**ADF: MS .NET  
Loan Management:   
Faculty Guide**

Loan Management: Faculty Guide

**Activity 1**

1. Go to Start > All Programs > Visual Studio 2010.
2. Create an empty solution ‘**LoanManagement**’.
3. Go to **Solution Explorer**, click on **Add New Solution Folder** icon and name the new folder as **LoanManagement.UI**.
4. Right click on the **LoanManagement.UI** solution folder ->select **Add** -> click on **New Project** option. **Add New Project** dialogue box will appear.
5. Select **Visual C#** from Installed Templates > select **.NET Framework 4** from the dropdown on top > select **Console Application** from the list of templates > name the project **LoanMangement.ConsoleUI** >> Click on **OK**. A **Program.cs** file will be created and opened automatically with **Main(string[] args)** method on it.
6. Inside the **Main** method, create two integer variables *a*, and *b* with default values assigned to them.
7. Develop a code block using the ***if-else*** construct that checks which one is greater and displays the result on the console. See the below code
8. Develop a code block that displays three options *1.Add, 2.Edit, 3.Delete* on the console window and get an input value; using **swith(*[expression]*)** construct a display message on the console window if user selects any of the three given options, otherwise display message as ‘Invalid option’ on the console window.
9. Develop a code block using ***for;*** take an integer value and iterate it through 10 times using for statement and display the value on to the console, for each iteration. See the below code

|  |
| --- |
| #region if else  int a = 0, b = 0;  if (a == b)  {  Console.WriteLine("a and b are equal.");  }  else if (a > b)  {  Console.WriteLine("a is greater than b.");  }  else  {  Console.WriteLine("b is greater than a.");  }  #endregion |
| #region switch  Console.WriteLine("Select any option.\n1. Add\n2. Edit\n3. Delete");  string input = Console.ReadLine();  switch (input)  {  case "1": Console.WriteLine("Add option is selected");  break;  case "2": Console.WriteLine("Edit option is selected");  break;  case "3": Console.WriteLine("Delete option is selected");  break;  default: Console.WriteLine("Invalid option");  break;  }  #endregion |
| #region looping  for (int i = 1; i <= 10; i++)  {  Console.WriteLine("I value is : {0}", i);  }  #endregion |

**Activity 2**

1. Go to the *Solution Explorer* > right click; go to *Add* > click on *New Solution Folder* and name the new solution folder ‘**Data**’.
2. Right click on *Data* folder; go to *Add* > click on *Add New Project*. A dialog box window will be open with a list of templates.
3. Select new class Library project from the list of templates and rename it as **LoanManagemet.Entity**.
4. Create Enum.cs file and add *Enum**type* as**LoanType** which includes *PersonalLoan, HomeLoan* as valuesand **Role** whichincludes*Customer, Employee* and *Manager* as values in the **LoanManagement.Entity** project.

|  |
| --- |
| public enum Role  {  Customer = 0,  Employee = 1,  Manager = 2  }  public enum LoanType  {  HomeLoan = 0,  PersonalLoan = 1  } |

1. Create an array name as ExpArray of type *int* with the size *9* and initialize an array with values(1,2,3,4,5,6,7,8,9) in **LoanManagement.ConsoleUI solution inside the main method of the program.cs file.**
2. Using *for* loop display each element of the array.

|  |
| --- |
| int[] values = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };  Console.WriteLine("Array Values are :");  for (int i = 0; i < values.Length; i++)  {  Console.WriteLine(values[i]);  } |

1. Read a new *int* value from console using *Console.Read* method and Convert it using *Convert.ToInt* method.
2. Declare a new array name as Dump of type *int* with the size *10* and using *CopyTo* method of the array, copy all the 9 elements of the first array to newly created array.
3. Assign previously read *int* value to 10th element of the second (newly created) array, and using *for* loop display the all elements.

|  |
| --- |
| Console.WriteLine("Add new value to an Array - (int):");  int value = Convert.ToInt32(Console.ReadLine());  int[] dump = new int[10];  //Array.Copy(values,dump,values.Length);  values.CopyTo(dump, 0);  dump[9] = value;  Console.WriteLine("Values after adding new value :");  foreach (int i in dump)  {  Console.WriteLine(i);  } |

1. Finally press F5 to run the program.

**Activity 3**

1. Go to the *program.cs* file of ***LoanManagement.ConsoleUI*;** Create new static method ***MethodCall()*** and repeat the steps which are developed in the activity.

|  |
| --- |
| public static void MethodCall()  {  int[] values = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };  Console.WriteLine("Array Values are :");  for (int i = 0; i < values.Length; i++)  {  Console.WriteLine(values[i]);  }  Console.WriteLine("Add new value to an Array - (int):");  int value = Convert.ToInt32(Console.ReadLine());  int[] dump = new int[10];  //Array.Copy(values,dump,values.Length);  values.CopyTo(dump, 0);  dump[9] = value;  Console.WriteLine("Values after adding new value :");  foreach (int i in dump)  {  Console.WriteLine(i);  }  } |

1. Create another method similar to the previous method, ***MethodCallWithParams(string)***, which takes a *string* value as an argument and adds it to another array (dump array) and display it.

|  |
| --- |
| public static void MethodCallWithParams(string newValue = "Default")  {  string[] values = new string[] { "One", "two", "three", "four" };  Console.WriteLine("Values before adding new Value");  foreach (string str in values)  {  Console.WriteLine(str);  }  string[] newValues = new string[5];  values.CopyTo(newValues, 0);  newValues[4] = newValue;  Console.WriteLine("Values after adding new Value");  foreach (string str in newValues)  {  Console.WriteLine(str);  }  } |

1. Inside the ***Main()*** method, call the two methods created in the step1 and step2.
2. Run the program.

|  |
| --- |
| Console.WriteLine("Method call without parameters");  MethodCall();  Console.WriteLine("Method call with parameters");  Console.WriteLine("Read new string");  string newstring = Console.ReadLine();  MethodCallWithParams(newstring); |

1. Modify previously crated method, which is in step2, to default parameterized method.
2. Call the ***MethodCallWithParams(string)*** without passing the required parameter and run the program again.

|  |
| --- |
| Console.WriteLine("Method call with optional parameter");  MethodCallWithParams(); |

**Activity 4**

1. Go to the *Solution Explorer* > right click; go to *Add* > click on *New Solution Folder* and name the new solution folder ‘**Data**’.
2. Right click on ***Data*** folder; go to *Add* > click on ***Add New Project***. A dialog box window will be open with a list of templates.
3. Select new class Library project from the list of templates and name it as **LoanManagemet.Entity**.
4. Add existing classes ***User.cs, Loan.cs, Approval.cs***to the ***LoanManagement.Entity*** project. [Classes contain their respective properties as mentioned in Data model.]
5. Add another existing class file ***common.cs*** file which contains static array variable of types *User*, and *Loan.*
6. Add Reference of **LoanManagement.Entity** in **Console.UI** project.
7. Create static method ***StringFormattedUser(User)*** method which takes User as parameter and displays the properties of *User* class in below formatted manor.

|  |
| --- |
| private static string StringFormattedUser(User user)  {  return string.Format("User details are : \n\nFirst Name : {0}\nLast Name : {1}\nRoleId : {2}\nUser Name : {3}\nPassword : {4}", user.FirstName, user.LastName, user.RoleId, user.UserName, user.Password);  } |

1. Using **for** constuct statement display each user details for **CommonData.users**.

|  |
| --- |
| Console.WriteLine("Display class properties");  foreach(User user in CommonData.users)  {  Console.WriteLine(StringFormattedUser(user));  } |

1. Run the program.

**Activity 5**

1. Go to ***LoanManagement.Entity*** project; create a new class ***EntityBase*** with three properties ***Id*** of type *int*, ***CreateDate*** of type *Nullable<System.DateTime>* and ***UpdateDate*** of type *Nullable<System.DateTime>*.
2. Add a method ***GetId()*** *with* ***int*** *as return type*, which returns *Id* value.

|  |
| --- |
| public class EntityBase  {  public int Id  {  get;  set;  }  public Nullable<System.DateTime> CreateDate  {  get;  set;  }  public Nullable<System.DateTime> UpdateDate  {  get;  set;  }  public virtual int GetId()  {  return this.Id;  }  } |

1. Go to *User* class in *LoanManagement.Entity* project, and inherit it from the ***EntityBase*** class.
2. Change the class signatures of *Loan* and *Approval* classes similarly to User class signature.

|  |
| --- |
| public class User : EntityBase  {}  public class Loan : EntityBase  {}  public class Approval :EntityBase  {} |

1. Remove ID property from User, Loan, Approval classes
2. Go to ***LoanManagement.ConsoleUI*** *project.*
3. Change the **StringFormattedUser** method to include *ID*, *CreateDate* and *UpdateDate* properties from the *EntityBase* class.

|  |
| --- |
| private static string StringFormattedUser(User user)  {  return string.Format("User details are : \n\nID : {0}\nFirst Name : {1}\nLast Name : {2}\nRoleId : {3}\nUser Name : {4}\nPassword : {5}\nCreated Date : {6}\nUpdated Date : {7}", user.Id, user.FirstName, user.LastName, user.RoleId, user.UserName, user.Password, user.CreateDate, user.UpdateDate);  } |

1. Go to the *Main()* method and create new *User* object, ***userobj***, with some values assigned to properties, including *Id*, *CreateDate* and *UpdateDate* properties.

