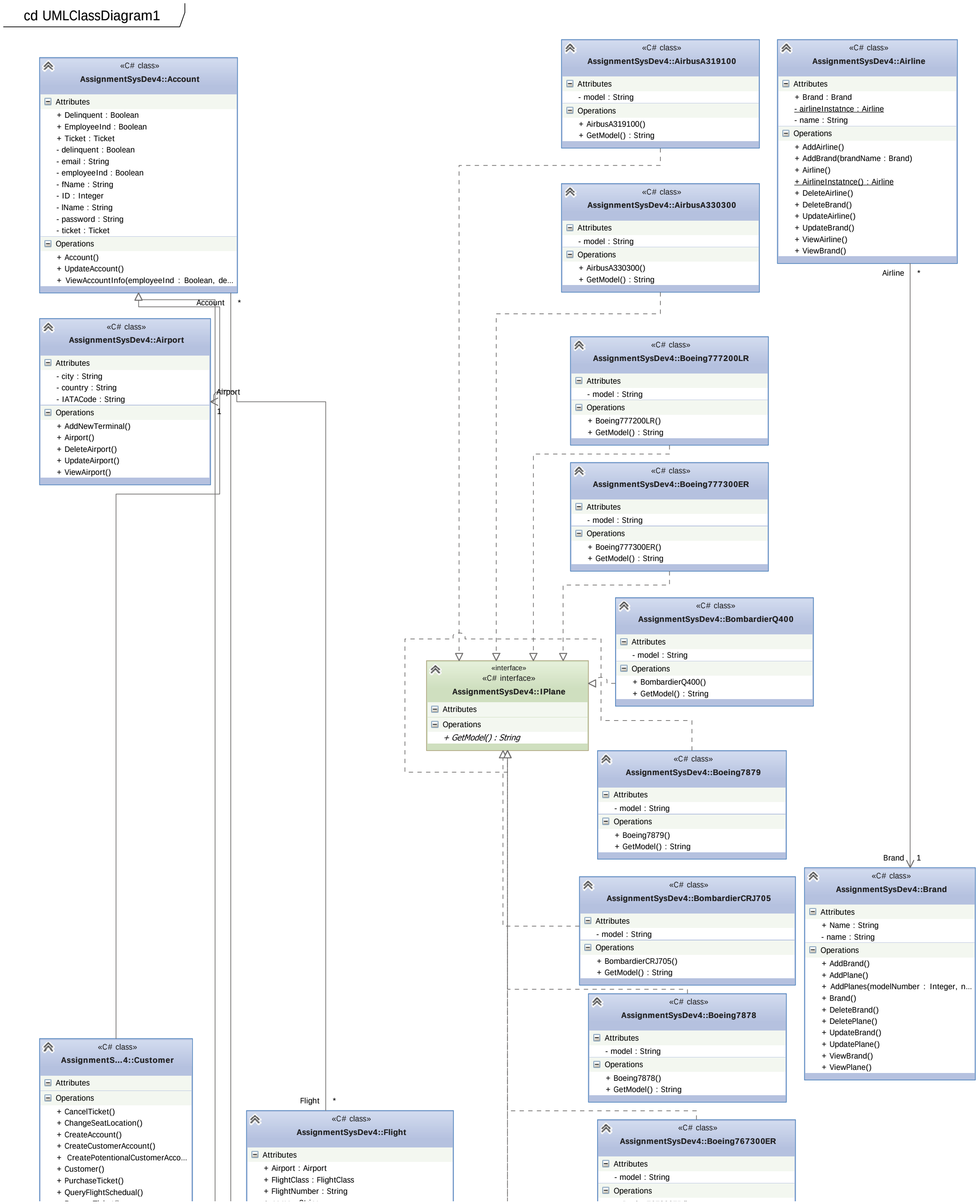
INFO8240 Assignment # 4

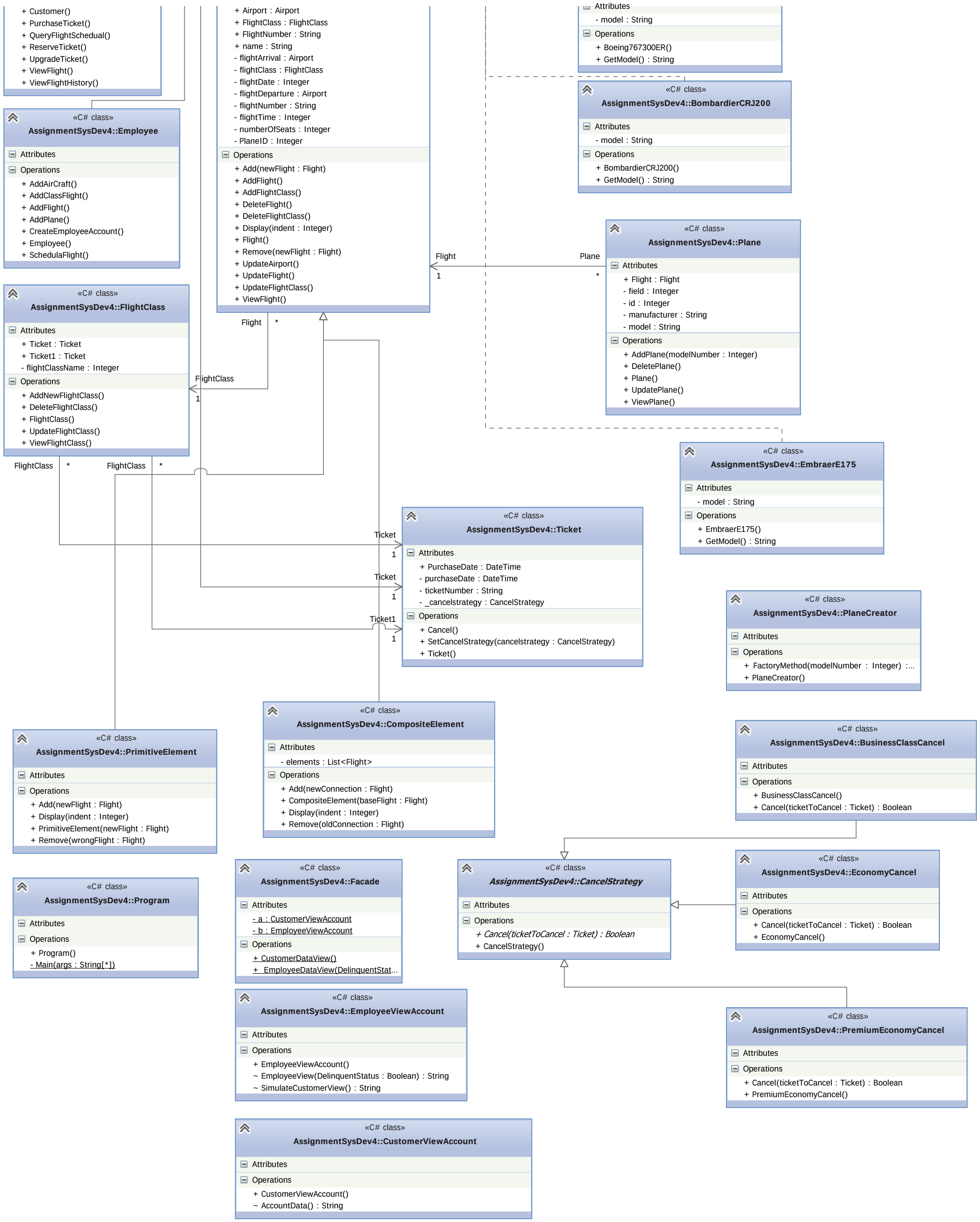
Changho Choi

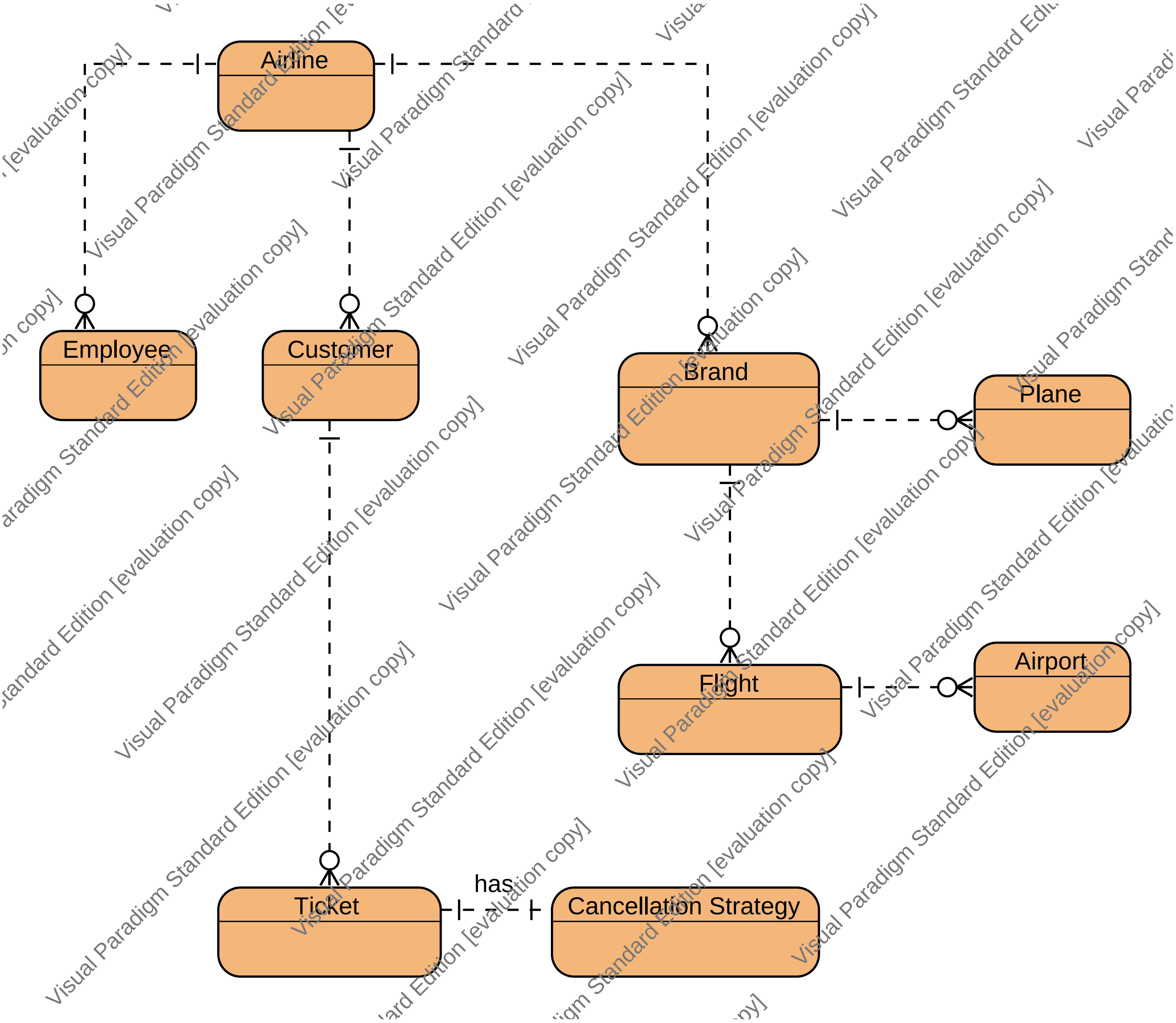
Hassan Nahhal

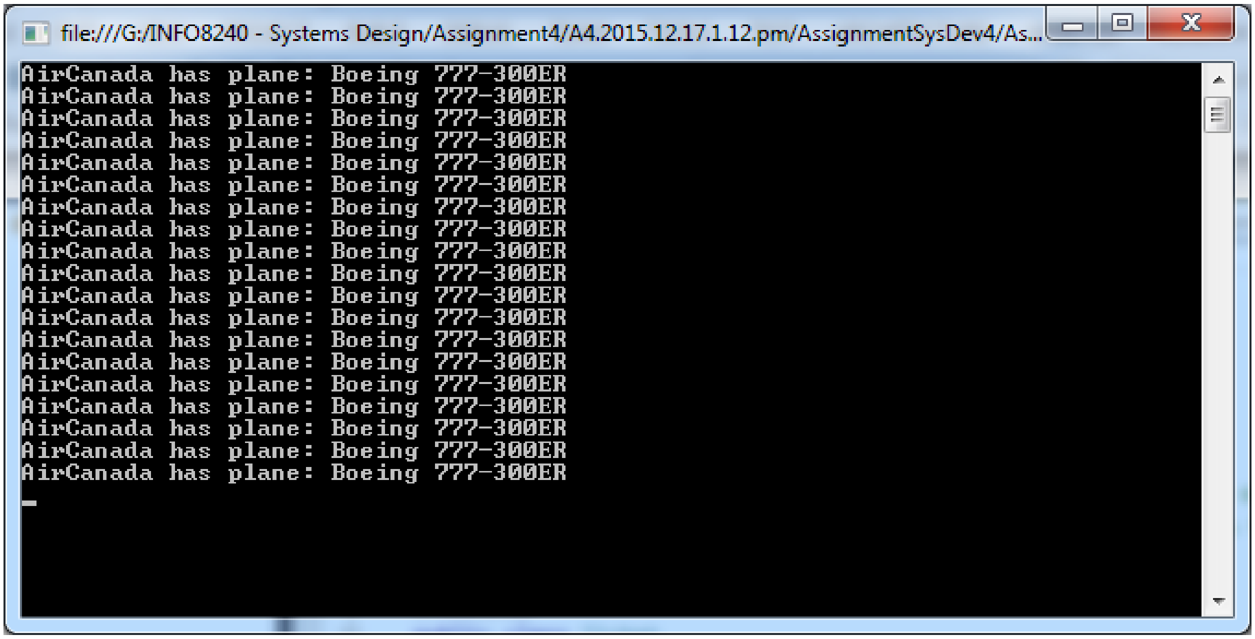
Sung Joe Kim

Nicholas Collins

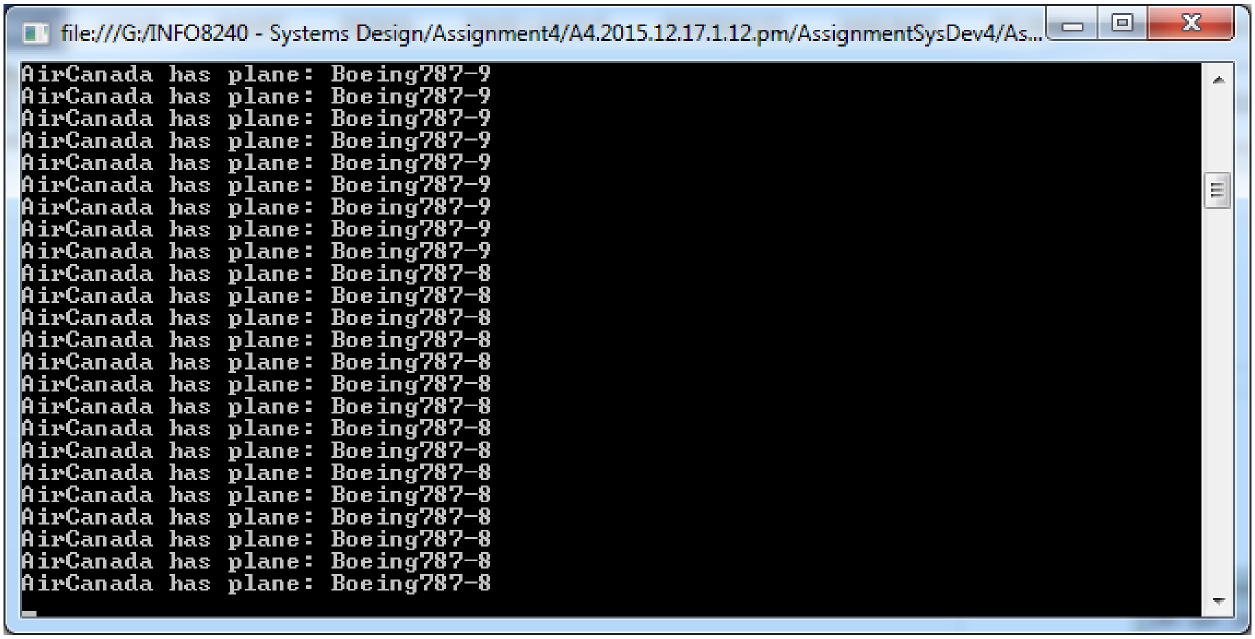

cd UMLClassDiagram1
«C# class»
AssignmentSysDev4::Account
Attributes
+ Delinquent : Boolean
+ EmployeeInd : Boolean
+ Ticket : Ticket
- delinquent : Boolean
- email : String
- employeeInd : Boolean
- fName : String
- ID : Integer
- lName : String
- password : String
- ticket : Ticket
Operations
+ Account()
+ UpdateAccount()
+ ViewAccountInfo(employeeInd : Boolean, de...
Account
*
«C# class»
AssignmentSysDev4::Airport
Attributes
- city : String
- country : String
- IATACode : String
Operations
+ AddNewTerminal()
+ Airport()
+ DeleteAirport()
+ UpdateAirport()
+ ViewAirport()
Airport
1
«C# class»
AssignmentS...4::Customer
Attributes
Operations
+ CancelTicket()
+ ChangeSeatLocation()
+ CreateAccount()
+ CreateCustomerAccount()
+ CreatePotentionalCustomerAcco...
+ Customer()
+ PurchaseTicket()
+ QueryFlightSchedual()
Flight
*
«C# class»
AssignmentSysDev4::Flight
Attributes
+ Airport : Airport
+ FlightClass : FlightClass
+ FlightNumber : String
«C# class»
AssignmentSysDev4::AirbusA319100
Attributes
- model : String
Operations
+ AirbusA319100()
+ GetModel() : String
«C# class»
AssignmentSysDev4::AirbusA330300
Attributes
- model : String
Operations
+ AirbusA330300()
+ GetModel() : String
«C# class»
