**OOP Basics (Cont’d)**

Data Hiding and Object Initialization

In this section, we will discuss about some key concepts on OOP:

* Data hiding (Encapsulation)
* Property
* Constructor
* Method overloading
* Constructor overloading
* XML Commenting

Data Hiding:

In object oriented programming, we must hide the internal data of an object i.e. we have to keep the data private. But in last several examples and practices, we have kept the data (field) public. For an example, see the following code, it was the Person class what we have written in *Walkthrough 1* where we have kept firstName, middleName and lastName public.

class Person

{

public string firstName;

public string middleName;

public string lastName;

}

This is a violation of encapsulation concept. So, we must update Person class, making all the data/field private.

class Person

{

private string firstName;

private string middleName;

private string lastName;

}

Then how can we access these personObj data from PersonInformationUI()? Putting a period after personObj, we are not getting firstName, middleName and lastName in the list. How can we solve this?

We can write methods to get the access of these private data as follows: Here, we write it only for firstName attribute only.

public void SetFirstName(string firstName)

{

this.firstName = firstName;

}

public string GetFirstName()

{

return firstName;

}

Now, we can set a value of firstName by SetFirstName(string firstName) method.

personObject.SetFirstName(irstNameTextBox.Text);

and to get data from personObj we may write:

string firstName = personObject.GetFirstName();

Property:

In *Data Hiding* topics, we see, for three attributes of person class you need to wirte six methods which really shows Person class little bit awkarding. We should not write methods for this purpose. Another cause is “Method represents the responsibility, so for accessing data we shouldn’t write methods.” Better we can try property. So, remove methods from our code and write property.

class Person

{

private string firstName;

private string middleName;

private string lastName;

public string FirstName

{

get { return firstName; }

set { firstName = value; }

}

public string MiddleName

{

get { return middleName; }

set { middleName = value; }

}

public string LastName

{

get { return lastName; }

set { lastName = value; }

}

}

To access data of person object using property from PersonInformationUI()class is as follows:

personObject.FirstName = firstNameTextBox.Text;

personObject.MiddleName = middleNameTextBox.Text;

personObject.LastName = lastNameTextBox.Text;

If we don’t need to share our data with our client code, we don’t need to write property for this data i.e. data should be private. If we want to keep the data read only, write the property with get part only, don’t write set part.

We can write property for getting access of combinational data only. We can write

MyFullName property instead of GetMyFullName() method of Person class.

public string MyFullName

{

get

{

return firstName + " " + middleName + " " + lastName;

}

}

Generally, these kinds of properties have only get part.

Auto Property:

Even we could Auto Property feature without writing the property and field seperately. See the Auto Property style for three data of Account class. If we haven’t any logic inside set and get method, Auto property is a good choise.

public string AccountNumber { set; get; }

public string CustomerName { set; get; }

public double Balance { private set; get; }

Practice 1: Open *WalkThrough1* from the *Practices* folder and change the person class’s public field to public Auto Property.

Constructor:

It’s used to initialize an object when it is created. Or even to do some initial activity.

public Person(string firstName, string middleName, string lastName)

{

this.firstName = firstName;

this.middleName = middleName;

this.lastName = lastName;

}

See, when we create object from Person class we have to provide these three data as follows:

Person personObject = new Person(firstNameTextBox.Text, middleNameTextBox.Text, lastNameTextBox.Text);

Practice 2: Open the code of *Practice 1* and write a constructor for Person class. And use this constructor from UI class.

Commenting and XML Commenting:

Method Overloading:

In our Calculator example, suppose we need a method by which we can add three numbers. How can we do it? We can write a method, ‘AddThreeNumber’ which will takes three numbers as parameter and returns the add result. So, we have two add methods:

‘Add’ for adding two numbers.

‘AddThreeNumber’ for adding three numbers.

Better we rename ‘Add’ method to AddTwoNumber. Good. But think about the caller of our code. After creating object of Calculator class, (s)he will get total five methods for arithmetic operation. ‘AddTwoNumber’ and ‘AddThreeNumber’ really do similar job, first one adds two numbers and second one adds three numbers. Ultimately both methods do the same job: Doing add.

So, if we write distinct methods for these kinds of similar jobs, caller of our code will see lots of methods of a class and this makes him more confuse about the object’s responsibilities. Better use method overloading concept here. Write Add method with two parameters and with three parameters.

public double Add(double firstNumber, double secondNumber)

public double Add(double firstNumber, double secondNumber, double thirdNumber)

Now the caller of our code is really happy, (s)he will only see only one *Add* method in his/her code. Finally, we can say similar responsibilities should be written using method overloading concept.