|  |
| --- |
| User userObj = new User();  userObj.Id = 100;  userObj.FirstName = "Srreni";  userObj.LastName = "Rao";  userObj.CreateDate = System.DateTime.Now;  userObj.UpdateDate = System.DateTime.Now; |

1. Using ***userobj*** display all the values along with other properties, on the console window.

|  |
| --- |
| Console.WriteLine(StringFormattedUser(obj));  Console.ReadLine(); |

**Activity 6**

1. Open **EntityBase** class, which was created in the previous activity.
2. Change the **GetId()** method of the **EntityBase** class to **virtual** method and make all the properties virtual.

|  |
| --- |
| Public virtual int GetId()  {  return this.Id;  } |

1. Open the *User* class and Override the **GetId()** method. Open the Loan class and override the GetId().

|  |
| --- |
| public override int GetId()  {  return this.Id;  } |

1. Go to Loanmanagement.ConsoleUI -> move to main() method
2. using previously created **userObj** object invoke **GetId()** method and display user id.

|  |
| --- |
| User userObj = new User();  Console.WriteLine(userObj.GetId()); |

1. Create new **loanObj** of type **Loan** class and initialize all properties with some values and using **loanObj** invoke **GetId()** method and display the loan details.

|  |
| --- |
| Loan loanObj = new Loan();  loanObj.Id = 200;  loanObj.Remarks = "Remarks";  loanObj.IsApproved = false;  loanObj.Description = "dfdfd";  Console.WriteLine(loanObj.GetId());  Console.ReadLine(); |

**Activity 7**

1. Open **EntityBase** class, that was is created in the previous activity.
2. Change the class to an abstract class.

|  |
| --- |
| public abstract class EntityBase  {} |

1. Change the GetId() method to abstract method (remove definition for the GetId() method and add abstract key word and remove virtual keyword).

|  |
| --- |
| public abstract int GetId(); |

1. Open the *User*, Approval and Loan class and Override the **GetId()** method.

|  |
| --- |
| public override int GetId()  {  return this.Id;  } |

1. Go to the **program.cs** file of the ***LoanManagement.ConsoleUI*** project and create a new instance of *User* class called ***userobj*** .
2. Call *GetId()* method of User class object called ***userobj***  and display the value on the console window along with other properties.

|  |
| --- |
| userObj.Id = 300;  Console.WriteLine(userObj.GetId());  Console.ReadLine(); |

**Activity 8**

1. Create a new **Collection.cs** class file in the **LoanManagement.Entity** project, change the class name to **FixupCollection** class, and inherit it from **System.Collections.CollectionBase** base class.
2. Implement ***ClearItems()*** *method with void as return type***,** which removes all the items from the collection, then ***Add(object item)*** method with void as return type, which adds a new object to the collection.

|  |
| --- |
| public void ClearItems()  {  List.Clear();  }  public void Add(object item)  {  if (!this.List.Contains(item))  {  List.Add(item);  }  } |

1. Go to the LoanManagement.ConsoleUI project.
2. Create a reference type variable of FixupCollection as ***approveCollection*** and initialize it using approvalDetails of CommonData class (CommonData.approvalDetails).
3. ***commonData.cs*** file, which contains a list of *Approval* Class-type objects with some values in it.

|  |
| --- |
| public static FixupCollection approvalDetails = new FixupCollection  {  new Approval  {  Id = 1,  Approver = new User(3),  Loan = new Loan(1),  Remarks = "<Remarks>",  DateApproved = System.DateTime.Now,  IsApproved = <true/false>,  CreateDate = System.DateTime.Now,  UpdateDate = System.DateTime.Now  },  new Approval  {  Id = 1,  Approver = new User(3),  Loan = new Loan(2),  Remarks = "<Remarks>",  DateApproved = System.DateTime.Now,  IsApproved = <true/false>,  CreateDate = System.DateTime.Now,  UpdateDate = System.DateTime.Now  },  new Approval  {  Id = 1,  Approver = new User(3),  Loan = new Loan(3),  Remarks = "<Remarks>",  DateApproved = System.DateTime.Now,  IsApproved = <true/false>,  CreateDate = System.DateTime.Now,  UpdateDate = System.DateTime.Now  }  }; |

1. Go to the *program.cs* file in *LoanManagement.ConsoleUI*.
2. Create a reference type variable of FixupCollection as ***approveCollection*** and initialize it using approvalDetails of CommonData class (CommonData.approvalDetails).

|  |
| --- |
| FixupCollection approveCollection = CommonData.approvalDetails;  Console.WriteLine("Collection count before calling Add : {0}",  approveCollection.Count);  approveCollection.Add(new Approval  {  Id = 4,  Approver = new User(3),  Loan = new Loan(3),  Remarks = "Third Loan Approved",  DateApproved = System.DateTime.Now,  IsApproved = false,  CreateDate = System.DateTime.Now,  UpdateDate = System.DateTime.Now  });  Console.WriteLine("Collection count after calling Add : {0}",  approveCollection.Count);  Console.WriteLine("\nApproval Details\n\n");  foreach (Approval approveobj in approveCollection)  {  Console.WriteLine("Approval Id : {0}\nLoan Id : {1}\nApprover :  {2}\nApprove Date : {3:dd/MM/yyyy}\nAmount :  {4:0.00}\n\n\n", approveobj.Id, approveobj.Loan.Id,  approveobj.Approver.Name, approveobj.DateApproved,  approveobj.Loan.Amount);  } |

1. Display the **approveCollection** properties details using *foreach*, in *Main()* method and run the program.

**Activity 9**

1. Open **program.cs** file of **LoanManagement.Console** project.
2. Create a delegate called **UserDelegate(string userName)** which returns a **user** object.

***Note:*** Create delegate object on top of the Program.cs class.

|  |
| --- |
| public delegate User UserDelegate(string userName); |

1. Create a static method called *GetUser(string UserName)* with return type as ***user*** class type which finds a user by *UserName* from the *CommonData.users* collection.

|  |
| --- |
| public static User GetUser(string UserName)  {  User[] userAry = CommonData.users;  int index = 0;  do  {  if (userAry[index].UserName == UserName)  {  return userAry[index];  }  index++;  } while (index < userAry.Length);  return null;  } |

1. Instantiate a new **UserDelegate** type object in main method with *GetUser* method as delegate method. Using **Invoke** method of delegate object, invoke the *GetUser()* method and display the result to console window in a formatted way using *StringFormatteduser()* method, which is created in the earlier activity.

|  |
| --- |
| UserDelegate objDelegate = new UserDelegate(GetUser);  User user = objDelegate.Invoke("sreenivas");  Console.WriteLine(StringFormattedUser(user)); |

1. Finally run the program.

**Activity 10**

1. Go to the solution Explorer and right click on the solution. Select *Add* and select *Add New Solution folder*. Rename the folder to *Controller*.
2. Add a new class library project **LoanManagement.Controller** to Controller folder.
3. Add ***IController.cs*** file from***LoanManagmentFiles*** *folder.*
4. Go to *Data* folder and add a new class library project **LoanManagement.DAL** to the *Data* solution folder, which was created in a previous activity.
5. Create a new file **IDAL.cs** in the Data solution folder and add a new generic *public* interface **IDAL<T>** with the methods, **Create(T obj)** which returns *int*, **GetObject(int Id)** which returns Generic type T, **Update(T obj)** which returns *bool* and finally, **Delete(T obj)** which returns *bool*.

|  |
| --- |
| public interface IDAL<T>  {  int Create(T obj);  T GetObject(int id);  bool Update(T obj);  bool Delete(T obj);  } |

1. Go to *Data* folder and open **FixupCollection.cs** file in **LoanManagement.Entity** project.
2. Add new public generic class **FixupCollection<T>** and inherit it from *System.Collections.ObjectModel.ObservableCollection<T>*. Add two methods: **ClearItems()** which clears all the items in the collection and **InsertItem(int index, T item)** which adds new item to the collection.

|  |
| --- |
| public class FixupCollection<T> : ObservableCollection<T>  {  protected override void ClearItems()  {  new List<T>(this).ForEach(t => Remove(t));  }    protected override void InsertItem(int index, T item)  {  if (!this.Contains(item))  {  base.InsertItem(index, item);  }  }  } |

1. Go to **LoanManagement.ConsoleUI** project *Main()* method.
2. Create new object **usersCollection** of type **FixupCollection<User>**.

|  |
| --- |
| FixupCollection<User> usersCollection = new FixupCollection<User>(); |

1. Using **usersCollection**object add two user objects to the collection using **usersCollection.Add** method and display the **usersCollection** using **StringFormattedUser()** method. Finally run the program.

|  |
| --- |
| usersCollection.Add(new User  {  ID = 5,  FirstName = "FirstName1",  LastName = "LastName1",  UserName = "UserName1",  Password = "Password1",  RoleId = (int)Role.Customer,  CreateDate = System.DateTime.Now,  UpdateDate = System.DateTime.Now  });  usersCollection.Add(new User  {  ID = 6,  FirstName = "FirstName2",  LastName = "LastName2",  UserName = "UserName2",  Password = "Password2",  RoleId = (int)Role.Manager,  CreateDate = System.DateTime.Now,  UpdateDate = System.DateTime.Now  });  foreach (User user in usersCollection)  {  Console.WriteLine(StringFormattedUser(user));  } |