AssignmentSysDev4::Boeing777200LR
Attributes
- model : String
Operations
+ Boeing777200LR()
+ GetModel() : String
«C# class»
AssignmentSysDev4::Boeing777300ER
Attributes
- model : String
Operations
+ Boeing777300ER()
+ GetModel() : String
«interface»
«C# interface»
AssignmentSysDev4::IPlane
Attributes
Operations
+ GetModel() : String
«C# class»
AssignmentSysDev4::BombardierQ400
Attributes
- model : String
Operations
+ BombardierQ400()
+ GetModel() : String
«C# class»
AssignmentSysDev4::Boeing7879
Attributes
- model : String
Operations
+ Boeing7879()
+ GetModel() : String
«C# class»
AssignmentSysDev4::BombardierCRJ705
Attributes
- model : String
Operations
+ BombardierCRJ705()
+ GetModel() : String
«C# class»
AssignmentSysDev4::Boeing7878
Attributes
- model : String
Operations
+ Boeing7878()
+ GetModel() : String
«C# class»
AssignmentSysDev4::Boeing767300ER
Attributes
- model : String
Operations
«C# class»
AssignmentSysDev4::Airline
Attributes
+ Brand : Brand
- airlineInstatnce : Airline
- name : String
Operations
+ AddAirline()
+ AddBrand(brandName : Brand)
+ Airline()
+ AirlineInstatnce() : Airline
+ DeleteAirline()
+ DeleteBrand()
+ UpdateAirline()
+ UpdateBrand()
+ ViewAirline()
+ ViewBrand()
Airline
*
Brand
1
«C# class»
AssignmentSysDev4::Brand
Attributes
+ Name : String
- name : String
Operations
+ AddBrand()
+ AddPlane()
+ AddPlanes(modelNumber : Integer, n...
+ Brand()
+ DeleteBrand()
+ DeletePlane()
+ UpdateBrand()
+ UpdatePlane()
+ ViewBrand()
+ ViewPlane()