Constructor Overloading:

We use constructor to initialize an object. If we want to initialize an object in several ways, we need constructor overloading. If the caller of Person class wants in initialize it in several ways we need to update our Person class:

public Person()

{

}

public Person(string firstName, string lastName)

{

this.firstName = firstName;

this.LastName = lastName;

}

public Person(string firstName, string middleName, string lastName)

{

this.firstName = firstName;

this.middleName = middleName;

this.lastName = lastName;

}

We can reuse Person(string firstName, string lastName)in our third constructor:

public Person()

{

}

public Person(string firstName, string lastName)

{

this.firstName = firstName;

this.LastName = lastName;

}

public Person(string firstName, string middleName, string lastName) : this (firstName, lastName)

{

this.middleName = middleName;

}

Now Person object can be created in three ways. That means, person object can be initialized in three different ways.

1. Person with no initialization.
2. Person with firstName and lastName
3. Person with firstName, middleName and lastName

Practice 3: Open *OOPWalkThrough1* project from *Practices* Folder and write three Constructors in Person class.

Practice 4: Open *BankAccountOperation* project from *Practices* Folder and make all public field private, write appropriate property, write one or two constructor and change the UI code accordingly.

Practice 5: Open *CircleExample* project from *Practices* Folder and make all public field private, write appropriate property, write one or two constructor and change the UI code accordingly.

Value Type and Ref Type:

What is Value Type?

What is Ref Type?

What is Out?

What about User defined Type?

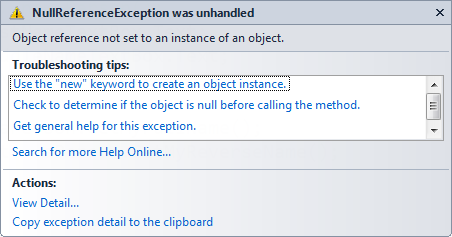
A reference of a class (user defined type can be null).

Person aPerson = null;

NullReferenceException will be thrown from application when you use a reference which holds null value instead of object. For example:

Person aPerson = null;

aPerson.GetName();



You can create as many objects as you want from a class. For example:

Person person1 = new Person();

Person person2 = new Person();

Person person3 = new Person();

Person person4 = new Person();

person1 = person2; // What will happen?

person1.GetName(); //What will happen?

Now, you are familiar with Object Oriented concept, you know how to find object from problem domain, how to write class and create object from it. Also, you know what is Property, constructor. Do a try to solve the following problems using OOP concept.

Practice 1: Make an application by which user can calculate result.

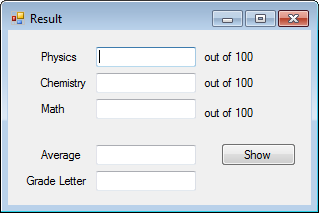
If average => 80 Grade will be A+

If average => 70 Grade will be B+

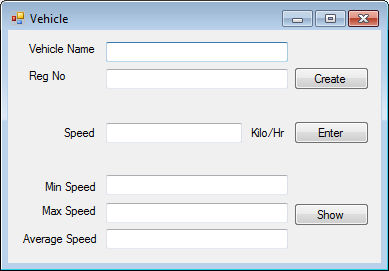
If average => 60 Grade will be C+

If average => 50 Grade will be D+

If average < 50 Grade will be F



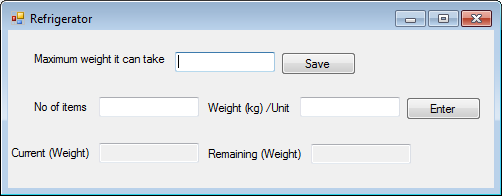
Practice 2: User will create a vehicle once at a time. Then enter as many Speed data as (s)he has. Anytime, (s)he will click *Show* button, Min, Max and Avg speed will be shown.



Practice 3: Make an application, Refrigerator as below. Initially user will set *Maximum weight it can take* (once only). Then *No of items* and *Weight/Unit* will be entered as much as user wants. When *Enter* button will be clicked *Current and Remaining* weight will be displayed. Noted that if there will be a possibility of overflow, system will show a message (in a messagebox) that “Your given amount can’t be entered because it will be over flown.”

Caution: Refrigerator class must return Current and Remaining weight in double (NOT string or any other data types.

For a good solution of this problem, you might need concept of exception handling. Try it.



**Introduction**

In the last two chapters, we have been familiarized with object and its attributes and responsibilities. We have even written code to implement our Object Oriented thoughts for solving several problems.