**Activity 11**

1. Go to the **LoanManagement.ConsoleUI** project, program.cs file.
2. Use the same object **usersCollection**, which is created in previous activity, and add some more User type objects to the **usersCollection**.
3. Add **using System.Threading.Tasks** statement at the top of the class.
4. Using foreach statement loop through the Collection. Inside the loop block, create a copy of the current user object (Assign current user object to another local object usr of type User.).
5. Create a Task object (use Task.Factory.StartNew). The Task object contains a delegate parameter which displays the user object properties in the console window using StringFormattedUser method which is created in earlier activity.

|  |
| --- |
| if (approveCollection != null)  {  Console.WriteLine("{0,-15}{1,-15}{2,-20}{3,-22}{4,-15}\n",  "Approval Id", "Loan Id", "Approver", "Approve Date", "Amount");  foreach (Approval approveobj in approveCollection)  {  Approval obj = approveobj;  Task task = Task.Factory.StartNew( () =>  {  Console.WriteLine("{0,-15}{1,-15}{2,-20}{3,-22}{4,-15}\n\n\n",  obj.Id, approveobj.Loan.Id,  obj.Approver.Name, approveobj.DateApproved,  obj.Loan.Amount);  });  }  } |

1. The code block might look similar below and run the program.

**Activity12**

1. Go to the **LoanManagement.ConsoleUI** project, **program.cs** file.
2. Create a string that contains the file path and check whether or not a file with the specified name exists in the specified path using the **File.Exists(string)** method.
3. If file does not exist, then create a text file with the specified path.
4. Create a generic **FixupCollection<Loan>** object and initialize it with some loan objects using **CommonData.loans**.
5. Add **using System.IO** statement on top of the class.
6. Loop through the Collection. Inside the loop, using **TextWriter** write the loan object properties values on to the file. And dispose the **TextWriter** object.
7. Using **TextReader** object read the file data till end and display it on to the console window. And dispose **TextReader** object. The code block might look similar to below.

|  |
| --- |
| string filepath = @"C:\sample.txt";  if (!File.Exists(filepath))  {  /\*Create and close the file\*/  File.Create(filepath).Close();  }  Console.WriteLine("Writing data on to the file\n");  using (TextWriter tWriter = new StreamWriter(filepath))  {  foreach (var loan in CommonData.loans)  {  tWriter.WriteLine("Order ID : " + loan.Id);  tWriter.WriteLine("Category : " + loan.Category);  tWriter.WriteLine("Status : " + loan.IsApproved);  tWriter.WriteLine("Amount : " + loan.Amount);  tWriter.WriteLine("Assigned To : " + loan.AssignedTo.Id);  tWriter.WriteLine("Description : " + loan.Description);  tWriter.WriteLine("Order User ID : " + loan.User.Id);  tWriter.WriteLine("Created Date : " + loan.CreateDate);  tWriter.WriteLine("Updated Date : " + loan.UpdateDate);  }  }  Console.WriteLine("Reading data from file\n");  using (TextReader tReader = new StreamReader(filepath))  {  Console.Write(tReader.ReadToEnd());  } |

1. Run the program.

**Activity 13**

1. Add **ExtendedMethodsHelper.cs** file to **LoanManagement.Entity** from **LoanManagementFiles** folder.
2. Go to **LoanManagement.DAL** project and add **LoanManagement.Entity** reference to the current DAL project.
3. Add an existing files **SqlHelper.cs**, **DALBase.cs**, **UserDAL.cs** and **LoanDAL.cs** from **LoanManagementFiles** folder.
4. Change the attribute **DbStringName** to your database connection string.
5. Implement another file for Loan Approval as **ApprovalDAL.cs** file which implements **IDAL** interface methods, which is similar to **LoanDAL** class. The class code might be similar to below.

|  |
| --- |
| public class LoanDAL : DALBase, IDAL<Loan>  {  #region Constructor(s)  public LoanDAL()  : base()  {  }  #endregion  #region public method(s)  public int Create(Loan obj)  {  SqlTransaction trans = null;  SqlParameter parCategoryId = new SqlParameter("CategoryId", (int)obj.Category);  SqlParameter parAmount = new SqlParameter("Amount", obj.Amount);  SqlParameter parUserId = new SqlParameter("UserId", obj.User.Id);  SqlParameter parAssignedTo = new SqlParameter("AssignedTo", obj.AssignedTo.Id);  SqlParameter parDescription = new SqlParameter("Description", obj.Description);  SqlParameter parIsApproved = new SqlParameter("IsApproved", obj.IsApproved);  try  {  Connection.Open();  trans = Connection.BeginTransaction();  SqlHelper.ExecuteNonQuery(trans, CommandType.StoredProcedure, "usp\_LoanInsert", parCategoryId, parAmount, parUserId,  parAssignedTo, parDescription, parIsApproved);  trans.Commit();  return GetCreatedObjectId("Loan", "LoanId");  }  catch (Exception ex)  {  throw ex;  }  finally  {  if (Connection.State == ConnectionState.Open)  {  Connection.Close();  }  }  }  public Loan GetObject(int id)  {  Loan loan = null;  SqlParameter parLoanId = new SqlParameter("LoanId", id);  Connection.Open();  DataTable tblResult = SqlHelper.ExecuteDataset(Connection, CommandType.StoredProcedure, "usp\_SelectLoanById", parLoanId).Tables[0];  Connection.Close();  if (tblResult.Rows.Count > 0)  {  loan = GetObjectFromDataRow(tblResult.Rows[0]);  }  return loan;  }  public bool Update(Loan obj)  {  SqlTransaction trans = null;  SqlParameter parLoanId = new SqlParameter("LoanId", obj.Id);  SqlParameter parCategoryId = new SqlParameter("CategoryId", (int)obj.Category);  SqlParameter parAmount = new SqlParameter("Amount", obj.Amount);  SqlParameter parUserId = new SqlParameter("UserId", obj.User.Id);  SqlParameter parAssignedTo = new SqlParameter("AssignedTo", obj.AssignedTo.Id);  SqlParameter parDescription = new SqlParameter("Description", obj.Description);  SqlParameter parIsApproved = new SqlParameter("IsApproved", obj.IsApproved);  try  {  Connection.Open();  trans = Connection.BeginTransaction();  SqlHelper.ExecuteNonQuery(trans, CommandType.StoredProcedure, "usp\_LoanUpdate", parLoanId, parCategoryId, parAmount, parUserId,  parAssignedTo, parDescription, parIsApproved);  trans.Commit();  return true;  }  catch (Exception ex)  {  throw ex;  }  finally  {  if (Connection.State == ConnectionState.Open)  {  Connection.Close();  }  }  }  public bool Delete(Loan obj)  {  SqlTransaction trans = null;  SqlParameter parLoanId = new SqlParameter("LoanId", obj.Id);  try  {  Connection.Open();  trans = Connection.BeginTransaction();  SqlHelper.ExecuteNonQuery(trans, CommandType.StoredProcedure, "usp\_LoanDelete", parLoanId);  trans.Commit();  Connection.Close();  return true;  }  catch (Exception ex)  {  trans.Rollback();  throw ex;  }  finally  {  if (Connection.State == ConnectionState.Open)  {  Connection.Close();  }  }  }  public FixupCollection<Loan> GetObjectsByEmployeeId(int assignedEmployeeId)  {  FixupCollection<Loan> objLoanCollection = new FixupCollection<Loan>();  SqlParameter parEmployeeId = new SqlParameter("AssignedTo", assignedEmployeeId);  Connection.Open();  DataTable tblResult = SqlHelper.ExecuteDataset(Connection, CommandType.StoredProcedure, "dbo.usp\_SelectLoansAssigned", parEmployeeId).Tables[0];  Connection.Close();  foreach (DataRow objLoanRow in tblResult.Rows)  {  objLoanCollection.Add(GetObjectFromDataRow(objLoanRow));  }  return objLoanCollection;  }  #endregion  #region Private Method(s)  private Loan GetObjectFromDataRow(DataRow resultRow)  {  Loan loan = new Loan();  loan.Id = Convert.ToInt32(resultRow["LoanId"]);  loan.Amount = Convert.ToInt32(resultRow["Amount"]);  loan.AssignedTo = new UserDAL().GetUser(Convert.ToInt32(resultRow["AssignedTo"]));  loan.Category = (LoanType) Enum.Parse(typeof(LoanType), resultRow["CategoryId"].ToString());  loan.Description = Convert.ToString(resultRow["Description"]);  loan.IsApproved = Convert.ToBoolean(resultRow["IsApproved"]);  loan.User = new UserDAL().GetUser(Convert.ToInt32(resultRow["UserId"]));  loan.CreateDate = Convert.ToDateTime(resultRow["CreateDate"]);  loan.UpdateDate = Convert.ToDateTime(resultRow["UpdateDate"]);  return loan;  }  #endregion  } |

1. Go to **LoanManagement.Controller** project and add **LoanManagement.Entity**, **LoanManagement.DAL** references to the current Controller project.
2. Add existing files **UserController.cs** and **LoanController.cs** files to the project from **LoanManagementFiles** folder.
3. Implement similar classes like **LoanController** and **ApprovalController** which implements **IController** interface methods.