+ Customer()
+ PurchaseTicket()
+ QueryFlightSchedual()
+ ReserveTicket()
+ UpgradeTicket()
+ ViewFlight()
+ ViewFlightHistory()
+ Airport : Airport
+ FlightClass : FlightClass
+ FlightNumber : String
+ name : String
- flightArrival : Airport
- flightClass : FlightClass
- flightDate : Integer
- flightDeparture : Airport
- flightNumber : String
- flightTime : Integer
- numberOfSeats : Integer
- PlaneID : Integer
Operations
+ Add(newFlight : Flight)
+ AddFlight()
+ AddFlightClass()
+ DeleteFlight()
+ DeleteFlightClass()
+ Display(indent : Integer)
+ Flight()
+ Remove(newFlight : Flight)
+ UpdateAirport()
+ UpdateFlight()
+ UpdateFlightClass()
+ ViewFlight()
Attributes
- model : String
Operations
+ Boeing767300ER()
+ GetModel() : String
«C# class»
AssignmentSysDev4::BombardierCRJ200
Attributes
- model : String
Operations
+ BombardierCRJ200()
+ GetModel() : String
«C# class»
AssignmentSysDev4::Employee
Attributes
Operations
+ AddAirCraft()
+ AddClassFlight()
+ AddFlight()
+ AddPlane()
+ CreateEmployeeAccount()
+ Employee()
+ SchedulaFlight()
«C# class»
AssignmentSysDev4::Plane
Attributes
+ Flight : Flight
- field : Integer
- id : Integer
- manufacturer : String
- model : String
Operations
+ AddPlane(modelNumber : Integer)
+ DeletePlane()
+ Plane()
+ UpdatePlane()
+ ViewPlane()
Flight
1
Plane
*
«C# class»
AssignmentSysDev4::FlightClass
Attributes
+ Ticket : Ticket
+ Ticket1 : Ticket
- flightClassName : Integer
Operations
+ AddNewFlightClass()
+ DeleteFlightClass()
+ FlightClass()
+ UpdateFlightClass()
+ ViewFlightClass()
Flight
*
FlightClass
1
FlightClass
*
FlightClass
*
«C# class»
AssignmentSysDev4::EmbraerE175
Attributes
- model : String
Operations
+ EmbraerE175()
+ GetModel() : String
«C# class»
AssignmentSysDev4::Ticket
Attributes
+ PurchaseDate : DateTime
- purchaseDate : DateTime
- ticketNumber : String
- _cancelstrategy : CancelStrategy
Operations
+ Cancel()
+ SetCancelStrategy(cancelstrategy : CancelStrategy)
+ Ticket()
Ticket
1
Ticket
1
Ticket1
1
«C# class»
AssignmentSysDev4::PlaneCreator
Attributes
Operations
+ FactoryMethod(modelNumber : Integer) :...
+ PlaneCreator()
«C# class»
AssignmentSysDev4::CompositeElement
Attributes
- elements : List<Flight>
Operations
+ Add(newConnection : Flight)
+ CompositeElement(baseFlight : Flight)
+ Display(indent : Integer)
+ Remove(oldConnection : Flight)
«C# class»
AssignmentSysDev4::PrimitiveElement
Attributes
Operations
+ Add(newFlight : Flight)
+ Display(indent : Integer)
+ PrimitiveElement(newFlight : Flight)
+ Remove(wrongFlight : Flight)
«C# class»
AssignmentSysDev4::BusinessClassCancel
Attributes
Operations
+ BusinessClassCancel()
+ Cancel(ticketToCancel : Ticket) : Boolean
«C# class»
AssignmentSysDev4::Program
Attributes
Operations
+ Program()
- Main(args : String[*])
«C# class»
AssignmentSysDev4::Facade
Attributes
- a : CustomerViewAccount
- b : EmployeeViewAccount
Operations
+ CustomerDataView()
+ EmployeeDataView(DelinquentStat...
«C# class»
AssignmentSysDev4::CancelStrategy
Attributes
Operations
+ Cancel(ticketToCancel : Ticket) : Boolean
+ CancelStrategy()
«C# class»
AssignmentSysDev4::EconomyCancel
Attributes
Operations
+ Cancel(ticketToCancel : Ticket) : Boolean
+ EconomyCancel()
«C# class»
AssignmentSysDev4::EmployeeViewAccount
Attributes
Operations
+ EmployeeViewAccount()
~ EmployeeView(DelinquentStatus : Boolean) : String
~ SimulateCustomerView() : String
«C# class»
AssignmentSysDev4::PremiumEconomyCancel
Attributes
Operations
+ Cancel(ticketToCancel : Ticket) : Boolean
+ PremiumEconomyCancel()
«C# class»
AssignmentSysDev4::CustomerViewAccount
Attributes
Operations
+ CustomerViewAccount()
~ AccountData() : String

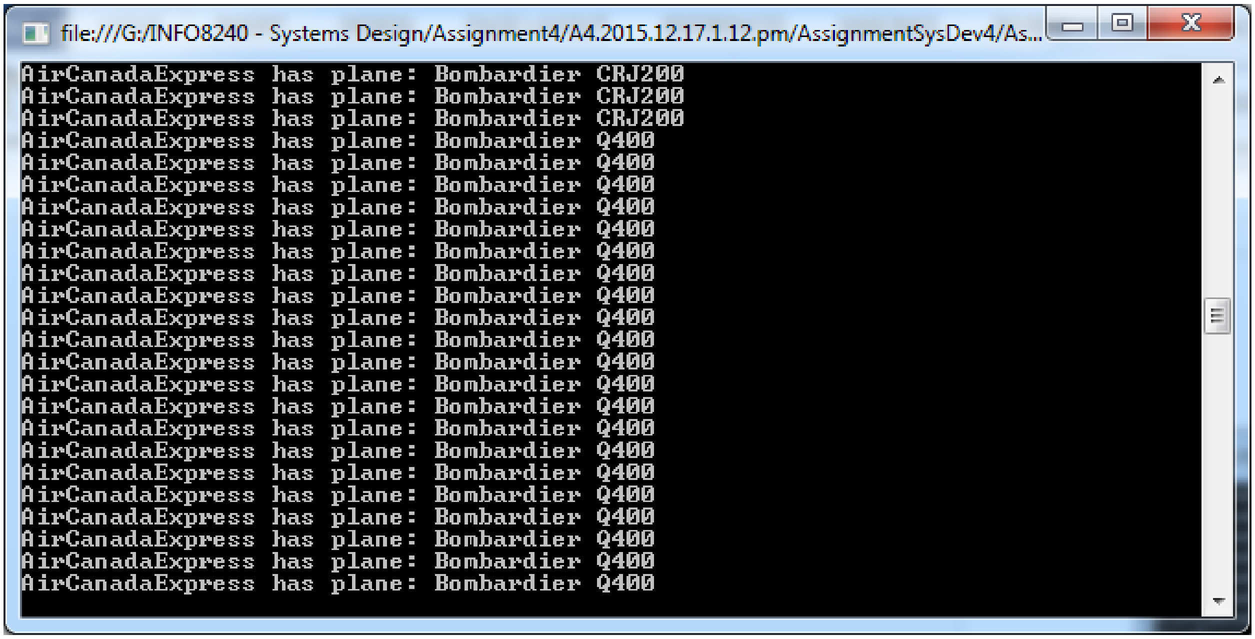

Airline
Employee
Customer
Brand
Plane
Flight
Airport
Ticket
has
Cancellation Strategy


file:///G:/INFO8240 - Systems Design/Assignment4/A4.2015.12.17.1.12.pm/AssignmentSysDev4/As...

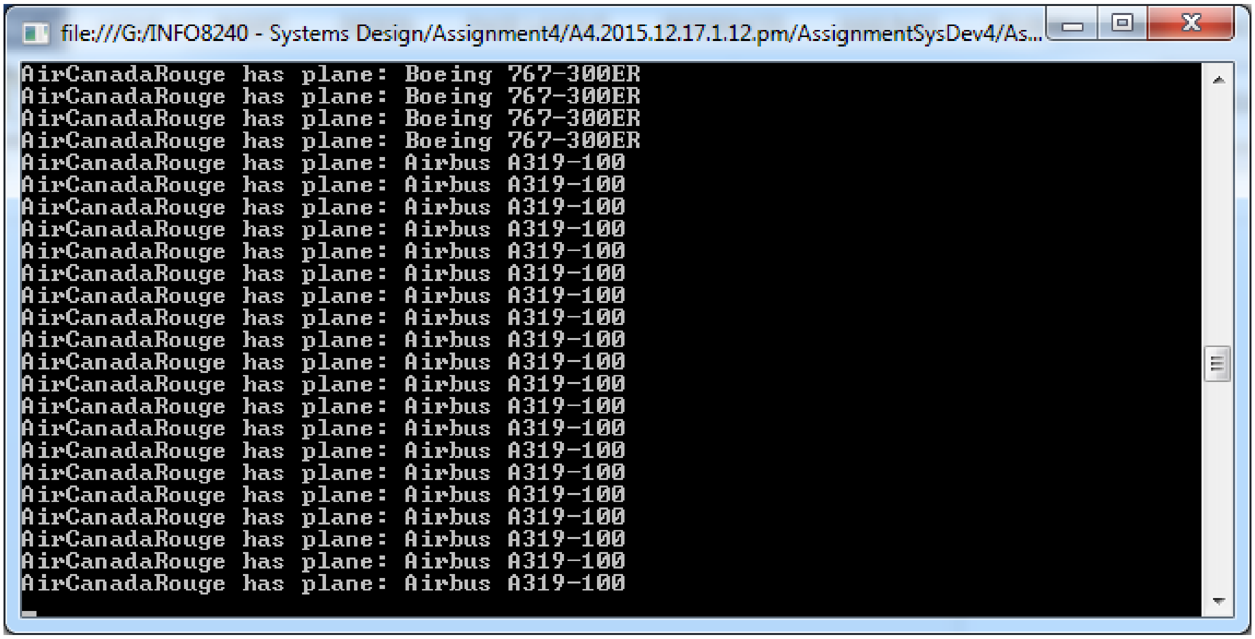
```
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
AirCanada has plane: Boeing 777-300ER
_
```


file:///G:/INFO8240 - Systems Design/Assignment4/A4.2015.12.17.1.12.pm/AssignmentSysDev4/As...

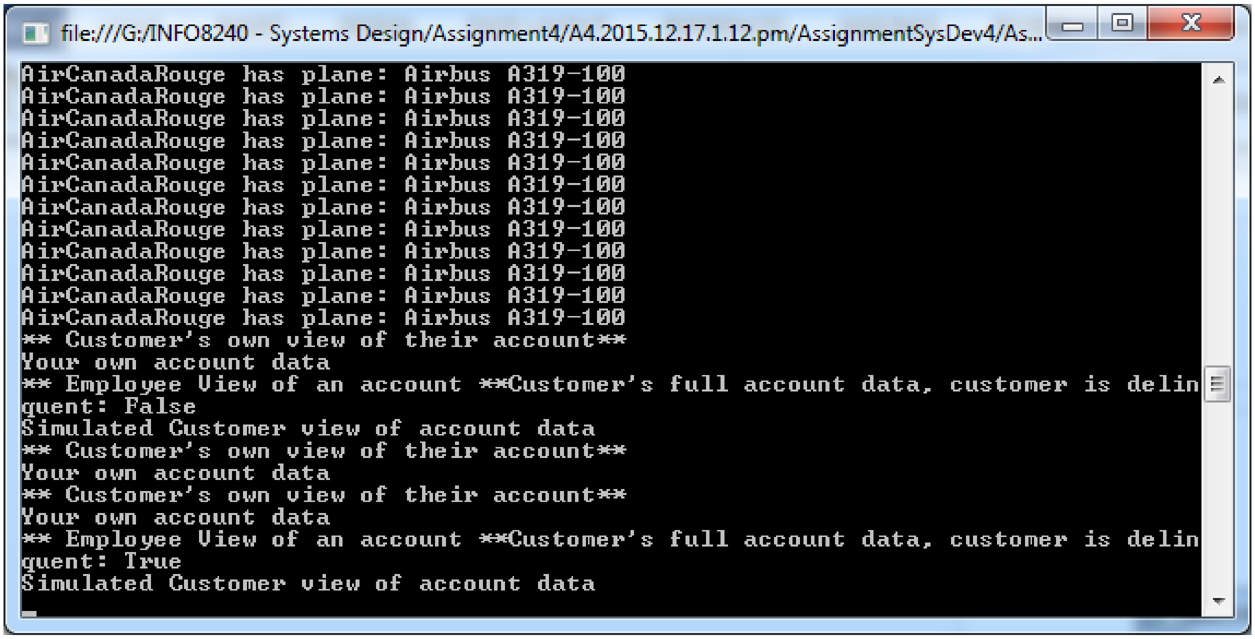
```
AirCanada has plane: Boeing787-9
AirCanada has plane: Boeing787-9
AirCanada has plane: Boeing787-9
AirCanada has plane: Boeing787-9
AirCanada has plane: Boeing787-9
AirCanada has plane: Boeing787-9
AirCanada has plane: Boeing787-9
AirCanada has plane: Boeing787-9
AirCanada has plane: Boeing787-9
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
AirCanada has plane: Boeing787-8
_
```


file:///G:/INFO8240 - Systems Design/Assignment4/A4.2015.12.17.1.12.pm/AssignmentSysDev4/As...

```
AirCanadaExpress has plane: Bombardier CRJ200
AirCanadaExpress has plane: Bombardier CRJ200
AirCanadaExpress has plane: Bombardier CRJ200
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
AirCanadaExpress has plane: Bombardier Q400
```

