You are hired by a company that creates airplane ticket selling systems. You are responsible for the system that sells tickets for a flight. Each flight has a fixed number of seats, base fare for the ticket and a list of already sold tickets. There are 2 types of tickets (economy and business) which have slight differences in how the price is calculated. Ticket machines sell (create) ticket objects.

Classes:

TicketSellingMachine: has the general data about the flight and also is responsible for creating Ticket objects and storing them. There are different machines for business and economy class.

- Has a constructor which takes flight code and departure time (as string) as parameters. Flight destination and starting point info are taken from the flight code.
- Has a list for already sold tickets. In that list are stored all the tickets that have been sold.
- Has a method for setting the base price and number of seats.
- Has a method for printing out info about all the sold tickets. (Ticket 1: ... Ticket 2:...)
- Has a method for writing info about all the tickets to .txt file.
- Has a method for printing out the number of free seats.
- Has a method for selling the ticket (this method should be divided into sub methods, see the example in the end.) which takes the buyers name as a parameter and uses DateTime.Now as buying time.
- Has a method for selling the ticket (this method should be divided into sub methods, see the example in the end.) which takes the buyers name and buyingTime as a parameter. Use this method for testing!

Tickets can be sold only if there are free places on the airplane. If there are no free places, then a notification should be displayed.

If ticket is sold, call also ticket's PrintInfo() method to print the info about the sold ticket.

Create also an interface for the class. Business class machines sell business type tickets and economy machines economy type tickets.

Ticket: represents the ticket object

- Has passengers name, flight date, from, to, price, ticket type and seat number (1 for the first ticket sold, 2 for the second ticket sold etc)
- Has a method for printing out all the info listed in the previous point in a following format:

  Name: Mari Maasikas

  Flight date: 21.03.2018

  Price: 567 euros

  From: Tallinn to Tokyo

  Ticket type: 1

  Seat number: 6

  Have a nice flight!

Additional info:

-FlightCodes

FlightCodes have a fixed structure:

1) The code for the operating company (length:2)
2) CityCode A (length:3)
3) CityCode B (length:3)
4) A number to show the direction: if the number is 2 it means the flight is from A->B, if the number is 5 then the flight is from B->A
5) Letter „b“ if there is business class available on this flight

Some city codes:

TYO – Tokyo
TLL – Tallinn
BER - Berlin
KBP – Kiev
KRS – Kuressaare

Example flight codes:

FlightCode	Starting point	Destination	Business class
EATLLTYO2b	Tallinn	Tokyo	Yes
EATLLTYO5b	Tokyo	Tallinn	Yes
TAKRSTLL5	Tallinn	Kuressaare	no

Destination and starting point for the flight should be set based on the flight code (they should be not asked separately).

Flights always have an economy class but do not always have a business class. If there is no business class on the flight, then business class tickets cannot be sold. (Business class tickets can only be sold when the flight code ends with „b“.) Test it!

Recommendation: use a separate method to get city name based on the city code.

-Ticket price

There are multiple factors involved for setting the price. All of them make the base price *x* times more expensive and should be checked every time a ticket is sold. The factors are checked and applied (if necessary) to the base price in the same order as listed here.

1) Day of the flight:

   If the flight is on Friday or Saturday, ticket is 15% more expensive.

2) How early is the ticket bought:

Price does not change when the ticket is sold 6 or more months before flight departure date. With every next month the price increases (6-n)*0.1 times. If there is less than a month remaining until the departure then the price increases 0.6 times.

*For example: 2 months before the flight date the price increases (6-2)*0.1=0.4 times.*

3) Occupancy rate (*täituvus*); how many of the available seats are sold out (if plane has 60 seats and 30 are sold out then occupancy rate is 30/60*100%=50%).

   If 0-25% of the seats are sold out: it does not affect the price

   If 26-50% of the seats are sold out: the price increases by 10%.

   If 51-75% of the seats are sold out: the price increases by 13%.

   If 76%-100% of the seats are sold out: the price increases by 17%.

   - For business class ticket the weekday does not affect the price.
   - Also the occupancy rates are double for business class (20% for 26-50%; 26% for 51%-75% and 34% for 76%-100%)

Example: economy ticket with base price 100; bought 3 months in advance for Sunday when the plane is 70% full.

*Friday price: 100*0.15 + 100=115*

3 months price: 115((6-3)*0.1) + 115 = 149.5*

*Occupancy rate price: 149.5 * 0.17 + 149.5 = 168.94*

Final price is: 168.94

-Departure time:

To simplify, we may assume that the plane departs always at 00:00 and thereby using only date in DateTime constructor is sufficient (we do not need to specify a time).