|  |
| --- |
| public class LoanController : IController<Loan>  {  #region Private Properties  private LoanDAL \_loanDAL = null;  #endregion  #region Constructor(s)    public LoanController()  {  this.\_loanDAL = new LoanDAL();  }  #endregion  #region Public Method(s)    public int Create(Loan obj)  {  return this.\_loanDAL.Create(obj);  }  public Loan GetObject(int id)  {  return this.\_loanDAL.GetObject(id);  }  public Loan GetObject(string objName)  {  return this.\_loanDAL.GetObject(int.Parse(objName));  }  public bool Update(Loan obj)  {  return this.\_loanDAL.Update(obj);  }  public bool Delete(Loan obj)  {  return this.\_loanDAL.Delete(obj);  }  public FixupCollection<Loan> GetLoansByEmployeeId(int employeeId)  {  return this.\_loanDAL.GetObjectsByEmployeeId(employeeId);  }  public FixupCollection<Loan> GetAllLoansAssignedToEmployee(int employeeId)  {  return this.\_loanDAL.GetObjectsByEmployeeId(employeeId);  }  public Loan[] GetLoansByEmployeeWCF(int employeeId)  {  EmployeeLoansServiceReference.LoanManagementServiceClient loanServiceClient = new EmployeeLoansServiceReference.LoanManagementServiceClient();  try  {  return loanServiceClient.GetLoansByEmployee(employeeId).LoansByEmployee;  }  catch (Exception ex)  {  throw ex;  }  finally  {  loanServiceClient.Close();  }  }  #endregion  } |

|  |
| --- |
| public class ApprovalController : IController<Approval>  {  #region Private Properties  /// <summary>  /// Loacal variable to hold the ApprovalDAl object  /// </summary>  private ApprovalDAL \_approvalDAL = null;  private string APPROVE\_LOG\_PATH;  #endregion  #region Constructor(s)  public ApprovalController()  {  this.\_approvalDAL = new ApprovalDAL();  }  #endregion  #region Public Method(s)  public int Create(Approval obj)  {  return this.\_approvalDAL.Create(obj);  }  public Approval GetObject(int id)  {  return this.\_approvalDAL.GetObject(id);  }  public Approval GetObject(string objName)  {  return this.\_approvalDAL.GetObject(int.Parse(objName));  }  public bool Update(Approval obj)  {  return this.\_approvalDAL.Update(obj);  }  public bool Delete(Approval obj)  {  return this.\_approvalDAL.Delete(obj);  }  #endregion  } |

1. Go to **LoanManagement.ConsoleUI** project **Main()** method.
2. Add **LoanManagement.Controller** namespace at the top.
3. Create new variable **objUser** of type **UserController**.
4. Using objUser invoke Create(user) method with user as parameter for inserting record. The code block looks like similar to below.

|  |
| --- |
| UserController objUser = new UserController();  Console.WriteLine("Insert new User into DB: ");  int newUserID = objUser.Create(new User { Id = 9, FirstName = "FirstName9",  LastName = "LastName9", UserName = "TestUser9",  Password = "Password", UserTypeId = (int)UserType.Customer,  CreateDate = System.DateTime.Now,  UpdateDate = System.DateTime.Now });  Console.WriteLine("New User ID: " + newUserID.ToString());  Console.WriteLine("------------------------------------------------------");  Console.ReadKey(); |

1. Using objUser invoke GetObject() with newUserId as parameter and assign return value to another User object named usrFromDB and display specific record. The code block looks like similar to below.

|  |
| --- |
| Console.WriteLine("Get User details by ID: ");  User usrFromDB = objUser.GetObject(newUserID);  if (usrFromDB != null)  {  Console.WriteLine(StringFormattedUser(usrFromDB));  }  Console.WriteLine("------------------------------------------------------");  Console.ReadKey(); |

1. Using objUser invoke Update(User) method with user object as parameter for updating record and display updated user details by repeating step13. The code block looks similar to below.

|  |
| --- |
| Console.WriteLine("Update User details by ID:");  usrFromDB.FirstName = "UpdateedFirstName";  usrFromDB.LastName = "UpdatedLastName";  bool isUpdated = objUser.Update(usrFromDB);  if (isUpdated)  Console.WriteLine("User details updated");  Console.WriteLine("------------------------------------------------------");  Console.ReadKey(); |

1. Using objUser invoke Delete(User) method with user as parameter for deleting record. The code block looks similar to below.

|  |
| --- |
| Console.WriteLine("Delete User Details from DB");  bool isDeleted = objUser.Delete(usrFromDB);  if (isDeleted)  {  Console.WriteLine("User deleted successfull");  }  Console.WriteLine("------------------------------------------------------");  Console.ReadKey(); |

1. Using objUser invoke GetObjects() method and displaying multiple records. The code block looks like similar to below.

|  |
| --- |
| FixupCollection<User> users = objUser.GetObjects((int)UserType.Employee);  Console.WriteLine("List of users in DB: ");  foreach (var usr in users)  {  Console.WriteLine(StringFormattedUser(usr));  } |

1. Finally execute the program.

**Activity14**

1. Go to **OrderManagement.Controller** project and add new file called **CustomerController.cs**.
2. Go to **OrderManagement .Entity** project and add another file from Add->AddNewItem->Data and **LINQtoSQL** Classes template and name it as **OrderManagement.dbml**.
3. Add Customer table **OrderManagement** database from ServerExplorer window into **OrderManagement.dbml** file.
4. Move to **CustomerController** class from **OrderManagement.Controller** project and create instance for **OrderManagementDataContext**.
5. In CustomeController class develop **Create(name,address,city,state,remarks)** method with *int* as return type for inserting customer record.

|  |
| --- |
| public int Create(string name, string address, string city, string state, string comments)  {  try  {  Customer c = new Customer { Name = name, Address = address,  City = city, State = state, Comments = comments };  db.Customers.InsertOnSubmit(c);  db.SubmitChanges();  return c.ID;  }  catch (Exception)  {  return 0;  }  } |

1. Develop another method Update(id, name,address,city,state,remarks) and bool as return type for updating the record.

|  |
| --- |
| public bool Update(int Id, string name, string address,  string city, string state, string comments)  {  try  {  Customer c = GetCustomer(Id);  if (c == null)  {  return false;  }  c.ID = Id;  c.Name = name;  c.Address = address;  c.City = city;  c.State = state;  c.Comments = comments;    db.SubmitChanges();  return true;  }  catch (Exception)  {  return false;  }  } |

1. Develop another method Delete(id) and bool as return type for deleting the record.

|  |
| --- |
| public bool Delete(int id)  {  Customer c = GetCustomer(id);  if (null == c)  {  return false;  }  db.Customers.DeleteOnSubmit(c);  db.SubmitChanges();  return true;  } |

1. Develop another method GetCustomer(Id) and Customer as return type for displaying specific record based on Customer id.

|  |
| --- |
| public Customer GetCustomer(int Id)  {  Customer cust = null;  var customers = from c in db.Customers  where c.ID == Id  select c;  if(customers != null)  cust = customers.FirstOrDefault<Customer>();  return cust;  } |

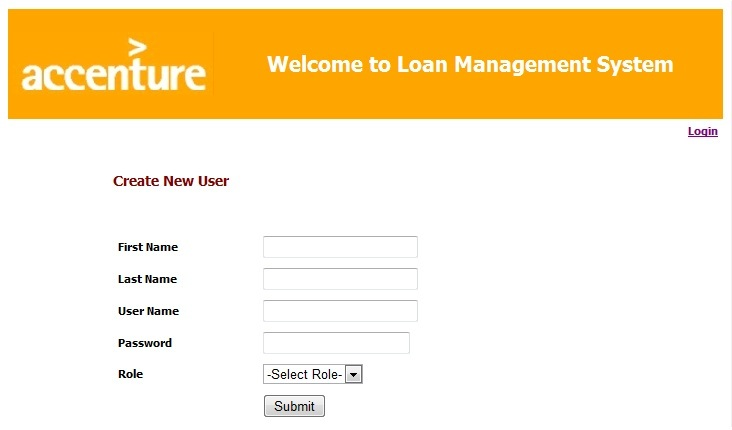
1. Develop another method called GetCustomers() with return type as IEnumarable<Customer> for displaying all records of customer.

|  |
| --- |
| public IEnumerable<Customer> GetCustomers()  {  return from c in db.Customers select c;  } |

1. Move to OrderManagement.ConsoleUI project and add reference to OrderManagement.Controller namespace in OrderManagement.ConsoleUI project.
2. Move to Main() method and call Create(),Update() and Delete() methods and execute the program.

**Activity15**

1. Go to the UI solution folder and create a new ***ASP.NET Web Application*** project called ***LoanManagement.WebUI***. (Select ASP.NET Empty Web Application from Add New Project Window)
2. Add **LoanManagement.Controller** and **LoanManagement.Entity** references to the current project.
3. Add a web page, ***UserCreate.aspx***. Design the page in such a way that it takes first name, last name, User name, password, and role as inputs to create a new user. [Note: FirstName, LastName, Username, and Passwords are Textboxes. Role is a dropdown control].
4. Add a new button control, ***CreateUser*** to the page.