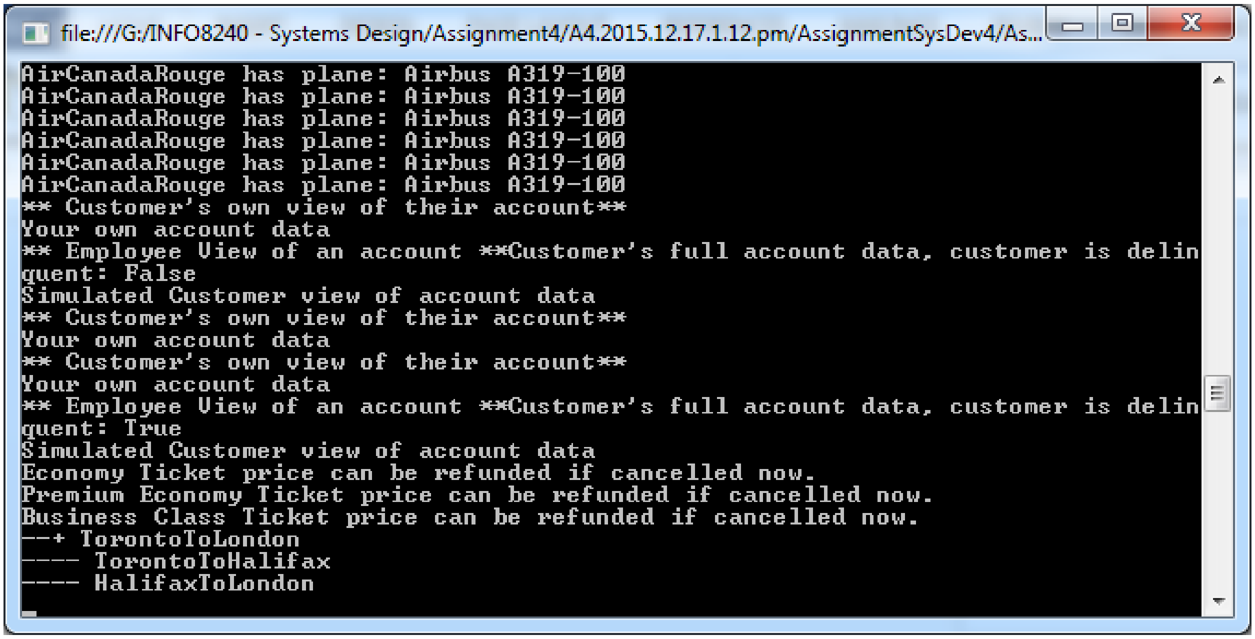

file:///G:/INFO8240 - Systems Design/Assignment4/A4.2015.12.17.1.12.pm/AssignmentSysDev4/As...

```
AirCanadaRouge has plane: Boeing 767-300ER
AirCanadaRouge has plane: Boeing 767-300ER
AirCanadaRouge has plane: Boeing 767-300ER
AirCanadaRouge has plane: Boeing 767-300ER
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
```



file:///G:/INFO8240 - Systems Design/Assignment4/A4.2015.12.17.1.12.pm/AssignmentSysDev4/As...

```
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
** Customer's own view of their account**
Your own account data
** Employee View of an account **Customer's full account data, customer is delin
quent: False
Simulated Customer view of account data
** Customer's own view of their account**
Your own account data
** Customer's own view of their account**
Your own account data
** Employee View of an account **Customer's full account data, customer is delin
quent: True
Simulated Customer view of account data
_
```



file:///G:/INFO8240 - Systems Design/Assignment4/A4.2015.12.17.1.12.pm/AssignmentSysDev4/As...

```
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
AirCanadaRouge has plane: Airbus A319-100
** Customer's own view of their account**
Your own account data
** Employee View of an account **Customer's full account data, customer is delin
quent: False
Simulated Customer view of account data
** Customer's own view of their account**
Your own account data
** Customer's own view of their account**
Your own account data
** Employee View of an account **Customer's full account data, customer is delin
quent: True
Simulated Customer view of account data
Economy Ticket price can be refunded if cancelled now.
Premium Economy Ticket price can be refunded if cancelled now.
Business Class Ticket price can be refunded if cancelled now.
--+ TorontoToLondon
---- TorontoToHalifax
---- HalifaxToLondon
_
```

Assignment4

Generated by Doxygen 1.8.10

Wed Dec 16 2015 18:16:10

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 AssignmentSysDev4 Namespace Reference

Classes

- class Account
- class AirbusA319100
- class AirbusA330300
- class Airline
- class Airport
- class Boeing767300ER
- class Boeing777200LR
- class Boeing777300ER

  This class defines a plane which extends the IPlane interface; other plane classes in this file have the same basic code, the different model numbers are specified for each one. This makes it possible to add more individual properties unique to a model of plane.
- class Boeing7878
- class Boeing7879
- class BombardierCRJ200
- class BombardierCRJ705
- class BombardierQ400
- class Brand
- class BusinessClassCancel

  A 'ConcreteStrategy' class This strategy class allows the purchase price of Business class tickets to be refunded if the ticket is cancelled within 96 hours, i.e. four days of puchase
- class CancelStrategy
- class CompositeElement

  Flights can also be used as Composite elements because one flight might be 'fixed', i.e. the customer has a narrow window of possible departure times at some point in their journey, but the primitive elements added to the list of the Composite element will be flights that are possible connections. The Composite pattern is used here because it allows for this kind of things, i.e. having a list of items with associated branches or 'leaf' items, and having the data grouped this way for flights will allow the system to show alternative flights to complete a journey.
- class Customer
- class EconomyCancel

  A 'ConcreteStrategy' class This stragegy class allows an Economy ticket's purchase price to be refunded if it is cancelled within 24 hours of purchase.
- class EmbraerE175
- class Employee
- class Flight
- class FlightClass

- interface IPlane

  Here we use the Factory design pattern to create planes. Each plane has a model, we allow the use of a simple integer model number. In a full system, planes would likely be picked from a list based on a database table.
- class Plane
- class PlaneCreator

  This Creator class uses a FactoryMethod to determine what type of plane object to set up.
- class PremiumEconomyCancel

  A 'ConcreteStrategy' class This strategy class allows a Premium Economy ticket's price to be refunded if the ticket is cancelled within 48 hours of purchase.
- class PrimitiveElement

  Flights can be primitive elements within the use of the composite pattern. This is because we wish to make lists of flights that offer alternative connections based on times, stopover times, etc. so customers have choices.
- class Program
- class Ticket

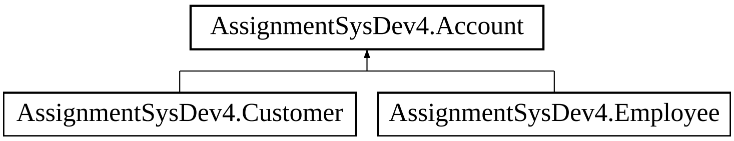
  The ticket class employs different cancellation strategies, set up in individual classes The use of the stragegy pattern for this would, potentially, allow other characteristics of the different ticket classes to be set up.

Chapter 5

Class Documentation

5.1 AssignmentSysDev4.Account Class Reference

Inheritance diagram for AssignmentSysDev4.Account:


AssignmentSysDev4.Account
AssignmentSysDev4.Customer
AssignmentSysDev4.Employee

Classes

- class **CustomerViewAccount**
- class **EmployeeViewAccount**
- class **Facade**

  A Facade is used here to allow Employees to see Detailed information about accounts, while customers have their own view of their account. This will allow the system to do things like allowing employees to simulate a customer's view and see data only to be shared with employees.

Public Member Functions

- void **ViewAccountInfo** (bool employeeInd, bool delinquentStatus)
- void **UpdateAccount** ()

Properties

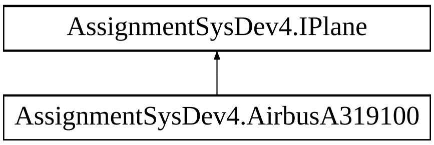
- Ticket **Ticket1** `[get, set]`
- bool **Delinquent** `[get, set]`
- bool **EmployeeInd** `[get, set]`

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Account.cs

5.2 AssignmentSysDev4.AirbusA319100 Class Reference

Inheritance diagram for AssignmentSysDev4.AirbusA319100:



Public Member Functions

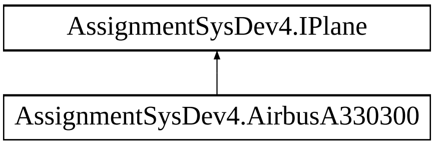
- string **GetModel** ()

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.3 AssignmentSysDev4.AirbusA330300 Class Reference

Inheritance diagram for AssignmentSysDev4.AirbusA330300:



Public Member Functions

- string **GetModel** ()

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.4 AssignmentSysDev4.Airline Class Reference

Public Member Functions

- void **AddAirline** ()
- void **ViewAirline** ()
- void **UpdateAirline** ()
- void **DeleteAirline** ()
- void **AddBrand** (Brand brandName)
- void **ViewBrand** ()
- void **UpdateBrand** ()
- void **DeleteBrand** ()

Static Public Member Functions

- static Airline **AirlineInstatnce** ()

Properties

- Brand **Brand** `[get, set]`

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Airline.cs

5.5 AssignmentSysDev4.Airport Class Reference

Public Member Functions

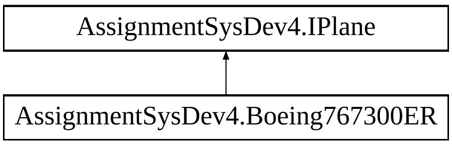
- void **AddNewTerminal** ()
- void **ViewAirport** ()
- void **DeleteAirport** ()
- void **UpdateAirport** ()

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Airport.cs

5.6 AssignmentSysDev4.Boeing767300ER Class Reference

Inheritance diagram for AssignmentSysDev4.Boeing767300ER:



Public Member Functions

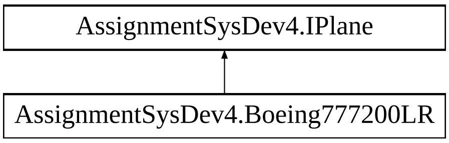
- string **GetModel** ()

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.7 AssignmentSysDev4.Boeing777200LR Class Reference

Inheritance diagram for AssignmentSysDev4.Boeing777200LR:



Public Member Functions

- string **GetModel** ()

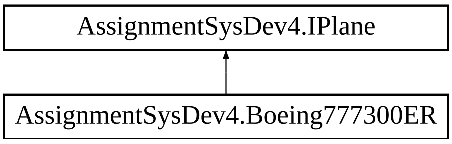
The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.8 AssignmentSysDev4.Boeing777300ER Class Reference

This class defines a plane which extends the IPlane interface; other plane classes in this file have the same basic code, the different model numbers are specified for each one. This makes it possible to add more individual properties unique to a model of plane.

