INFORMATION & COMMUNICATION TECHNOLOGY ENGLISH COURSE

Course	:	Mock assessment
Teachers	:	GL, GRTS, HVT, KUAH, KPYS, MANO, SCWN & THAQ
Accepted resources	:	You are allowed to use everything on paper (books, notes, etc.) and on your laptop, but only what you bring in; you are not allowed to borrow something from someone else. During the assessment it is not allowed to use the network. You should make the exam yourself: so no communication with MSDN or google for help and no communication with other students, like using Facebook, e-mail, Skype, Dropbox, mobile phone, etc.

Grading scheme

Question	:	*1*	*2*	*3*	*4*
Points	:	3	24	42	31

+-

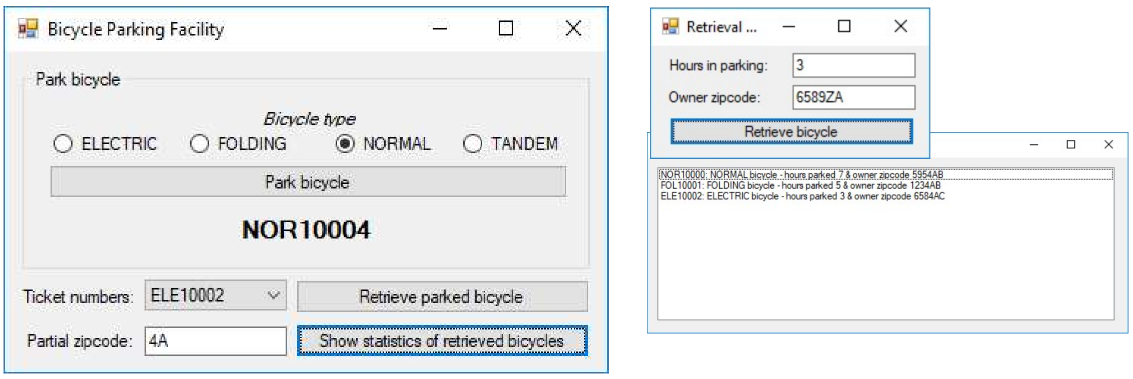
Introduction to a bicycle parking facility

A bicycle parking facility operated with tickets number to keep track of which bicycles are parked and retrieved from the facility. To gain insight about where the clients come from, a zipcode is asked when a bicycle is retrieved. Based on the zipcode, an overview is shown of the retrieved bicyles and hours in parking.

You're tasked to create a simplified application to administrate the bicycles entering and leaving the parking. The application should be able to generate ticket numbers, calculate the parking price and show information about bicycles from certain areas.

For this app, the user interface has already been made (see Figure 1). You will need to add and implement the classes to make the app work (see Diagram 1).

More specific instructions will be provided in the assignments.



Bicycle type: ELECTRIC FOLDING [✓ NORMAL] TANDEM

NOR10004

Ticket numbers: ELE10002

Partial zipcode: 4A

Hours in parking: 3

Owner zipcode: 6589ZA

```
NOR10000: NORMAL bicycle - hours parked 7 & owner zipcode 5954AB
FOL10001: FOLDING bicycle - hours parked 5 & owner zipcode 1234AB
ELE10002: ELECTRIC bicycle - hours parked 3 & owner zipcode 6584AC
```

Figure 1: The supplied GUI

When you implement the program you should follow the UML diagram below.