1. Open the ***UserCreate.aspx.cs*** page and import the ***LoanManagement.Controller*** and ***LoanManagement.Entity*** namespaces.
2. Bind the Role dropdown with list of roles that are mentioned in Enum type Role on Page\_Load event. The code looks like as

|  |
| --- |
| if (!IsPostBack)  {  ddlRole.DataSource = Enum.GetNames(typeof(Role));  ddlRole.DataBind();  } |

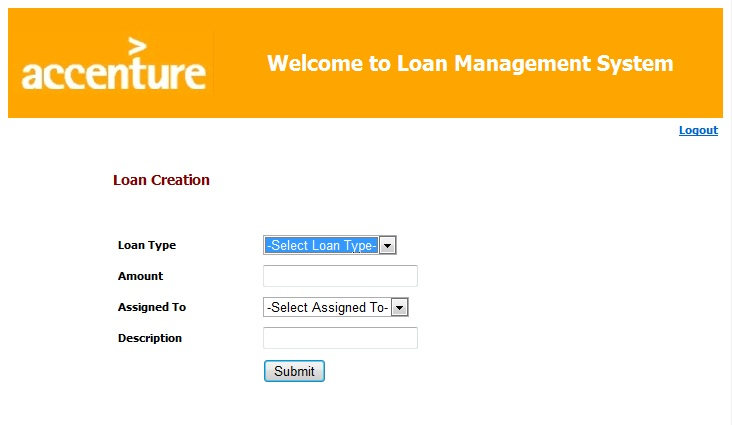
1. Implement the **CreateUser\_Click** event of UserCreate.aspx that creates new user with input values from the above input screen. The code similar to following:
2. Inside the event implementation, create a new ***UserController*** instance. Using that instance, call the ***Create()*** method to create a new user.

|  |
| --- |
| int userId;  /\*Dynamic type created here\*/  dynamic objUserController = new UserController();  User usr = new User  {  FirstName = txtFirstName.Text.Trim(),  LastName = txtLastName.Text.Trim(),  UserName = txtUserName.Text.Trim(),  Password = txtPassword.Text.Trim(),  RoleId = (int)((Role)Enum.Parse(typeof(Role),  ddlRole.SelectedValue))  };  try  {  /\* Create a User, display the reference# \*/  userId = objUserController.Create(usr);  /\* Named parameter feature is used here \*/  lblError.Text = usr.Message(referenceID : userId);  }  catch  {  /\* On exception show the error message \*/  lblError.Text = "Error in User Creation. Please try again.";  }  finally  {  /\*Reset the form fields\*/  txtUserName.Text = string.Empty;  txtFirstName.Text = string.Empty;  txtLastName.Text = string.Empty;  txtPassword.Text = string.Empty;  ddlRole.SelectedValue = "0";  } |

1. Add existing **Login.aspx, Login.aspx.cs, *welcome.aspx, welcome.aspx.cs,*** **LoanManagement.Master** and **LoanManagement.Master.cs** files from **LoanManagementFiles** folder.
2. Open the ***Login.aspx.cs*** page and import the ***LoanManagement.Controller*** and ***LoanManagement.Entity*** namespaces.
3. Implement ***Login\_Click*** event that validates the user credentials.
4. Inside the button click event, create a new ***UserController*** instance. Pass the Username to the ***GetObject()*** method of ***UserController*** and gets the User details.

|  |
| --- |
| string strError = "Invalid User Name / Password";  UserController objUserController = new UserController();  User objUser = objUserController.GetObject(txtUserName.Text.Trim());  if (objUser == null)  {  /\* If no user exists show message \*/  lblError.Text = strError;  return;  }  else if (objUser.RoleId != (int)Role.Customer)  {  /\* If User role is not Employee or Manager \*/  lblError.Text = strError;  return;  }  else if (objUser.Password != txtPassword.Text.Trim())  {  /\* If password is incorrect \*/  lblError.Text = "Invalid Password";  return;  }  Session["objUser"] = objUser;  Response.RedirectPermanent("Welcome.aspx"); |

1. Check if the User has permission to access the content. If not, display the error message on the page. Otherwise, redirect the page to the ***welcome.aspx*** page.
2. Add another content page ***LoanCreate.aspx*** for the **LoanManagementMaster.Master** that requests new loan details and inserts new loan request details into the database.
3. Design the page with two dropdowns - Loan Type, and AssignedTo in addition to two Textboxes for Amount, and Description
4. Populate the loan type and AssignedTo dropdowns with *Enum* ***LoanType*** and from User DB table where the role of user is Employee.
5. Add a new button control to the aspx page. Open ***LoanCreate.aspx.cs*** and implement the button click event. The screen looks similar to below.



1. Implement the **CreateLoanRequest\_Click** event to create a new loan request.
2. Inside the button click implementation, create ***LoanController*** instance and call the ***Create()*** method of the controller by passing input values given by the user.

|  |
| --- |
| int loanId;  User currentUser = (User) Session["objUser"];  LoanController objLoanController = new LoanController();    /\* Create a loan object with default values set \*/  Loan objLoan = new Loan  {  Amount = Convert.ToDouble(txtAmount.Text.Trim()),  AssignedTo = new User(Convert.ToInt32(ddlAssignedTo.SelectedValue)),  Category = (LoanType) Enum.Parse(typeof(LoanType), ddlLoanType.SelectedValue),  Description = txtDescription.Text.Trim(),  IsApproved = false,  User = currentUser  };  try  {  /\* Create a loan request and display the reference# \*/  loanId = objLoanController.Create(objLoan);  lblError.Text = "Loan request created successfully. Reference No: " +  loanId.ToString();  }  catch  {  /\* On exception show the error message \*/  lblError.Text = "Error in Loan Creation. Please try again.";  }  finally  {  /\*Reset the form fields\*/  ddlLoanType.SelectedValue = "0";  txtAmount.Text = string.Empty;  ddlAssignedTo.SelectedValue = "0";  txtDescription.Text = string.Empty;  } |

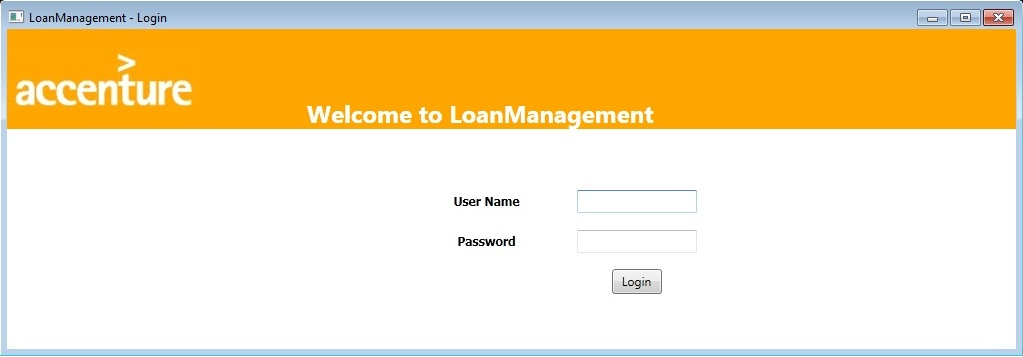
1. Right click on **LoanManagement.WebUI** project and click on ‘Set as Startup Project’ and run the application.

**Activity 16**

1. Go to the ***UI*** solution folder and add the new WPF project ***LoanManagement.WindowsUI*.** (Select WPF Application from Add New Project Wizard) After you add the new project, two ‘*.xaml*’ files, ***App.xaml*** and ***MainWindow.xaml***, will be created automatically.
2. Add references to the ***LoanManagement.Controller*** and ***LoanManagement.Entity*** projects.
3. Open the ***App.xaml*** file and add a new property ***LoggedinUser*** of type User to hold the current logged in user data.
4. Implement the ***Application\_Exit*** method which closes all the windows upon application close.
5. Add **Resource Dictionary** file, **style.xaml** from **LoanManagementFiles** folder
6. Open **MainWindow.xaml** and add a window resource file using the **Window.Resources** tag.

|  |
| --- |
| <Window x:Class="LoanManagement.WindowsUI.MainWindow"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  Title="LoanManagement - Login" Height="Auto" Width="Auto"  WindowState="Maximized" Loaded="Window\_Loaded">  ***<Window.Resources>***  ***<ResourceDictionary Source="Style.xaml"/>***  ***</Window.Resources>***  //rest of code goes here.  </Window> |

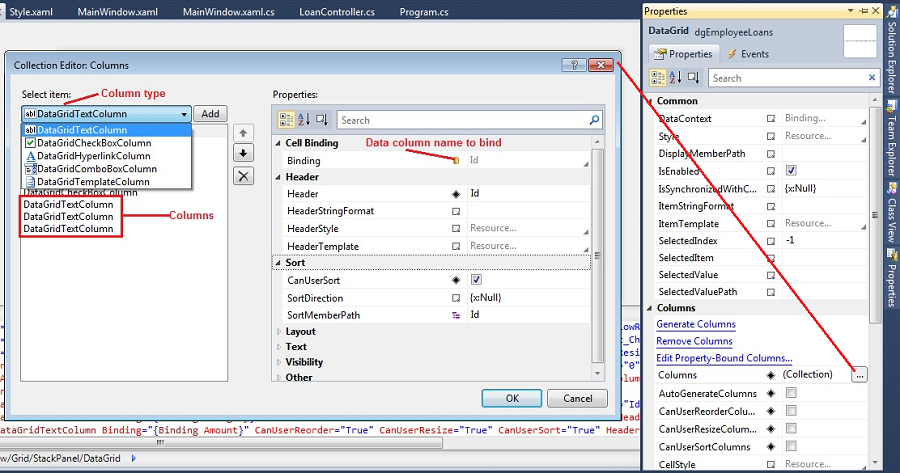
1. Drag and drop a ***Grid Control*** that has four rows and two columns, on to the designer area. Drag and drop two ***TextBlock*** and ***TextBox*** controls and a ***Button*** control on to the grid from the toolbox which will allow the user to provide their user name and password as inputs for login. The screen looks like similar to below.