Inheritance diagram for AssignmentSysDev4.Boeing777300ER:



Public Member Functions

- string **GetModel** ()

5.8.1 Detailed Description

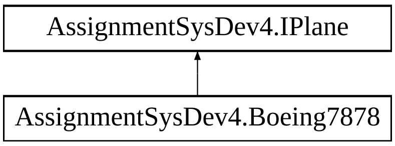
This class defines a plane which extends the IPlane interface; other plane classes in this file have the same basic code, the different model numbers are specified for each one. This makes it possible to add more individual properties unique to a model of plane.

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.9 AssignmentSysDev4.Boeing7878 Class Reference

Inheritance diagram for AssignmentSysDev4.Boeing7878:



Public Member Functions

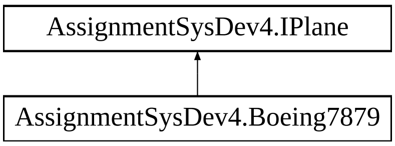
- string **GetModel** ()

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.10 AssignmentSysDev4.Boeing7879 Class Reference

Inheritance diagram for AssignmentSysDev4.Boeing7879:



Public Member Functions

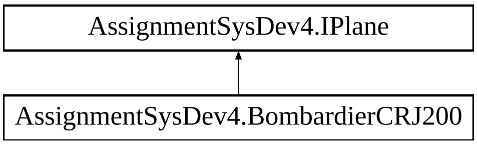
- string **GetModel** ()

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.11 AssignmentSysDev4.BombardierCRJ200 Class Reference

Inheritance diagram for AssignmentSysDev4.BombardierCRJ200:



Public Member Functions

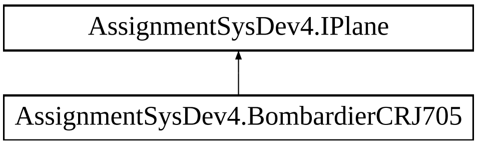
- string **GetModel** ()

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.12 AssignmentSysDev4.BombardierCRJ705 Class Reference

Inheritance diagram for AssignmentSysDev4.BombardierCRJ705:



Public Member Functions

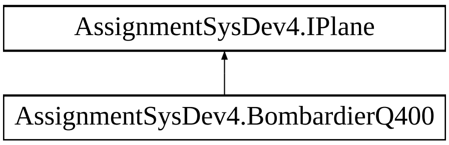
- string **GetModel** ()

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.13 AssignmentSysDev4.BombardierQ400 Class Reference

Inheritance diagram for AssignmentSysDev4.BombardierQ400:



Public Member Functions

- string **GetModel** ()

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.14 AssignmentSysDev4.Brand Class Reference

Public Member Functions

- void **AddBrand** ()
- void **UpdateBrand** ()
- void **DeleteBrand** ()
- void **ViewBrand** ()
- void **AddPlane** ()
- void AddPlanes (int modelNumber, int numberToAdd)

  Here we will take a modelNumber and a number of planes to add to a Brand. Then we can use methods defined in the PlaneCreator class to add those planes to the brand. PlaneCreator will use the Factory method.
- void **ViewPlane** ()
- void **UpdatePlane** ()
- void **DeletePlane** ()

Properties

- string **Name** `[get, set]`

5.14.1 Member Function Documentation

5.14.1.1 void AssignmentSysDev4.Brand.AddPlanes (int *modelNumber,* int *numberToAdd*)

Here we will take a modelNumber and a number of planes to add to a Brand. Then we can use methods defined in the PlaneCreator class to add those planes to the brand. PlaneCreator will use the Factory method.

Parameters

modelNumber	
numberToAdd	

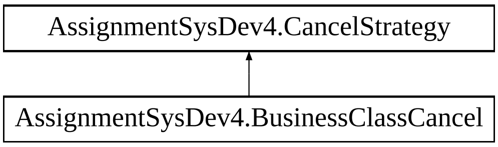
The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Brand.cs

5.15 AssignmentSysDev4.BusinessClassCancel Class Reference

A 'ConcreteStrategy' class This strategy class allows the purchase price of Business class tickets to be refunded if the ticket is cancelled within 96 hours, i.e. four days of puchase

Inheritance diagram for AssignmentSysDev4.BusinessClassCancel:


AssignmentSysDev4.CancelStrategy
AssignmentSysDev4.BusinessClassCancel

Public Member Functions

- override bool **Cancel** (Ticket ticketToCancel)

5.15.1 Detailed Description

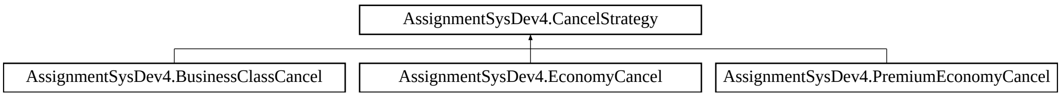
A 'ConcreteStrategy' class This strategy class allows the purchase price of Business class tickets to be refunded if the ticket is cancelled within 96 hours, i.e. four days of puchase

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Ticket.cs

5.16 AssignmentSysDev4.CancelStrategy Class Reference

Inheritance diagram for AssignmentSysDev4.CancelStrategy:


AssignmentSysDev4.CancelStrategy
AssignmentSysDev4.BusinessClassCancel
AssignmentSysDev4.EconomyCancel
AssignmentSysDev4.PremiumEconomyCancel

Public Member Functions

- abstract bool **Cancel** (Ticket ticketToCancel)

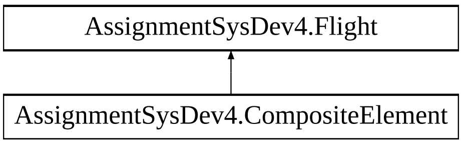
The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Ticket.cs

5.17 AssignmentSysDev4.CompositeElement Class Reference

Flights can also be used as Composite elements because one flight might be 'fixed', i.e. the customer has a narrow window of possible departure times at some point in their journey, but the primitive elements added to the list of the Composite element will be flights that are possible connections. The Composite pattern is used here because it allows for this kind of things, i.e. having a list of items with associated branches or 'leaf' items, and having the data grouped this way for flights will allow the system to show alternative flights to complete a journey.

Inheritance diagram for AssignmentSysDev4.CompositeElement:



Public Member Functions

- **CompositeElement** (Flight baseFlight)
- override void **Add** (Flight newConnection)
- override void **Remove** (Flight oldConnection)
- override void **Display** (int indent)

Additional Inherited Members

5.17.1 Detailed Description

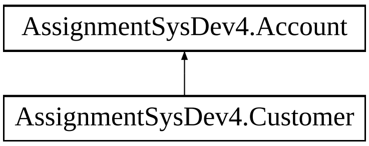
Flights can also be used as Composite elements because one flight might be 'fixed', i.e. the customer has a narrow window of possible departure times at some point in their journey, but the primitive elements added to the list of the Composite element will be flights that are possible connections. The Composite pattern is used here because it allows for this kind of things, i.e. having a list of items with associated branches or 'leaf' items, and having the data grouped this way for flights will allow the system to show alternative flights to complete a journey.

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Flight.cs

5.18 AssignmentSysDev4.Customer Class Reference

Inheritance diagram for AssignmentSysDev4.Customer:



Public Member Functions

- void **QueryFlightSchedual** ()
- void **ReserveTicket** ()
- void **PurchaseTicket** ()

- void **CancelTicket** ()
- void **UpgradeTicket** ()
- void **ChangeSeatLocation** ()
- void **ViewFlightHistory** ()
- void **CreateAccount** ()
- void **ViewFlight** ()
- void **CreateCustomerAccount** ()
- void **CreatePotentionalCustomerAccount** ()

Additional Inherited Members

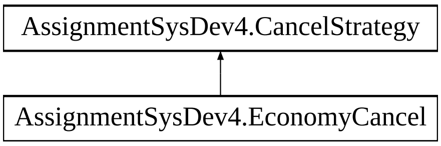
The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Customer.cs

5.19 AssignmentSysDev4.EconomyCancel Class Reference

A 'ConcreteStrategy' class This stragegy class allows an Economy ticket's purchase price to be refunded if it is cancelled within 24 hours of purchase.

Inheritance diagram for AssignmentSysDev4.EconomyCancel:


AssignmentSysDev4.CancelStrategy
AssignmentSysDev4.EconomyCancel

Public Member Functions

- override bool **Cancel** (Ticket ticketToCancel)

5.19.1 Detailed Description

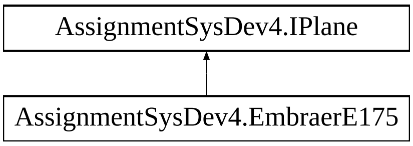
A 'ConcreteStrategy' class This stragegy class allows an Economy ticket's purchase price to be refunded if it is cancelled within 24 hours of purchase.

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Ticket.cs

5.20 AssignmentSysDev4.EmbraerE175 Class Reference

Inheritance diagram for AssignmentSysDev4.EmbraerE175:



Public Member Functions

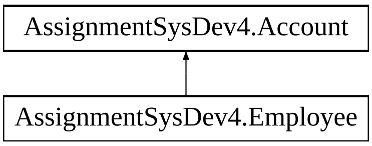
- string **GetModel** ()

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.21 AssignmentSysDev4.Employee Class Reference

Inheritance diagram for AssignmentSysDev4.Employee:



Public Member Functions

- void **AddPlane** ()
- void **AddClassFlight** ()
- void **SchedulaFlight** ()
- void **AddAirCraft** ()
- void **AddFlight** ()
- void **CreateEmployeeAccount** ()

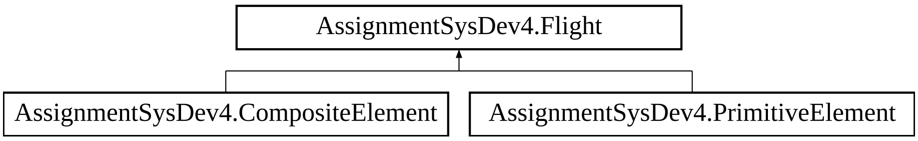
Additional Inherited Members

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Employee.cs

5.22 AssignmentSysDev4.Flight Class Reference

Inheritance diagram for AssignmentSysDev4.Flight:


AssignmentSysDev4.Flight
AssignmentSysDev4.CompositeElement
AssignmentSysDev4.PrimitiveElement

Public Member Functions

- virtual void **Add** (Flight newFlight)
- virtual void **Remove** (Flight newFlight)
- virtual void **Display** (int indent)
- void **ViewFlight** ()
- void **DeleteFlight** ()

- void **UpdateFlight** ()
- void **AddFlight** ()
- void **AddFlightClass** ()
- void **UpdateFlightClass** ()
- void **DeleteFlightClass** ()
- void **UpdateAirport** ()

Public Attributes

- string **name**

Properties

- FlightClass **FlightClass** `[get, set]`
- string **FlightNumber** `[get, set]`
- Airport **Airport** `[get, set]`