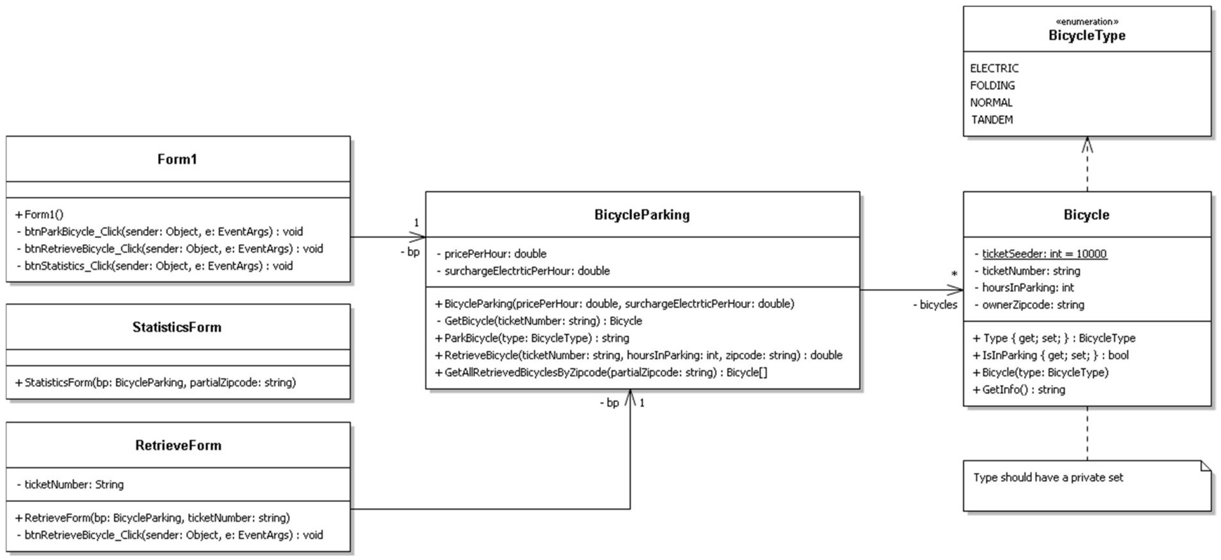

«enumeration»
BicycleType
ELECTRIC
FOLDING
NORMAL
TANDEM
Form1
+ Form1()
- btnParkBicycle_Click(sender: Object, e: EventArgs) : void
- btnRetrieveBicycle_Click(sender: Object, e: EventArgs) : void
- btnStatistics_Click(sender: Object, e: EventArgs) : void
1
- bp
BicycleParking
- pricePerHour: double
- surchargeElectrticPerHour: double
+ BicycleParking(pricePerHour: double, surchargeElectrticPerHour: double)
- GetBicycle(ticketNumber: string) : Bicycle
+ ParkBicycle(type: BicycleType) : string
+ RetrieveBicycle(ticketNumber: string, hoursInParking: int, zipcode: string) : double
+ GetAllRetrievedBicyclesByZipcode(partialZipcode: string) : Bicycle[]
*
- bicycles
Bicycle
- ticketSeeder: int = 10000
- ticketNumber: string
- hoursInParking: int
- ownerZipcode: string
+ Type { get; set; } : BicycleType
+ IsInParking { get; set; } : bool
+ Bicycle(type: BicycleType)
+ GetInfo() : string
Type should have a private set
StatisticsForm
+ StatisticsForm(bp: BicycleParking, partialZipcode: string)
RetrieveForm
- ticketNumber: String
+ RetrieveForm(bp: BicycleParking, ticketNumber: string)
- btnRetrieveBicycle_Click(sender: Object, e: EventArgs) : void
- bp
1

Diagram 1: Incomplete start-up UML Class Diagram

Use the startup-project to implement the assignments. You will have to implement a BicycleType enumeration (assignment 1), Bicycle class (assignment 2), BicycleParking class (assignment 3) and everything in the three Form classes (assignment 4).

<u>When needed, you may add additional methods or properties to, for example, make the instance variables accessible.</u>

Assignment 1: The “BicycleType” enumeration (3 points)

This application considers four (4) possible bicycle types. Create an enumerator called **BicycleType** in the file “BicycleType.cs”. It must contain the mentioned values as specified by Diagram 1.

Assignment 2: The “Bicycle” class (6 + 10 + 8 = 24 points)

Assignment 2a:

The class **Bicycle** must have three (3) instance variables and two (2) properties to store information about a bicycle. The *Type* of a bicycle can only be changed from within the class/object.