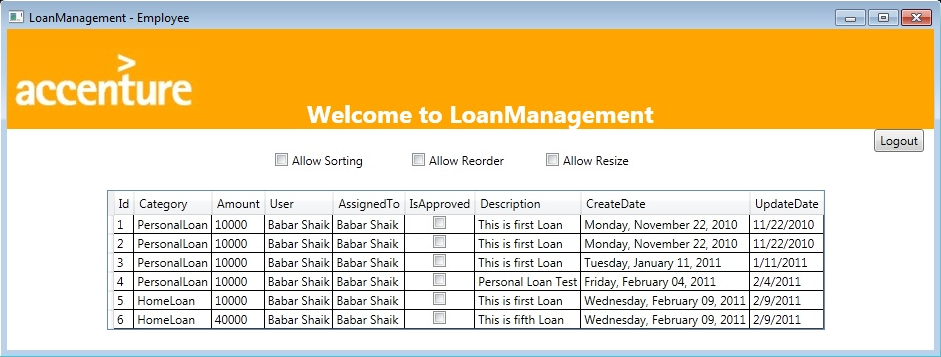
1. Implement the login button click event to handle the user login.

|  |
| --- |
| private void Login\_Click(object sender, RoutedEventArgs e)  {  string strError = "Invalid User Name / Password";  UserController objUserController = new UserController();  User objUser = objUserController.GetObject(UserName.Text.Trim());  if (objUser == null)  {  /\* If no user exists \*/  MessageBox.Show(strError);  return;  }  else if (objUser.RoleId == (int)Role.Customer)  {  /\* If user role is invalid \*/  MessageBox.Show(strError);  UserName.Focus();  return;  }  else if (objUser.Password != Password.Password.Trim())  {  /\* If password is incorrect \*/  MessageBox.Show("Invalid Password");  Password.Focus();  return;  }    /\* Set the current logged in user to Applications' LoggedinUser property and hide  the current window \*/  App.LoggedinUser = objUser;  this.Hide();  /\* Redirect the Loggedin User to respective role page \*/  if (objUser.RoleId == (int) Role.Employee)  {  Employee objEmployeeWindow = new Employee();  objEmployeeWindow.Show();  }  else if (objUser.RoleId == (int) Role.Manager)  {  Manager objManagerWin = new Manager();  objManagerWin.Show();  }  } |

1. Add a new file ***Employee.xaml***, to the current project, and then add ***Style.xaml*** resource reference in the current file. Add grid control with 4 rows and a column, and then set the height and width of the grid rows appropriately.
2. Inside the grid, add another grid with one row and 2 columns which include a design window header with logo and header text, if any.
3. Drag and drop a ***StackPanel*** control onto the second row of the outer grid. Inside ***StackPanel***, add a ***Button*** control to implement logout functionality then add three ***CheckBox*** controls to the third row of the outer grid, to allow the user to sort, resize, and reorder the *DataGrid*.
4. Drag and drop a ***StackPanel*** control on to the page, then inside ***StackPanel***, add ***DataGrid*** control. Inside the ***DataGrid*** control, add ***DataGridTextColumn*** to display the data. The properties screen looks below.



The screen might look like below



1. Go to **LoanManagement.Entity** project, and add new readonly property **Name** which gives Concatenated value of **FistName** and **LastName**.
2. Go to **LoanManagement.Entity** project, and add new property Remarks which is a temporary support property on loan approval process.
3. Go to **LoanManagement.WindowsUI** project **Employee.Xaml** page, and on load event, call the ***GetLoansByEmployeeId (currentLoggedInEmployeeID)*** method of the **LoanController** class.

|  |
| --- |
| private void Window\_Loaded(object sender, RoutedEventArgs e)  {  /\* Load DataGird with Loans assgined to current loggedin user \*/  dgEmployeeLoans.ItemsSource = new  LoanController().GetLoansByEmployeeId(App.LoggedinUser.Id);  } |

1. Allow the user to sort, resize, and reorder the data grid columns by setting the respective properties of the columns to true. If the respective check box is checked, implement their checked event. The code blocks look similar to below.

|  |
| --- |
| private void AllowResize\_Checked(object sender, RoutedEventArgs e)  {  dgEmployeeLoans.CanUserResizeColumns = Convert.ToBoolean(AllowResize.IsChecked);  }    private void AllowReorder\_Checked(object sender, RoutedEventArgs e)  {  dgEmployeeLoans.CanUserReorderColumns =  Convert.ToBoolean(AllowReorder.IsChecked);  }  private void AllowSort\_Checked(object sender, RoutedEventArgs e)  {  dgEmployeeLoans.CanUserSortColumns = Convert.ToBoolean(AllowSort.IsChecked);  } |

1. Implement the logout functionality. Set the **LoggedInUser** property to *null*. And finally run the program. The code block might look similar to below.

|  |
| --- |
| private void Logout\_Click(object sender, RoutedEventArgs e)  {  App.LoggedinUser = null;  MainWindow loginWin = new MainWindow();  loginWin.Show();  this.Close();  } |

**Activity 17**

1. Go to Solution Explorer and create a new solution folder called ***Business***.
2. Add a new WCF Service Application from the list of templates available in the *Add New Project* window and name the project **LoanManagement.Service** with C# as the coding language, then click on *OK*.
3. Add **LoanManagement.DAL** and **LoanManagement.Entity** references to the current project.
4. Create a new WCF service file **ILoanManagementService.cs** and add it to the current project. Inside the file, implement an interface, **ILoanManagement** with a method ***GetLoansByEmployee(int employeeId)***. This returns the ***EmployeeLoan*** object, which is nothing but a **DataContract** object. Add the **ServiceContract** attribute to the interface, and then add the **OperationContract** attribute to the method defined in that interface. The code block looks similar to below.

|  |
| --- |
| using **System.Runtime.Serialization;**  using System.ServiceModel;  using LoanManagement.Entity;  namespace LoanManagement.Service  {  [ServiceContract]  public interface ILoanManagementService  {  [OperationContract]  EmployeeLoan GetLoansByEmployee(int employeeId);  }  } |

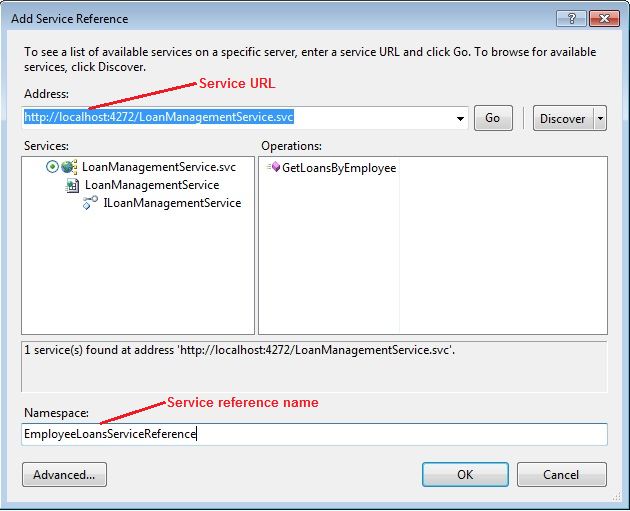
1. Add new class, ***EmployeeLoan*** to the same file which contains the property, ***LoansByEmployee***, to hold the list of loans assigned to an employee, as a *FixupCollection<Loan>* object. Add **DataContract** attribute to the *EmployeeLoan* class, then add ***DataMember*** attribute to the property. The code block looks similar to below.

|  |
| --- |
| [DataContract]  public class EmployeeLoan  {  private FixupCollection<Loan> loans;  [DataMember]  public FixupCollection<Loan> LoansByEmployee  {  get { return this.loans; }  set { this.loans = value; }  }  } |

1. Open **LoanManagementService.svc.cs** file and inherit it from **ILoanMangementService** interface and implement all the methods of the interface.
2. Import the LoanManagement.DAL and LoanManagement.Entity namespaces to the file.
3. Inside the method implementation, create a new ***LoanDAL*** instance. With this new instance, call the ***GetObjectsByEmployeeId(int)*** method for a given *employeeId* and assign it to the ***LoanByEmployee*** property of the **DataContract** class. The code block might similar to below.

|  |
| --- |
| using LoanManagement.DAL;  namespace LoanManagement.Service  {  public class LoanManagementService : ILoanManagementService  {  public EmployeeLoan GetLoansByEmployee(int employeeId)  {  return new EmployeeLoan  {  LoansByEmployee = new LoanDAL().GetObjectsByEmployeeId(employeeId)  };  }  }  } |

1. To Test the service, go to the **Solution Explorer,** then to the **Controller** folder. Right click on the **LoanManagement.Controller** project, and then select ***Add Service Referece***. A popup window will be opened as show below.