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Flight.cs

5.23 AssignmentSysDev4.FlightClass Class Reference

Public Member Functions

- void **AddNewFlightClass** ()
- void **UpdateFlightClass** ()
- void **ViewFlightClass** ()
- void **DeleteFlightClass** ()

Properties

- Ticket **Ticket** `[get, set]`
- Ticket **Ticket1** `[get, set]`

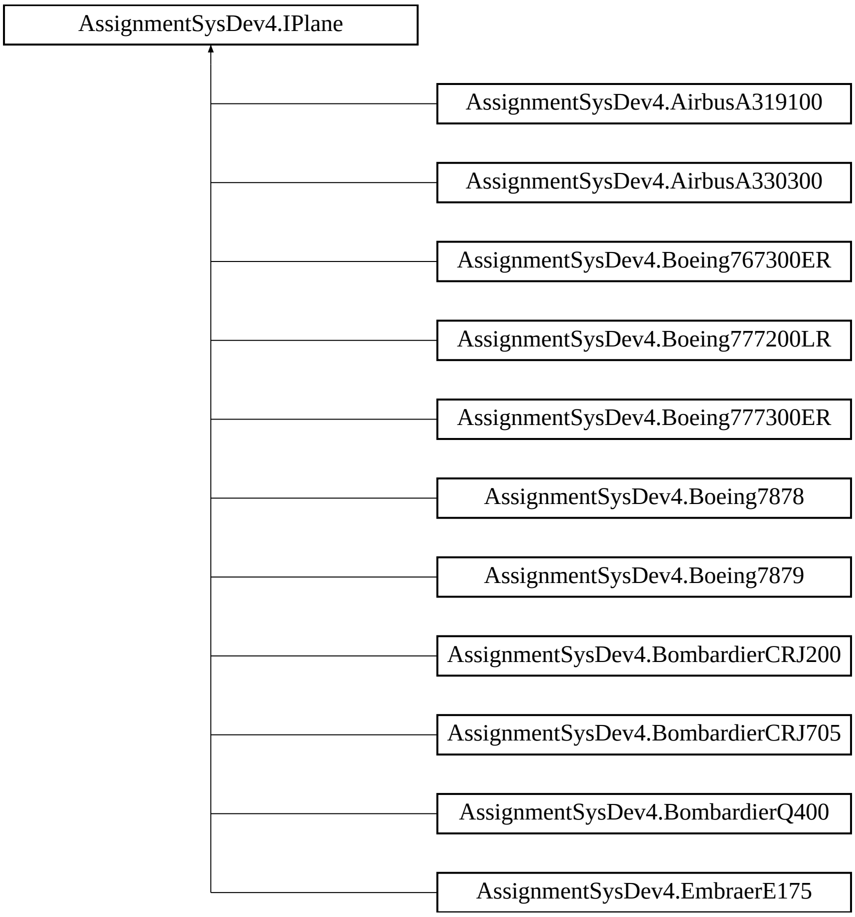
The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/FlightClass.cs

5.24 AssignmentSysDev4.IPlane Interface Reference

Here we use the Factory design pattern to create planes. Each plane has a model, we allow the use of a simple integer model number. In a full system, planes would likely be picked from a list based on a database table.

Inheritance diagram for AssignmentSysDev4.IPlane:


AssignmentSysDev4.IPlane
AssignmentSysDev4.AirbusA319100
AssignmentSysDev4.AirbusA330300
AssignmentSysDev4.Boeing767300ER
AssignmentSysDev4.Boeing777200LR
AssignmentSysDev4.Boeing777300ER
AssignmentSysDev4.Boeing7878
AssignmentSysDev4.Boeing7879
AssignmentSysDev4.BombardierCRJ200
AssignmentSysDev4.BombardierCRJ705
AssignmentSysDev4.BombardierQ400
AssignmentSysDev4.EmbraerE175

Public Member Functions

- string **GetModel** ()

5.24.1 Detailed Description

Here we use the Factory design pattern to create planes. Each plane has a model, we allow the use of a simple integer model number. In a full system, planes would likely be picked from a list based on a database table.

The documentation for this interface was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.25 AssignmentSysDev4.Plane.IPlane Interface Reference

Public Member Functions

- string **GetModel** ()

The documentation for this interface was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Plane.cs

5.26 AssignmentSysDev4.Plane Class Reference

Classes

- interface IPlane

Public Member Functions

- void **ViewPlane** ()
- void **UpdatePlane** ()
- void **DeletePlane** ()
- void **AddPlane** (int modelNumber)

Properties

- Flight **Flight** `[get, set]`

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Plane.cs

5.27 AssignmentSysDev4.PlaneCreator Class Reference

This Creator class uses a FactoryMethod to determine what type of plane object to set up.

Public Member Functions

- IPlane **FactoryMethod** (int modelNumber)

5.27.1 Detailed Description

This Creator class uses a FactoryMethod to determine what type of plane object to set up.

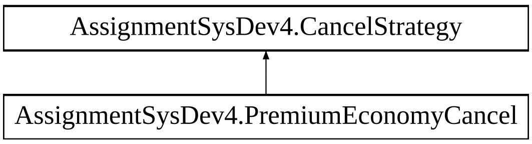
The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/PlaneCreator.cs

5.28 AssignmentSysDev4.PremiumEconomyCancel Class Reference

A 'ConcreteStrategy' class This strategy class allows a Premium Economy ticket's price to be refunded if the ticket is cancelled within 48 hours of purchase.

Inheritance diagram for AssignmentSysDev4.PremiumEconomyCancel:


AssignmentSysDev4.CancelStrategy
AssignmentSysDev4.PremiumEconomyCancel

Public Member Functions

- override bool **Cancel** (Ticket ticketToCancel)

5.28.1 Detailed Description

A 'ConcreteStrategy' class This strategy class allows a Premium Economy ticket's price to be refunded if the ticket is cancelled within 48 hours of purchase.

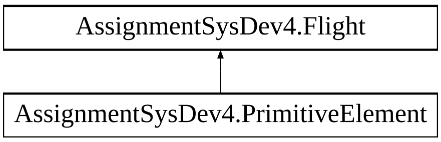
The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Ticket.cs

5.29 AssignmentSysDev4.PrimitiveElement Class Reference

Flights can be primitive elements within the use of the composite pattern. This is because we wish to make lists of flights that offer alternative connections based on times, stopover times, etc. so customers have choices.

Inheritance diagram for AssignmentSysDev4.PrimitiveElement:



Public Member Functions

- **PrimitiveElement** (Flight newFlight)
- override void **Add** (Flight newFlight)
- override void **Remove** (Flight wrongFlight)
- override void **Display** (int indent)

Additional Inherited Members

5.29.1 Detailed Description

Flights can be primitive elements within the use of the composite pattern. This is because we wish to make lists of flights that offer alternative connections based on times, stopover times, etc. so customers have choices.

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Flight.cs

5.30 AssignmentSysDev4.Program Class Reference

5.30.1 Detailed Description

An executive employee may want to create a batch of employee Accounts all at once Sometimes companies will hire people for lower level jobs in groups so they can go through orientation and training as a group.

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Program.cs

5.31 AssignmentSysDev4.Ticket Class Reference

The ticket class employs different cancellation strategies, set up in individual classes The use of the stragegy pattern for this would, potentially, allow other characteristics of the different ticket classes to be set up.

Public Member Functions

- void **SetCancelStrategy** (CancelStrategy cancelstrategy)
- void **Cancel** ()

Properties

- DateTime **PurchaseDate** `[get, set]`

5.31.1 Detailed Description

The ticket class employs different cancellation strategies, set up in individual classes The use of the stragegy pattern for this would, potentially, allow other characteristics of the different ticket classes to be set up.

The documentation for this class was generated from the following file:

- G:/INFO8240 - Systems Design/Assignment4/AssignmentSysDev4/AssignmentSysDev4/Ticket.cs

Index

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

namespace AssignmentSysDev4
{
    public class Account
    {
        private string email;
        private string password;
        private string lName;
        private string fName;
        private int ID;
        private Ticket ticket;
        private bool employeeInd = false;
        private bool delinquent = false;

        public Ticket Ticket
        {
            get
            {
                return ticket;
            }
            set
            {
                ticket = value;
            }
        }

        public bool Delinquent
        {
            get { return delinquent; }
            set
            {
                delinquent = value;
            }
        }


        public bool EmployeeInd
        {
            get
            {
                return employeeInd;

            }
            set
            {
                employeeInd = value;
            }
        }

        public void ViewAccountInfo(bool employeeInd, bool delinquentStatus)
        {


            if (employeeInd)
            {
                Facade.CustomerDataView();
                Facade.EmployeeDataView(delinquentStatus);

            }
            else
            {
                Facade.CustomerDataView();
            }

            Console.ReadKey();
        }

        internal class CustomerViewAccount
        {


            internal string AccountData()
            {
                return "Your own account data";
            }

        }

        internal class EmployeeViewAccount
        {
            internal string EmployeeView(bool DelinquentStatus)
            {
                return "Customer's full account data, customer is delinquent: " + DelinquentStatus;
            }

            internal string SimulateCustomerView()
            {
                return "Simulated Customer view of account data";
            }

        }


        /// <summary>
        /// A Facade is used here to allow Employees to see Detailed information about accounts,
        /// while customers have their own view of their account.  This will allow the system
        /// to do things like allowing employees to simulate a customer's view and see data
        /// only to be shared with employees.
        /// </summary>
        public static class Facade
        {
            static CustomerViewAccount a = new CustomerViewAccount();
            static EmployeeViewAccount b = new EmployeeViewAccount();

            public static void CustomerDataView()
            {
                Console.WriteLine("** Customer's own view of their account**");
                Console.WriteLine(a.AccountData());

            }

            public static void EmployeeDataView(bool DelinquentStatus)
            {
                Console.Write("** Employee View of an account **");
                Console.WriteLine(b.EmployeeView(DelinquentStatus));
                Console.WriteLine(b.SimulateCustomerView());
            }
        }

        public void UpdateAccount()
        {
            throw new System.NotImplementedException();
        }
    }
}
```

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

namespace AssignmentSysDev4
{
    public class Airline
    {
        private static Airline airlineInstatnce;

        private string name;

        //singleton Design patters
        //only one instance of Airline is allowed
        //lazy intialization
        public static Airline AirlineInstatnce ()
        {
            if ( airlineInstatnce == null )
            {
                airlineInstatnce = new Airline ();
            }

               return airlineInstatnce;
        }

        public Brand Brand
        {
            get
            {
                throw new System.NotImplementedException ();
            }
            set
            {
            }
        }

        public void AddAirline ()
        {
            throw new System.NotImplementedException ();
        }

        public void ViewAirline ()
        {
            throw new System.NotImplementedException ();
        }

        public void UpdateAirline ()
        {
            throw new System.NotImplementedException ();
        }

        public void DeleteAirline ()
        {
            throw new System.NotImplementedException();
        }

        public void AddBrand(Brand brandName)
        {
           // throw new System.NotImplementedException();

        }

        public void ViewBrand()
        {
            throw new System.NotImplementedException();
        }

        public void UpdateBrand()
        {
            throw new System.NotImplementedException();
        }

        public void DeleteBrand()
        {
            throw new System.NotImplementedException();
        }
    }
}
```