Implement the required code based on Diagram 1.

Note that the `public string GetInfo()` is supplied to you in the *Bicycle*-class. Uncomment this method.

Assignment 2b:

The instance variables *ownerZipcode* and *hoursInParking* should only get valid values assigned to them:

- *ownerZipcode*: cannot get an empty string assigned to it. When this happens the *ownerZipcode* should become *Unknown* instead;
- *hoursInParking*: can only be 1 or more.
  In case the value is not valid, assign the default value *1.*

Implement the required code for this.

Assignment 2c:

The constructor of the class **Bicycle** should initialize a bicycle object according to:

- The value of its input parameter. Note that *ownerZipcode* and *hoursInParking* should not be initialized as this should only be set when a bicycle is retrieved;
- That it is in the parking facility;
- Assign a unique ticked number with the format *<first 3 letters of the type><auto-incremented unique number>*
  E.g. *NOR10000*, *ELE10002*, *NOR10003*, etc.

  HINT: Getting the first 3 letter from a string can be done as following, where *str* is of the type string:
  ```
      str.Substring(0,3);
  ```

Implement the constructor: `public Bicycle(BicycleType type)`.

Assignment 3: The “BicycleParking” class (3 + 4 + 7 + 4 + 14 + 10 = 42 points)

Assignment 3a:
The class **BicycleParking** should be able to store information in instance variables.

Implement the required code based on Diagram 1.

Assignment 3b:
The constructor of the class **BicycleParking** should initialize the BicycleParking-object according to the value of its input parameters and an empty collection to store Bicycle-objects.

Implement the constructor: `public BicycleParking(double pricePerHour, double surchargeElectricPerHour)`

Assignment 3c:
It should be possible to search for a bicycle with a specific ticket number. Make sure a Bicycle-object is only returned if one is found, in all other cases return *null*.

Implement the method: `private Bicycle GetBicycle(string ticketNumber)`

Assignment 3d:
It should be possible to park a bicycle and receive the generated ticket number. Create and add a new Bicycle-object to the appropriate collection. Finally return the generated ticket number.

Implement the method: `public string ParkBicycle(BicycleType type)`

Assignment 3e:
It should be possible to retrieve a parked bicycle and receive the parking cost for it. This should be done in two steps:

1. Make sure the correct bicycle, which must still be in the parking, is updated according with the input parameters (i.e. *hoursInParking* and *zipcode*). Do not forget to also mark the bicycle as retrieved.
2. Calculate the parking price based on the bicycle type and amount of *hoursInParking*:
    - Normal bicycles should pay the regular *pricePerHour;*
    - Electric bicycles should pay the regular price and an additional fee based on *surchargeElectricPerHour*;
    - Folding bicycles are half the size, so they pay half the regular price;
    - Tandem bicycles are double the size, so they pay double the regular price.

Return the parking price if a bicycle with the ticket number is in the parking facility and was not already retrieved. In all other cases return -1.

Implement the method: `public double RetrieveBicycle(string ticketNumber, int hoursInParking, string zipcode)`

Assignment 3f:
It should be possible to see all the retrieved bicycles come from a specific zone. This is based on the owner’s zipcode and it should be possible to only supply a partial zipcode (e.g. if *4A* is supplied, all bicycles with an owner’s zipcode such as *1234AB, 2584AZ,* etc. should be returned).

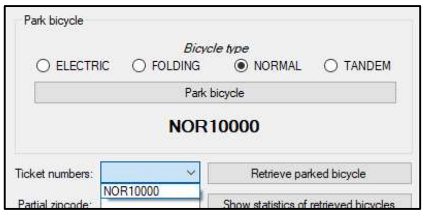
Implement the method: `public Bicycle[] GetAllRetrievedBicyclesByZipcode(string partialZipcode)`

Assignment 4: The form classes (3 + 7 + 12 + 9 = 31 points)

Assignment 4a:
When the application starts, the required BicycleParking-object should be created. Determine for yourself what prices you want to charge.