Though we have solved very simple problems, but at least we have learned ‘How to find object’ from a problem domain.

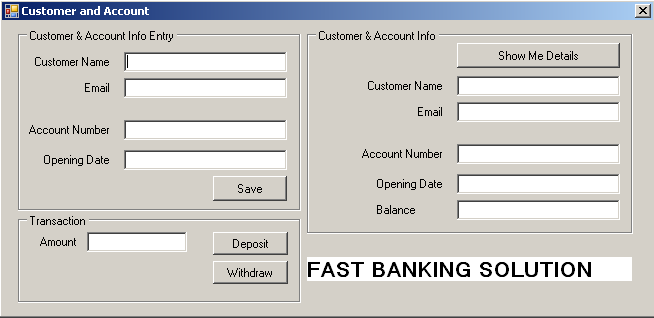
Moreover, I have talked about Property, Constructor and know their essences inside our code.

So far we have done is: ‘finding only one object from problem domain’. But truly, in real problem you may find several objects and the relationship between them. In this chapter, we will talk about Association Relationship and also implement it inside our code.

**Association Relationship between two objects:**

Association relationship means an object depends on another object to complete its responsibilities. For an example, Customer object depends on account object to complete its responsibilities.

How many objects we find from the following story? Try to find out two objects here.



Yes, we have got here two domain objects:

*Customer*

has name, email address.

*Account*

has account number, balance and opening date.

Also deposit and withdraw are the functionality of it.

Is there any relationship between Account and Customer object?

Yes, Customer has an account i.e. Customer has a relation with account type object and also we can tell Customer depends on Account. This type of relationship is called *Association* relationship. Here, one customer has only one account and visa-versa so it’s a one-to-one relationship.

Now, it’s time to write both classes. See the code below:

public class Account

{

public string Number { set; get; }

public string OpeningDate { set; get; }

public double Balance { private set; get; }

public Account(string number, string openingDate):this()

{

Number = number;

OpeningDate = openingDate;

}

public Account()

{

Balance = 0;

}

public string Deposit(double amount)

{

Balance += amount;

return "Deposited";

}

public string Withdraw(double amount)

{

if (Balance - amount >= 0)

{

Balance -= amount;

return "Withdrawn";

}

else

{

return "No sufficient balance to withdraw";

}

}

public class Customer

{

public string Name { set; get; }

public string Email { set; get; }

public Account AnAccount { set; get; }

public Customer(string name, string email)

{

Name = name;

Email = email;

}

public Customer()

{

}

}

Code of Save button click will be as follows:

private void saveButton\_Click(object sender, EventArgs e)

{

customerObj = new Customer();

Account anAccount = new Account(accountNumberEntryTextBox.Text,   
 openingDateEntryTextBox.Text);

customerObj.Name = customerNameEntryTextBox.Text;

customerObj.Email = emailEntryTextBox.Text;

customerObj.CustomerAccount = anAccount;

MessageBox.Show(customerObj.Name + " has opened an account with " +

customerObj.CustomerAccount.Number + " number");

}

Customer type reference, customerObj has been declared after the definition of BankUI class so that we can use this reference in several event handler methods.

public partial class BankUI : Form

{

private Customer customerObj = null;

Code of Deposit button click as follows:

private void depositButton\_Click(object sender, EventArgs e)

{

amount = Convert.ToDouble(amountTextBox.Text);

if (customerObj.CustomerAccount.Deposit(amount))

{

MessageBox.Show(amount + " Tk. has been deposited");

}

else

{

MessageBox.Show("Operation failed");

}

}

Code of Withdraw button click as follows:

private void withdrawButton\_Click(object sender, EventArgs e)

{

amount = Convert.ToDouble(amountTextBox.Text);

if (customerObj.CustomerAccount.Withdraw(amount))

{

MessageBox.Show(amount + " Tk. has been withdrawn");

}

else

{

MessageBox.Show("Insufficient amount");

}

}

Code of ShowDetails button click as follows:

private void showDetailsButton\_Click(object sender, EventArgs e)

{

customerNameDisplayTextBox.Text = customerObj.Name;

emailDisplayTextBox.Text = customerObj.Email;

accountNumberDisplayTextBox.Text = customerObj.CustomerAccount.Number;

openingDateDisplayTextBox.Text = customerObj.CustomerAccount.OpeningDate;