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

namespace AssignmentSysDev4
{
    public class Airport
    {
        private string IATACode;
        private string city;
        private string country;

        public void AddNewTerminal()
        {
            throw new System.NotImplementedException();
        }

        public void ViewAirport()
        {
            throw new System.NotImplementedException();
        }

        public void DeleteAirport()
        {
            throw new System.NotImplementedException();
        }

        public void UpdateAirport()
        {
            throw new System.NotImplementedException();
        }
    }
}
```

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

namespace AssignmentSysDev4
{
    public class Brand
    {
        private string name;


        public string Name
        {
            get {return name;}
            set { name = value; }

        }

        public void AddBrand()
        {
            throw new System.NotImplementedException();
        }

        public void UpdateBrand()
        {
            throw new System.NotImplementedException();
        }

        public void DeleteBrand()
        {
            throw new System.NotImplementedException();
        }

        public void ViewBrand()
        {
            throw new System.NotImplementedException();
        }

        public void AddPlane()
        {
            throw new System.NotImplementedException();
        }

        /// <summary>
        /// Here we will take a modelNumber and a number of planes to add to a Brand.
        /// Then we can use methods defined in the PlaneCreator class to add those
        /// planes to the brand.  PlaneCreator will use the Factory method.
        /// </summary>
        /// <param name="modelNumber"></param>
        /// <param name="numberToAdd"></param>
        public void AddPlanes(int modelNumber, int numberToAdd)
        {
            for (int x = 0; x < numberToAdd; x++)
            {
                PlaneCreator myPlaneCreator = new PlaneCreator();
                IPlane ConcretePlane = myPlaneCreator.FactoryMethod(modelNumber);
                Console.WriteLine(this.name + " has plane: " + ConcretePlane.GetModel());
            }
        }

        public void ViewPlane()
        {
            throw new System.NotImplementedException();
        }
        public void UpdatePlane()
        {
            throw new System.NotImplementedException();
        }

        public void DeletePlane()
        {
            throw new System.NotImplementedException();
        }
    }
}
```

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

namespace AssignmentSysDev4
{
    public class Customer : Account
    {

        public void QueryFlightSchedual()
        {
            throw new System.NotImplementedException();
        }

        public void ReserveTicket()
        {
            throw new System.NotImplementedException();
        }

        public void PurchaseTicket()
        {
            throw new System.NotImplementedException();
        }

        public void CancelTicket()
        {
            throw new System.NotImplementedException();
        }

        public void UpgradeTicket()
        {
            throw new System.NotImplementedException();
        }

        public void ChangeSeatLocation()
        {
            throw new System.NotImplementedException();
        }

        public void ViewFlightHistory()
        {
            throw new System.NotImplementedException();
        }

        public void CreateAccount()
        {
            throw new System.NotImplementedException();
        }

        public void ViewFlight()
        {
            throw new System.NotImplementedException();
        }

        public void CreateCustomerAccount()
        {
            throw new System.NotImplementedException();
        }

        public void CreatePotentionalCustomerAccount()
        {
            throw new System.NotImplementedException();
        }
    }
}
```

Discussion

PlaneCreator – A Factory design pattern was used to create plane objects. This was set up so that the levels of indirection made it easy for Program.cs to add planes to a Brand, simply by calling the AddPlanes method of the relevant brand, with a plane model number and a number of planes to add as parameters.

In this case, the use of a Factory design pattern seemed appropriate because planes are, in fact, a kind of manufactured good. A specific model of plane will have unique characteristics, which one can specify in the FactoryMethod of PlaneCreator. Different planes would have certain characteristics that would have to be completed in detail, and the use of the IPlane interface can enforce this.

In the case of the Airline, establishing it as a Singleton made sense because they are unique; you would not have to "Air Canada" companies at the same time.

For tickets, the different time limits were set up using the Strategy pattern. Testing this did raise some challenges when testing the code, because it was desirable to make sure the output changed when the allotted time had passed after which a ticket could no longer be cancelled. This was accomplished by hard-coding some dates temporarily for testing purposes. The use of the strategy pattern in this cases was selected because the strategy pattern lends itself to this type of situations, when you have several items that are similar but have slightly different details in how you execute some operations.

Flights use the Composite pattern because the composite pattern allows one to have composite objects and associated primitive objects (sometimes referred to as 'leaf' objects). In the case of selecting flights, this pattern was therefore desirable because one might want to consider a list of alternative flights and present it to the customer or an employee helping a customer. If the customer wants to go from one starting point to an end point, the system may have to find flights that start on the desired date, then search for connections and show all possible connecting flights for the customer to choose from that would complete their journey. So the first flight in a list would be fixed, followed by alternative connections. On a long journey, this 'tree' of flights could become complex with many branches and sub-branches, here a simple case is used to illustrate.

\

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

namespace AssignmentSysDev4
{



    public class Employee : Account
    {
        public void AddPlane()
        {
            throw new System.NotImplementedException();
        }

        public void AddClassFlight()
        {
            throw new System.NotImplementedException();
        }

        public void SchedulaFlight()
        {
            throw new System.NotImplementedException();
        }

        public void AddAirCraft()
        {
            throw new System.NotImplementedException();
        }

        public void AddFlight()
        {
            throw new System.NotImplementedException();
        }

        public void CreateEmployeeAccount()
        {
            throw new System.NotImplementedException();
        }




    }
}
```

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

namespace AssignmentSysDev4
{
    public class Flight
    {
        private string flightNumber;
        private int flightDate;
        private Airport flightDeparture = new Airport ();
        private Airport flightArrival = new Airport ();
        private int flightTime;
        private int numberOfSeats;
        private int PlaneID;
        public string name;
        private FlightClass flightClass = new FlightClass ();

        public virtual void Add(Flight newFlight) {}
        public virtual void Remove(Flight newFlight) { }

        public virtual void Display(int indent) { }
        public FlightClass FlightClass
        {
            get
            {
                throw new System.NotImplementedException ();
            }
            set
            {
            }
        }

        public string FlightNumber
        {
            get
            {
                return flightNumber;
            }
            set
            {
                flightNumber = value;
            }
        }

        public Airport Airport
        {
            get
            {
                throw new System.NotImplementedException ();
            }
            set
            {
            }
        }

        public void ViewFlight ()
        {
            throw new System.NotImplementedException ();
        }

        public void DeleteFlight ()
        {
            throw new System.NotImplementedException ();
        }
        public void UpdateFlight ()
        {
            throw new System.NotImplementedException ();

        }

        public void AddFlight()
        {
            throw new System.NotImplementedException();
        }

        //Façade Design Pattern
        public void AddFlightClass()
        {

            flightClass.AddNewFlightClass();
        }

        //Facade Design Pattern
        public void UpdateFlightClass()
        {

            flightClass.UpdateFlightClass();
        }

        //Facade Design Pattern
        public void DeleteFlightClass()
        {

            flightClass.DeleteFlightClass();
        }

        //Façade Design Pattern
        public void UpdateAirport()
        {

            Airport.UpdateAirport();
        }


    }

    /// <summary>
    /// Flights can be primitive elements within the use of the
    /// composite pattern.  This is because we wish to make lists
    /// of flights that offer alternative connections based on
    /// times, stopover times, etc. so customers have choices.
    /// </summary>
    class PrimitiveElement : Flight {
        // Constructor
        public PrimitiveElement(Flight newFlight)
            : base()
        {
            this.name = newFlight.name;
        }

        public override void Add(Flight newFlight)
        {
            Console.WriteLine(
              "Cannot add to a PrimitiveElement");
        }

        public override void Remove(Flight wrongFlight)
        {
            Console.WriteLine(
              "Cannot remove from a PrimitiveElement");
        }

        public override void Display(int indent)
        {
            Console.WriteLine(
              new String('-', indent) + " " + base.name);
        }
    }

    /// <summary>
    /// Flights can also be used as Composite elements because one flight
    /// might be 'fixed', i.e. the customer has a narrow window of possible
    /// departure times at some point in their journey, but the primitive elements
    /// added to the list of the Composite element will be flights that are
    /// possible connections.  The Composite pattern is used here because
    /// it allows for this kind of things, i.e. having a list of items with
    /// associated branches or 'leaf' items, and having the data grouped
    /// this way for flights will allow the system to show alternative
    /// flights to complete a journey.
    /// </summary>
    class CompositeElement : Flight
    {
        private List<Flight> elements =
          new List<Flight>();

        // Constructor
        public CompositeElement(Flight baseFlight)
            : base()
        {
            this.name = baseFlight.name;
        }

        public override void Add(Flight newConnection)
        {
            elements.Add(newConnection);
        }

        public override void Remove(Flight oldConnection)
        {
            elements.Remove(oldConnection);
        }

        public override void Display(int indent)
        {
            Console.WriteLine(new String('-', indent) +
              "+ " + base.name);

            // Display each child element on this node
            foreach (Flight d in elements)
            {

                d.Display(indent + indent);

            }
        }
    }
}
```

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

namespace AssignmentSysDev4
{
    public class FlightClass
    {
        private int flightClassName;

        public Ticket Ticket
        {
            get
            {
                throw new System.NotImplementedException();
            }
            set
            {
            }
        }

        public Ticket Ticket1
        {
            get
            {
                throw new System.NotImplementedException();
            }
            set
            {
            }
        }

        public void AddNewFlightClass()
        {
            throw new System.NotImplementedException();
        }

        public void UpdateFlightClass()
        {
            throw new System.NotImplementedException();
        }

        public void ViewFlightClass()
        {
            throw new System.NotImplementedException();
        }

        public void DeleteFlightClass()
        {
            throw new System.NotImplementedException();
        }
    }
}
```

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

namespace AssignmentSysDev4
{
    public class Plane
    {
        private int id;
        private string model;
        private string manufacturer;
        private int field;

        public interface IPlane
        {

            //void method();
            string GetModel();