Implement the constructor: `public Form1()`

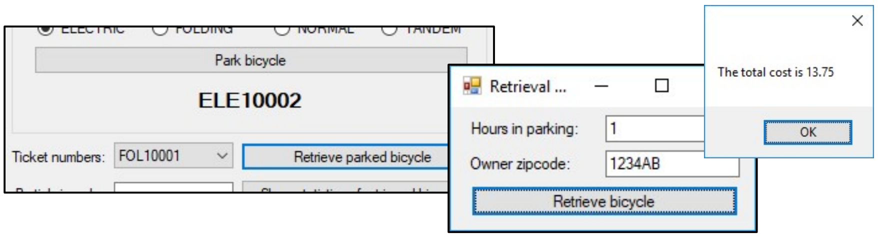
Assignment 4b:
When the 'Park bicycle'-button is clicked, a bicycle with the selected type should be added.
The returned ticket number should be shown in the appropriate label and added to the Ticket number ComboBox.
In the next assignment you will make use of the ComboBox to return a bicycle.



Implement the method: `private void btnParkBicycle_Click(object sender, EventArgs e)`

Assignment 4c:
To retrieve a bicycle, a ticket number must be selected in the ComboBox and then the 'Retrieve parked bicycle'-button should be clicked.

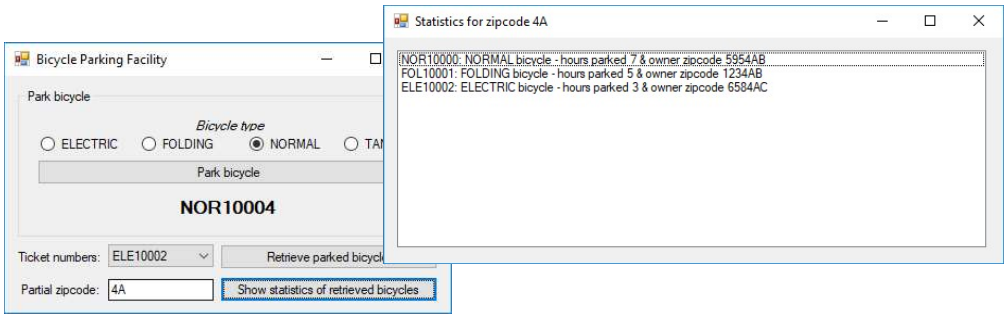
The **RetrieveForm** should be shown where additional information can be supplied. When the '*Retrieve bicycle*'-button is clicked, the price for the bicycle with the selected ticket number should be displayed in a message box.
If the price is -1, you should show an appropriate message such as "No parked bicycle was found with selected ticket number".
After showing a message box, the **RetrieveForm** should automatically close itself.



The total cost is 13.75

Implement the method: `private void btnRetrieveBicycle_Click(object sender, EventArgs e)` and make the required changes in the code of the **RetrieveForm** class based on Diagram 1.

Assignment 4d:
It should be possible to view which retrieved bicycles come from a specific zone. A (partial) zipcode should be supplied and then the 'Show statistics of retrieved bicycles'-button should be clicked.



```
NOR10000: NORMAL bicycle - hours parked 7 & owner zipcode 5954AB
FOL10001: FOLDING bicycle - hours parked 5 & owner zipcode 1234AB
ELE10002: ELECTRIC bicycle - hours parked 3 & owner zipcode 6584AC
```

- assignment 4d continues on next page -

The **StatisticsForm** should be shown and the title bar should display a string in the format *Statistics for zipcode <partial zipcode>*. In addition, the retrieved bicycles should be displayed in the ListBox.

Implement the method: `private void btnStatistics_Click(object sender, EventArgs e)` and make the required changed in the code of the **StatisticsForm** class based on Diagram 1.

END of mock assessment.