Examples: https://www.dotnetperls.com/datetime-parse

-Ticket type

1- Business class ticket
2- Economy class ticket

Testing:

Add nUnit and create at least 10 tests (Focus on calculating the price. Test the sub-methods). Add tests after creating methods! (Not when the code is complete).

Main method:

Create at least 4 different flight ticket machines. Test different scenarios (plane is full, business class cannot be created etc). You can use the flight/airport codes provided here or create your own values. Have at least 4 different cities.

With 2 machines sell at least 60 tickets (with each). Create an array with 10 human names and choose the names for the ticket buyers randomly from that array. Dates can be the same or use random for generating different months.

Things to keep in mind:

- Use clean code principles!
- Add classes to separate files.
- When doing class inheritance use as little code as possible (write over as few methods as possible!)
- Void methods cannot be tested with nUnit. If you want to test them, then add return type to method or properties (for example: int numberOfFreeSeats) which you can then use in tests.
- If you are feeling adventurous, try using Dictionary for storing city codes and names (https://www.dotnetperls.com/dictionary). This is not mandatory.
- Do it step-by-step. Do not try to do it all at once.

Example of the dividing the SellTicket method into submethods:

```
    /*Create ticket object and add it to sold tickets list. Use DateTime now as
  buying time*/
    public void SellTicket(string name)
    {
    }

  /*Create ticket object and add it to sold tickets list. Specify the buying time
  yourself. Use this method for testing purposes*/
    public void SellTicket(string name, DateTime buyingTime)
    {
    }
    /*Creates and returns the ticket object. Do not change the signature*/
    internal AirplaneTicket CreateTicket()
    {
    }

    /*Apply weekday rate to the price. You may change the signature*/
    internal double ApplyWeekdayRate(double price)
    {
    }

    /*Apply time rate to the price. You may change the signature*/
    public double ApplyTimeRate(double price)
    {
    }

    /*Apply occupancy rate to the price. You may change the signature*/
    internal double ApplyOccupancyRate(double price)
    {
    }
```

Example of possible usage:

```
        static void Main(string[] args)
         {
             /*Creates a new ticket machine for a flight from Tallinn to Tokyo on sept 5,
2018*/
             BusinessTicketMachine btm = new BusinessTicketMachine("EATLLTYO2b", "2018-09-
05");

             EconomyTicketMachine etm = new EconomyTicketMachine("EATLLTYO2b", "2018-09-
        05");


             /*Sets base ticket price to 100 and the number of seats to 60*/
             etm.SetPriceData(100, 60);

             etm.SellTicket("Mary Jones"); //sells ticket to Mary Jones. Uses DateTime now
for calculating the price.
             /*The day is Friday so the price is 100*0.15+100 = 115. The flight is more
than 6 months away and it is the first ticket so these factors
               do not affect the price.*/

             etm.SellTicket("Mary Jones", "2018-09-04"); //sells ticket to Mary Jones.
uses sept 4 2018 for calculating the price
             /*flight is on friday so base fare is 115. It is bought 1 day before
departure so the price increases by 60% (0.6 times). 115*0.6+115=184. It is the second
ticket sold and the occupancy rate is under 25 so it does not affect the price.*/

             etm.PrintInfo(); //prints out info for both of the sold tickets
             etm.PrintFreeSeats(); //prints 58
         }
```

Advanced (6p):

1) Usually the number of tickets sold is bigger than the number of seats in the plane. This is due to some people not showing up for their flight. For this there is a feature for adding the number of extra seats to the flights that can be sold more than the number of seats. This applies for the economy class only. Create an additional constructor for ticket machine which takes the number of extra seats to add as a parameter.
2) For business class tickets there is also an option for buying an refundable ticket. This means the price is 20% more expensive. This is asked from the user:

„Your business class ticket price is 500 euros. Would you like to buy a refundable ticket instead? It would cost 600 euros. y/n “

If user enters „y“, a refundable ticket is sold.

If user enters „n“, a regular business class ticket is sold.

This info is added to the ticket (displayed in PrintInfo() method)

3) Add a method for displaying two of the highest, two of the lowest ticket prices and the average ticket price. Add it to ticket machine class. (Do not use built in Min and Max functions, create the logic yourself with loop(s)). If there are less than 5 tickets sold, then print only one highest and one lowest price. If there are less than 3 tickets sold, then this method will display a notification saying „Too few tickets!“.