        }

        public Flight Flight
        {
            get
            {
                throw new System.NotImplementedException();
            }
            set
            {
            }
        }

        public void ViewPlane()
        {
            throw new System.NotImplementedException();
        }

        public void UpdatePlane()
        {
            throw new System.NotImplementedException();
        }

        public void DeletePlane()
        {
            throw new System.NotImplementedException();
        }

        public void AddPlane(int modelNumber)
        {
            throw new System.NotImplementedException();
        }
    }
}
```

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


namespace AssignmentSysDev4
{
    /// <summary>
    /// Here we use the Factory design pattern to create planes.  Each plane has a
    /// model, we allow the use of a simple integer model number.  In a full system,
    /// planes would likely be picked from a list based on a database table.
    /// </summary>
    public interface IPlane
    {

        //void method();
        string GetModel();

    }

    /// <summary>
    /// This class defines a plane which extends the IPlane interface;
    /// other plane classes in this file have the same basic code,
    /// the different model numbers are specified for each one.
    /// This makes it possible to add more individual properties unique
    /// to a model of plane.
    /// </summary>
    class Boeing777300ER : IPlane
    {
        //public void Method() { }
        string model = "Boeing 777-300ER";
        public string GetModel()
        {
            return model;
        }

    }

    class Boeing777200LR : IPlane
    {

        string model = "Boeing 777-200LR";
        public string GetModel()
        {
            return model;
        }
    }

    class AirbusA330300 : IPlane
    {

        string model = "Airbus A330-300";
        public string GetModel()
        {
            return model;
        }
    }


    class Boeing7879 : IPlane
    {

        string model = "Boeing787-9";
        public string GetModel()
        {
            return model;
        }
    }


    class Boeing7878 : IPlane
    {

        string model = "Boeing787-8";
        public string GetModel()
        {
            return model;
        }
    }


    class EmbraerE175 : IPlane
    {

        string model = "EmbraerE175";
        public string GetModel()
        {
            return model;
        }
    }

    class BombardierCRJ705 : IPlane
    {

        string model = "Bombardier CRJ705";
        public string GetModel()
        {
            return model;
        }
    }

    class BombardierCRJ200 : IPlane
    {

        string model = "Bombardier CRJ200";
        public string GetModel()
        {
            return model;
        }
    }

    class BombardierQ400 : IPlane
    {

        string model = "Bombardier Q400";
        public string GetModel()
        {
            return model;
        }
    }


    class Boeing767300ER : IPlane
    {

        string model = "Boeing 767-300ER";
        public string GetModel()
        {
            return model;
        }
    }

     class AirbusA319100 : IPlane
    {

         string model = "Airbus A319-100";
        public string GetModel()
        {
            return model;
        }
    }

    /// <summary>
    /// This Creator class uses a FactoryMethod to determine what
    /// type of plane object to set up.
    /// </summary>
    public class PlaneCreator
    {
        public IPlane FactoryMethod(int modelNumber)
        {

            if (modelNumber == 1)
            {
                Boeing777300ER newPlane = new Boeing777300ER();
                return newPlane;

            }
            else if (modelNumber == 2)
            {
                Boeing777200LR newPlane = new Boeing777200LR();
                return newPlane;
            }
            else if (modelNumber == 3)
            {

                AirbusA330300 newPlane = new AirbusA330300();
                return newPlane;
            }

            else if (modelNumber == 4)
            {
                Boeing7879 newPlane = new Boeing7879();
                return newPlane;

            }
            else if (modelNumber == 5)
            {
                Boeing7878 newPlane = new Boeing7878();
                return newPlane;
            }
            else if (modelNumber == 6)
            {

                EmbraerE175 newPlane = new EmbraerE175();
                return newPlane;
            }
            else if (modelNumber == 7)
            {

                BombardierCRJ705 newPlane = new BombardierCRJ705();
                return newPlane;
            }

            else if (modelNumber == 8)
            {
                BombardierCRJ200 newPlane = new BombardierCRJ200();
                return newPlane;

            }
            else if (modelNumber == 9)
            {
                BombardierQ400 newPlane = new BombardierQ400();
                return newPlane;
            }
            else if (modelNumber == 10)
            {

                Boeing767300ER newPlane = new Boeing767300ER();
                return newPlane;
            }
            else
            {
                AirbusA319100 newPlane = new AirbusA319100();
                return newPlane;
            }


        }
    }
}
```

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


namespace AssignmentSysDev4
{
    /// An executive employee may want to create a batch of employee Accounts all at once
    /// Sometimes companies will hire people for lower level jobs in groups so they
    /// can go through orientation and training as a group.
    class Program
    {

        /// <summary>
        /// This main method will call various classes that demonstrate different design patterns.  These  ↙
    include
        /// Singleton, Facade, Composite, Factory and Strategy.
        /// Airline is Singleton
        /// Adding planes to a Brand uses Factory
        /// Getting a list of possible connecting flights uses Composite
        /// Facade is used to control Employees versus Customers and what they can do.
        /// Tickets use Strategy pattern to reflect the different cancellation policies.
        ///
        /// </summary>
        /// <param name="args"></param>
        static void Main(string[] args)
        {


           // Employee Executive = new Employee();

          //  Executive.CreateNewEmployeeAccounts();

            Airline AirCanadaLine = new Airline();


            Brand AirCanada = new Brand();
            AirCanada.Name = "AirCanada";
            AirCanadaLine.AddBrand(AirCanada);
            //AirCanada.Name = "Air Canada";

            //Brand Air Canada has 19 Boeing 777-300ER, 6 Boeing 777-200LR, 8 Airbus A330-300, 22 Boeing  ↙
    787-9, and 15 Boeing 787-8 aircrafts


           AirCanada.AddPlanes(1, 19);

           Console.ReadKey();

           AirCanada.AddPlanes(2, 6);
           AirCanada.AddPlanes(3, 8);
           AirCanada.AddPlanes(4, 22);
           AirCanada.AddPlanes(5, 15);

           Console.ReadKey();

          // Air Canada Express operates 15 Embraer E175, 16 Bombardier CRJ705, 25 Bombardier CRJ200, and  ↙
    21 Bombardier Q400

           Brand AirCanadaExpress = new Brand();
           AirCanadaExpress.Name = "AirCanadaExpress";
           AirCanadaLine.AddBrand(AirCanadaExpress);

           AirCanadaExpress.AddPlanes(6, 15);
```

```
            AirCanadaExpress.AddPlanes(7, 16);
            AirCanadaExpress.AddPlanes(8, 25);
            AirCanadaExpress.AddPlanes(9, 21);

            Console.ReadKey();

            //Air Canada Rouge operates 9 Boeing 767-300ER and 20 Airbus A319-100 aircraft.

            Brand AirCanadaRouge = new Brand();
            AirCanadaRouge.Name = "AirCanadaRouge";
            AirCanadaLine.AddBrand(AirCanadaRouge);

            AirCanadaRouge.AddPlanes(10, 9);
            AirCanadaRouge.AddPlanes(11, 20);

            Console.ReadKey();

            Account Employee = new Account();
            Employee.EmployeeInd = true;

            //An employee should not have a delinquent account of their own
            //And they can view their own account infor
            Employee.ViewAccountInfo(Employee.EmployeeInd, false);

            Account Customer = new Account();
            Customer.Delinquent = true;
            //An employee should be able to see whether a customer account has delinquent
            //status for something like failing to pay for tickets on time.
            Customer.ViewAccountInfo(Customer.EmployeeInd, Customer.Delinquent);
            Customer.ViewAccountInfo(Employee.EmployeeInd, Customer.Delinquent);



            Ticket economyTicket = new Ticket();
            Customer.Ticket = economyTicket;
            economyTicket.PurchaseDate = DateTime.Now;
            economyTicket.SetCancelStrategy(new EconomyCancel());
            economyTicket.Cancel();

            Ticket premiumEconomyTicket = new Ticket();
            premiumEconomyTicket.PurchaseDate = DateTime.Now;
            premiumEconomyTicket.SetCancelStrategy(new PremiumEconomyCancel());
            premiumEconomyTicket.Cancel();

            Ticket businessClassTicket = new Ticket();
            businessClassTicket.PurchaseDate = DateTime.Now;
            businessClassTicket.SetCancelStrategy(new BusinessClassCancel());
            businessClassTicket.Cancel();


            Flight possibleConnection1 = new Flight();
            possibleConnection1.name = "TorontoToHalifax";

            Flight possibleConnection2 = new Flight();
            possibleConnection2.name = "HalifaxToLondon";

            Flight StartFlight = new Flight();
            StartFlight.name = "TorontoToLondon";
            CompositeElement root = new CompositeElement(StartFlight);

            possibleConnection1.name = "TorontoToHalifax";
            PrimitiveElement newFlightPrimitive =  new PrimitiveElement(possibleConnection1);


            root.Add(new PrimitiveElement(possibleConnection1));
            root.Add(new PrimitiveElement(possibleConnection2));
            root.Display(2);

            Console.ReadKey();

        }
    }
}
```

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

namespace AssignmentSysDev4
{
    /// <summary>
    /// The ticket class employs different cancellation strategies, set up in individual classes
    /// The use of the stragegy pattern for this would, potentially, allow other characteristics
    /// of the different ticket classes to be set up.
    /// </summary>
    public class Ticket
    {
        private string ticketNumber;
        private DateTime purchaseDate;

         private CancelStrategy _cancelstrategy;

        public void SetCancelStrategy(CancelStrategy cancelstrategy)
        {
            this._cancelstrategy = cancelstrategy;
        }


        public void Cancel()
        {
            _cancelstrategy.Cancel(this);

        }



        public DateTime PurchaseDate
        {
            get
            {
                return purchaseDate;

            }
            set
            {
                purchaseDate = value;
            }
        }

    }

    public abstract class CancelStrategy
    {
        public abstract bool Cancel(Ticket ticketToCancel);
    }

    /// <summary>
    /// A 'ConcreteStrategy' class
    /// This stragegy class allows an Economy ticket's purchase price to be refunded if it is cancelled    ↙
    within 24 hours of purchase.
    /// </summary>
    class EconomyCancel : CancelStrategy
    {
        public override bool Cancel(Ticket ticketToCancel)
        {
            int hoursToCancelForEconomy = 24;

            DateTime CurrentTime = DateTime.Now;
            if ((CurrentTime - ticketToCancel.PurchaseDate).TotalHours <= hoursToCancelForEconomy)
            {
                Console.WriteLine("Economy Ticket price can be refunded if cancelled now.");
                return true;
            }
            else
            {
                Console.WriteLine("Economy Ticket price can not be refunded cancelled now.");
                return false;
            }

        }
    }

    /// <summary>
    /// A 'ConcreteStrategy' class
    /// This strategy class allows a Premium Economy ticket's price to be refunded if the ticket is        ↙
    cancelled within 48 hours of purchase.
    ///
    /// </summary>
    class PremiumEconomyCancel : CancelStrategy
    {
        public override bool Cancel(Ticket ticketToCancel)
        {
            DateTime CurrentTime = DateTime.Now;
            int hoursToCancelForPremium = 48;

            if ((CurrentTime - ticketToCancel.PurchaseDate).TotalHours <= hoursToCancelForPremium)
            {
                Console.WriteLine("Premium Economy Ticket price can be refunded if cancelled now.");
                return true;
            }
            else
            {
                Console.WriteLine("Premium Economy Ticket price can not be refunded if cancelled now.");
                return false;
            }

        }
    }

    /// <summary>
    /// A 'ConcreteStrategy' class
    /// This strategy class allows the purchase price of Business class tickets to be refunded if the      ↙
    ticket is cancelled within 96 hours, i.e. four days of puchase
    /// </summary>
    class BusinessClassCancel : CancelStrategy
    {
        public override bool Cancel(Ticket ticketToCancel)
        {
            DateTime CurrentTime = DateTime.Now;
            int hoursToCancelForBusiness = 96;

            if ((CurrentTime - ticketToCancel.PurchaseDate).TotalHours <= hoursToCancelForBusiness)
            {
                Console.WriteLine("Business Class Ticket price can be refunded if cancelled now.");
                return true;
            }
            else
            {
                Console.WriteLine("Business Class Ticket can not be refunded if cancelled now.");
                return false;
            }
        }
    }
```

```
}
```