:

1. Enter the valid address of the service URL then enter the service reference name (i.e. Namespace). Finally, click *OK* to add the service reference.
2. Open the **LoanController.cs** file that was created in a previous activity within the LoanManagement.Controller project. Implement a method, **GetLoansByEmployeeWCF** that calls the service and gets a list of loans assigned to an employee and returns the same.

|  |
| --- |
| public Loan[] GetLoansByEmployeeWCF(int employeeId)  {  EmployeeLoansServiceReference.LoanManagementServiceClient loanServiceClient =  new EmployeeLoansServiceReference.LoanManagementServiceClient();  try  {  return loanServiceClient.GetLoansByEmployee(employeeId).LoansByEmployee;  }  catch (Exception ex)  {  throw ex;  }  finally  {  loanServiceClient.Close();  }  } |

1. Go to the **UI** solution folder and right click on **LoanManagement.ConsoleUI**. Add a **reference** to the **LoanMamangement.Controller** then open the **Program.cs** file of the **LoanManagement.ConsoleUI** project. Finally, add a new region block which implements the logic to call the WCF service. [Note: Copy and paste the same app.config file that was created when adding the reference of service to the LoanManagement.Controller project.] The code block might similar to below.

|  |
| --- |
| Loan[] loans = null;  LoanController objLoanController = new LoanController();  try  {  loans = objLoanController.GetLoansByEmployeeWCF(2);  Console.WriteLine("{0,-8}{1,-15}{2,-10:0.00}{3,-40}{4,-40},{5,-10},{6,-14:dd/MM/yyyy},{7,-14:dd/MM/yyyy}\n",  "Loan Id", "Category", "Amount", "User Name", "Assigned To", "Approved", "Created Date", "Updated Date");  foreach (Loan objLoan in loans)  {  Console.WriteLine("\n{0,-8}{1,-15}{2,-10:0.00}{3,-40}{4,-40},{5,-10},{6,-14:dd/MM/yyyy},{7,-14:dd/MM/yyyy}",  objLoan.Id, objLoan.Category, objLoan.Amount, objLoan.User.Name,  objLoan.AssignedTo.Name, objLoan.IsApproved.ToString(),  objLoan.CreateDate, objLoan.UpdateDate);  }  }  catch (Exception ex)  {  Console.WriteLine(ex.Message);  } |

1. Finally run the ConsoleUI program.

**Activity 18**

1. Go to the **LoanManagement.WindowUI** project and add a new Windows WPF file, ***Manager.xaml*** to the current project. Add the ***Style.xaml*** resource reference in the current file, and then add Grid Control with four rows and one column. Finally, set the grid row height and width appropriately, and add page header images, if any, as well as header text.
2. Add **StackPanel** with two button controls, *View Log* and *Logout*, inside the second row of the outer grid. Add another ***StackPanel*** with ***TextBlock*** and ***ComboBox*** controls in it.

|  |
| --- |
| <StackPanel VerticalAlignment="Center" Grid.Row="2" HorizontalAlignment="Left"  Margin="100,0,0,0" Grid.Column="0">  <TextBlock Name="EmployeeLabel" Text="Select Employee"/>  <ComboBox Name="EmployeesCombo" DisplayMemberPath="Name" SelectedValuePath="Id"  SelectionChanged="EmployeesCombo\_SelectionChanged"  SelectedValue="{Binding User.Name}">  </ComboBox>  </StackPanel> |

1. Add another ***StackPanel*** to the fourth row of the outer grid and name it *GridLayout*. Inside the *GridLayout*, add ***DataGrid*** with the necessary columns and the *Button* control *btnApprove*.

|  |
| --- |
| <StackPanel x:Name="GridLayout" Visibility="Hidden" VerticalAlignment="Top"  HorizontalAlignment="Left" Grid.Row="3" Margin="100,50,20,20"  Grid.Column="0">  <DataGrid AutoGenerateColumns="False" Height="Auto" Name="dgEmployeeLoans"  Width="Auto" CanUserDeleteRows="True" CanUserAddRows="False"  CanUserResizeColumns="True" CanUserReorderColumns="True">  <DataGrid.Columns>  <DataGridTextColumn Binding="{Binding Id}" CanUserReorder="False"  CanUserResize="False" CanUserSort="True" Header="Order Id"  IsReadOnly="True" />  <DataGridTextColumn Binding="{Binding Category}" CanUserReorder="False"  CanUserResize="False" CanUserSort="True" Header="Category"  IsReadOnly="True" />  <DataGridTextColumn Binding="{Binding Amount}" CanUserReorder="False"  CanUserResize="False" CanUserSort="True" Header="Amount"  IsReadOnly="True" />  <DataGridTextColumn Binding="{Binding User.Name}"  CanUserReorder="False" CanUserResize="False"  CanUserSort="True" Header="User" IsReadOnly="True" />  <DataGridTextColumn Binding="{Binding AssignedTo.Name}"  CanUserReorder="False" CanUserResize="False" CanUserSort="True"  Header="AssignedTo" IsReadOnly="True" />  <DataGridTextColumn Binding="{Binding Description}"  CanUserReorder="False" CanUserResize="False" CanUserSort="True"  Header="Description" IsReadOnly="True" />  <DataGridTextColumn Binding="{Binding CreateDate}"  CanUserReorder="False" CanUserResize="False" CanUserSort="True"  Header="CreateDate" IsReadOnly="True" />  <DataGridTextColumn Binding="{Binding UpdateDate}"  CanUserReorder="False" CanUserResize="False"  CanUserSort="True" Header="UpdateDate" IsReadOnly="True" />  <DataGridCheckBoxColumn x:Name="chkApprove"  Binding="{Binding IsApproved, Mode=TwoWay}"  CanUserReorder="True" CanUserResize="True"  CanUserSort="True" Header="IsApprove"/>  <DataGridTextColumn Header="Remarks" x:Name="txtRemarks" Width="180"  Binding="{Binding Remarks,Mode=TwoWay}" MaxWidth="200"/>  </DataGrid.Columns>  </DataGrid>  <Button Name="btnApprove" Content="Appove" Width="50"  HorizontalAlignment="Left" Height="25" Click="Approve\_Click"></Button>  </StackPanel> |

1. Add another ***StackPanel*** to the fifth row of the outer grid, and then add a ***FlowDocumentReader*** control inside the ***StackPanel*** to view the log file data.

|  |
| --- |
| <StackPanel Name="logFileContent" Visibility="Hidden" HorizontalAlignment="Left"  VerticalAlignment="Top" Grid.Row="4">  <FlowDocumentReader Name="FlowDocReader" Width="500" Height="300"  VerticalAlignment="Stretch" HorizontalAlignment="Stretch">  </FlowDocumentReader>  </StackPanel> |

1. Implement the ***Window\_Loaded*** event to load the ***ComboBox*** with a list of Employee Names. Implement the ***ComboBox*** selection changed event to display the list of pending loans for a particular employee. After adding all the xaml code blocks the screen looks similar to below.

|  |
| --- |
|  |

|  |
| --- |
| private void Window\_Loaded(object sender, RoutedEventArgs e)  {  FixupCollection<User> users =  new UserController().GetObjects((int)Role.Employee);  /\* Creating anonymous type \*/  var objUsersToBind = from objuser in users  select new { objuser.Id, objuser.Name };  EmployeesCombo.ItemsSource = objUsersToBind;  } |

|  |
| --- |
| private void EmployeesCombo\_SelectionChanged(object sender,  SelectionChangedEventArgs e)  {  /\* Loan controller internally calls the WCF service for list of loans\*/  List<Loan> objPendingLoansByEmployee = (from objLoan in new  LoanController().GetLoansByEmployeeId(  Convert.ToInt32(EmployeesCombo.SelectedValue))  where objLoan.IsApproved == false  select objLoan).ToList<Loan>();  if (objPendingLoansByEmployee.Count > 0)  {  /\* If selected employee has loans assigned to him bind the datatgrid\*/  dgEmployeeLoans.ItemsSource = objPendingLoansByEmployee;  GridLayout.Visibility = System.Windows.Visibility.Visible;  }  else  {  /\* if no loans exists for selected employee then popup a message \*/  GridLayout.Visibility = System.Windows.Visibility.Visible;  MessageBox.Show("No Approvals pending.");  }  } |

1. Implement another event, ***Logout\_Click*** to logout the current user. The code block similar to below.

|  |
| --- |
| private void Logout\_Click(object sender, RoutedEventArgs e)  {  App.LoggedinUser = null;  MainWindow loginWin = new MainWindow();  loginWin.Show();  this.Close();  } |

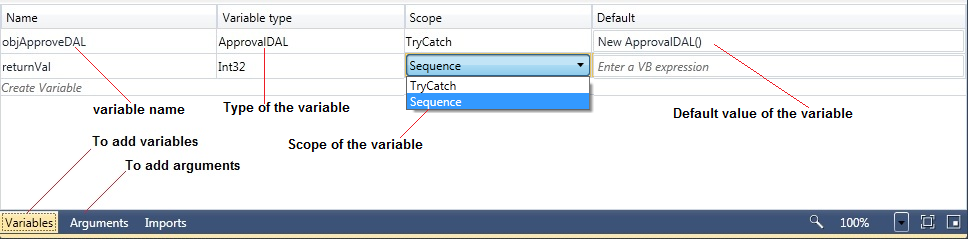
1. Implement the ***Approve\_Click*** event to process the loan approval. The code block might similar to below.