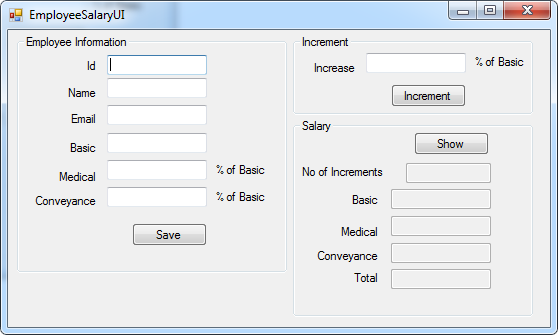
balanceTextBox.Text = customerObj.CustomerAccount.Balance.ToString();

}

You will find the above code in *Walkthroughs* folder. Project name: *FastBankAccountApp*

**Practice 1:** Try to do the same what we have above. Just repeat it in your own way and do a discusstion with your peer.

**Practice 2:** Employee Salary Application:



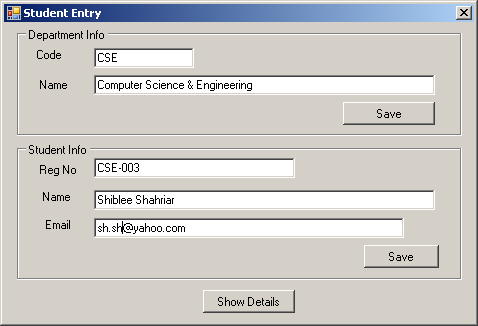
Using this application, user will save employee information one time only then (s)he will give increment to employee as many times as (s)he wants.

**Guideline**: First do brain-storming to find out two objects along with your peer. Use pen and pencil and then start coding. Do not open the solution before your best try.

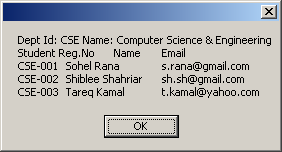
Find the solution in *Solution* folder. Project name: *EmployeeSalaryApp* and compare it with your code.

**One-to-Many Association relationship:**

Consider the following application. User will enter department information and then (s)he will enter maximum 15 students under the saved department.

****

When *Show Details* is clicked, department and all saved information will be displayed in a MessageBox as following format.

****

We will find two objects here: Student and Department. But this example is not same as previous examples and practices. Here, a department might have many students and a student can register in a single department. See the following code:

Student class code:

public class Student

{

private string regNo;

private string name;

private string email;

public Student(string regNo, string name, string email)

{

this.regNo = regNo;

this.name = name;

this.email = email;

}

public string RegNo

{

get { return regNo; }

}

public string Email

{

get { return email; }

}

public string Name

{

get { return name; }

}

}

Department class code:

public class Department

{

private string code;

private string name;

private List<Student> studentList;

private const int MAX\_NO\_OF\_STUDENTS = 10;

public Department(string code, string name) : this()

{

this.code = code;

this.name = name;

}

public Department()

{

studentList = new List<Student>();

}

public string Code

{

get { return code; }

}

public string Name

{

get { return name; }

}

public string AddStudent(Student studentObj)

{

if (studentList.Count <= MAX\_NO\_OF\_STUDENTS)

{

studentList.Add(studentObj);

return "Student has been saved in dept.";

}

else

{

return "Maximum no of students (" + MAX\_NO\_OF\_STUDENTS + ")

has been saved. So, provided student information is not

saved.";

}

}

public List<Student> GetStudentList()

{

return studentList;

}

}

Find the above project code in *Walkthroughs* folder. Project name: *StudentDeptWalkthrough*.

**Practice 3:** Try the above one in your own way.

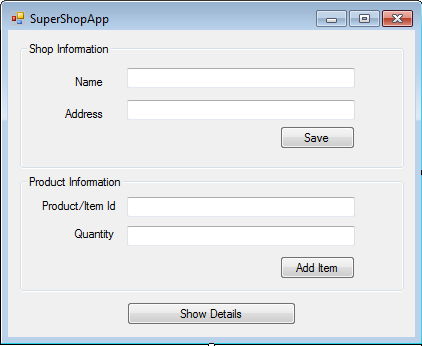
**Practice 4:** Update the code of above example so that Student Registration number will be unique.Try your best and then compare your code with the solution provided.

See the solution in Solution folder. Project name: *StudentDeptSln*

There’s an alternative solution *StudentDept Using Dictionary,* where Dictionary is used instead of List is a prefarable for this problem.

**Practice 5:** Make an application for Super Shop Store:

User will save the shop name and address initially. After that (s)he will save as many product as (s)he wants. If the product Id already exists, quantity will be updated instead of newly added. See UI:

****