|  |
| --- |
| private void Approve\_Click(object sender, RoutedEventArgs e)  {  /\* To hold the list of error messages if Remarks is not provided \*/  List<string> lstNullRemarks = new List<string>();  /\* list of loan to update \*/  List<Loan> loansToUpdate = dgEmployeeLoans.ItemsSource as List<Loan>;  /\* List of Approval objects to approve loan \*/  FixupCollection<Approval> objApproveCollection = new FixupCollection<Approval>();  /\* Add Loans that need to be approved to LoansToUpdate Collection  \* and also set the list of error messages, if any.  \* Parallel programming Implementation \*/  Parallel.ForEach<Loan>(  loansToUpdate, // Source  objLoan => // Action  {  if (objLoan.IsApproved)  {  if (string.IsNullOrEmpty(objLoan.Remarks))  {  lstNullRemarks.Add("Enter Remarks for Loan " + objLoan.Id.ToString());  }  objApproveCollection.Add(new Approval  {  Loan = objLoan,  Approver = App.LoggedinUser,  IsApproved = true,  Remarks = objLoan.Remarks,  DateApproved = System.DateTime.Now  });  }  });  /\*foreach (Loan objLoan in loansToUpdate)  {  if (objLoan.IsApproved)  {  if (string.IsNullOrEmpty(objLoan.Remarks))  {  lstNullRemarks.Add("Enter Remarks for Loan " + objLoan.Id.ToString());  continue;  }  objApproveCollection.Add(new Approval  {  Loan = objLoan,  Approver = App.LoggedinUser,  IsApproved = true,  Remarks = objLoan.Remarks,  DateApproved = System.DateTime.Now  });  }  }\*/  if (lstNullRemarks.Count > 0)  {  /\* LINQ query to display the message where Remarks is not given for corresponding Loan Id \*/  MessageBox.Show(lstNullRemarks.Select(s => s).Aggregate((a, b) => a + "\n" + b), "Approve Status", MessageBoxButton.OK);  lstNullRemarks.Clear();  return;  }  if (objApproveCollection.Count > 0)  {  ApprovalController objApproval = new ApprovalController();  try  {  /\* Invoke a Workflow through controller \*/  objApproval.InvokeWF(objApproveCollection);  /\* Recall the binding \*/  this.EmployeesCombo\_SelectionChanged(EmployeesCombo, null);  MessageBox.Show("Loans approved successfully", "Loan Approve Status", MessageBoxButton.OK, MessageBoxImage.Information);  }  catch  {  /\* Popup an error message when an exception is occured in the process \*/  MessageBox.Show("Failed to Approve Loans", "Loan Approve Status", MessageBoxButton.OK, MessageBoxImage.Error);  }  finally  {  /\* Clear the collections when execution runs successfully \*/  objApproveCollection.Clear();  objApproveCollection = null;  loansToUpdate.Clear();  loansToUpdate = null;  }  }  else  {  /\* Popup message when no loans are selected for approval \*/  MessageBox.Show("Select atleast one Loan to approve.");  }  } |

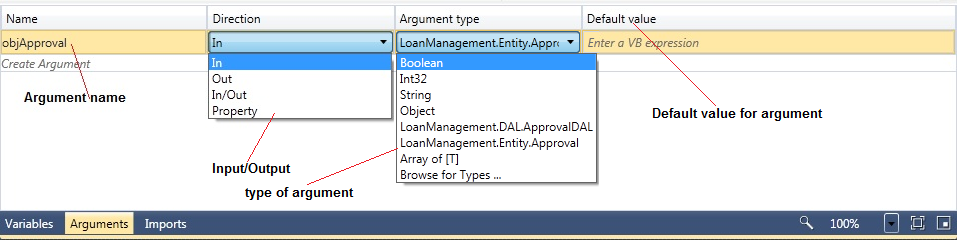
1. Implement the ***View Log*** button click event using the memory mapped files concept. The code block might similar to below. [**Note:** User VIEW\_FLAG variable of type bool to check whether to show the logged details or not.]

|  |
| --- |
| private void ViewApproveLog\_Click(object sender, RoutedEventArgs e)  {  if (VIEW\_FLAG)  {  logFileContent.Visibility = System.Windows.Visibility.Visible;  System.Text.StringBuilder logContent = new System.Text.StringBuilder();  string currentExecutionPath =  System.IO.Path.GetDirectoryName(  System.Reflection.Assembly.GetExecutingAssembly().GetName().CodeBase);  string APPROVE\_LOG\_PATH =  string.Format(@"{0}\LoanManagementErrorLog.txt",  currentExecutionPath.Replace(@"\bin\Debug",  string.Empty).Replace(@"file:\",string.Empty));  using (System.IO.StreamReader approvelog = new  System.IO.StreamReader(APPROVE\_LOG\_PATH))  {  logContent.Append(approvelog.ReadToEnd());  }  Paragraph fileContent = new Paragraph();  fileContent.Inlines.Add(logContent.ToString());    FlowDocument document = new FlowDocument(fileContent);  FlowDocReader.Document = document;  /\* Set the View Log Flag\*/  VIEW\_FLAG = false;  btnViewLogFile.Content = "Close Log";  }  else  {  /\* Reset the button and FlowDocument viewer \*/  logFileContent.Visibility = System.Windows.Visibility.Hidden;  VIEW\_FLAG = true;  btnViewLogFile.Content = "View Log";  }  } |

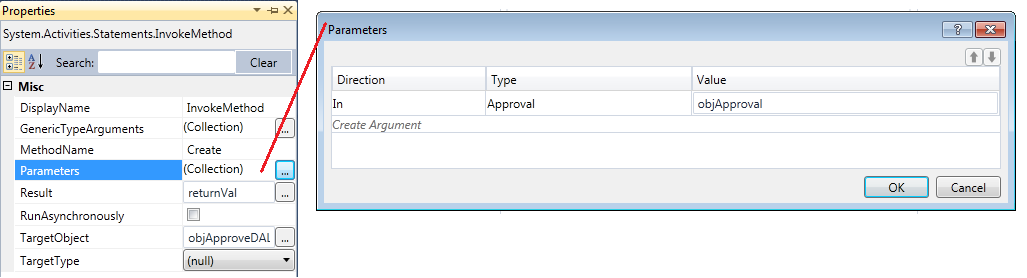
1. Add a new WorkFlow project, ***LoanManagement.Business*** to the ***Business*** solution folder.
2. Add ***LoanManagement.DAL*** and ***LoanManagement.Entity*** references to the current project.
3. Add new activity designer file, ***LoanApproveActivity.xaml*** to the current project.
4. Drag and drop Sequential Activity from the tool box to the designer area of the ***LoanManagementActivity.xaml*** page. In the bottom-left portion of the designer area, find the ***Variables*** tab and click on it. In that window add a new property, **returnVal** of type int.
5. Drag and drop a try-catch activity inside the sequence activity, then add the new variable **objApprovalDAL** of **ApprovalDAL** type in the variables window.

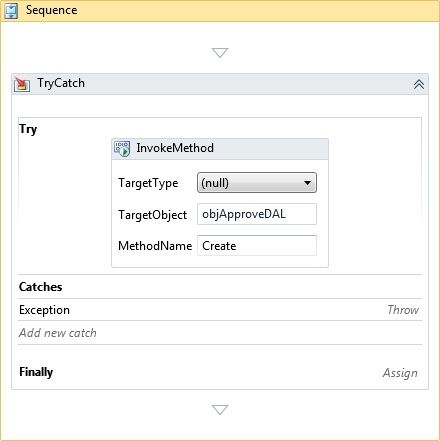


1. Go to the arguments tab and add a new argument, **objApproval** of type Approval.



1. Add the ***InvokeMethod()*** activity from tools to the *try-catch* block, then add **TargetObject** as **objApprovalDAL**, which was created in the 13th step. Set the method name to ***Create*** and add the parameters to the method by selecting the properties for the invoke method and by selecting parameters. Click on the three dots and a parameters popup window will open. In that window, select the parameter direction, type, and value.





1. Open the ***ApprovalController.cs*** file which was created previously in the ***LoanManagement.Controller*** project. Select the project and right click on it, then add references to ***System.Activities*** and to the ***System.Activities.Presentation***.
2. Implement a method, ***InvokeWF()*** that takes the ***FixupCollection***, a collection of *Approval* objects as input argument and calls the WF. Log the approval status too. The code blocks might look similar to below.

|  |
| --- |
| public void InvokeWF(FixupCollection<Approval> objApproveCollection)  {  /\* Create a WorkFlow Activity instance and invoke the WorkFlow  \* Object Initializer implementation  \* Parallel programming implementaion using Task \*/  foreach (Approval objApproval in objApproveCollection)  {  Appro val objApprovalTemp = objApproval;    /\* Parallel programming implementation using Task  Task task = Task.Factory.StartNew(() =>  {\*/  Activity loanApproveActivity = new LoanManagement.Business.LoanApproveActivity()  {  objApproval = new InArgument<Approval>(arg => objApprovalTemp)  };  WorkflowInvoker.Invoke(loanApproveActivity);  /\* Log the approval process to a file \*/  if (!File.Exists(APPROVE\_LOG\_PATH))  {  File.Create(APPROVE\_LOG\_PATH).Close();  string errorMessage1 = string.Format("{0,-10}{1,-50}{2,20}\n", "Loan Id", "Approved By", "Date & Time");  LogApprovalToFile(errorMessage1);  }  /\* Log the exception details \*/  string errorMessage2 = string.Format("{0,-10}{1,-50}{2,20}\n", objApprovalTemp.Loan.Id, objApprovalTemp.Approver.Name, System.DateTime.Now);  LogApprovalToFile(errorMessage2);  //});  }  } |
| private void LogApprovalToFile(string logMessage)  {  using (StreamWriter approvelogfile = File.AppendText(APPROVE\_LOG\_PATH))  {  approvelogfile.WriteLine(logMessage);  approvelogfile.Close();  }  } |

1. Open ***LoanManagement.ConsoleUI*** project -> implement code block that calls the *ApprovalController*’s ***InvokeWF()*** method. The code looks similar to below.

|  |
| --- |
| ApprovalController objApprove = new ApprovalController();  FixupCollection<Approval> loansToApprove = new FixupCollection<Approval>();  loansToApprove.Add(new Approval  {  Approver = new User(3),  Loan = new Loan(2),  IsApproved = true,  DateApproved = DateTime.Now,  Remarks = "Approved",  CreateDate = DateTime.Now,  UpdateDate = DateTime.Now  });  try  {  objApprove.InvokeWF(loansToApprove);  Console.WriteLine("Loan(s) approved successfully.");  }  catch (Exception ex)  {  Console.WriteLine(ex.Message);  } |