The complete code for the class Project:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace PTSLibrary
{
   class Project
   {
      private string name;
      private DateTime expectedStartDate;
      private DateTime expectedEndDate;
      private Customer theCustomer;
      private Guid projectId;
      private List<Task> tasks;
      public List<Task> Tasks
      {
         get { return tasks; }
         set { tasks = value; }
      }
      public Customer TheCustomer
      {
         get { return theCustomer; }
         set { theCustomer = value; }
      }
      public Guid ProjectId
      {
         get { return projectId; }
      }
      public string Name
      {
         get { return name; }
         set { name = value; }
      }
      public DateTime ExpectedStartDate
      {
         get { return expectedStartDate; }
         set { expectedStartDate = value; }
      }
      public DateTime ExpectedEndDate
      {
         get { return expectedEndDate; }
         set { expectedEndDate = value; }
      }
      public Project(string name, DateTime startDate, DateTime endDate, Guid projectId,
          Customer customer)
      {
         this.name = name;
         this.expectedStartDate = startDate;
         this.expectedEndDate = endDate;
         this.projectId = projectId;
         this.theCustomer = customer;
```

```
        }
        public Project(string name, DateTime startDate, DateTime endDate, Guid projectId,
            List<Task> tasks)
        {
            this.name = name;
            this.expectedStartDate = startDate;
            this.expectedEndDate = endDate;
            this.projectId = projectId;
            this.tasks = tasks;
        }
    }
}
```

Complete code listing for the class SuperDAO:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;
using System.Data;
namespace PTSLibrary.DAO
{
    class SuperDAO
    {
        protected Customer GetCustomer(int custId)
        {
            string sql;
            SqlConnection cn;
            SqlCommand cmd;
            SqlDataReader dr;
            Customer cust;
            sql = "SELECT * FROM Customer WHERE CustomerId = " + custId;
            cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
            cmd = new SqlCommand(sql, cn);
            try
            {
                cn.Open();
                dr = cmd.ExecuteReader(CommandBehavior.SingleRow);
                dr.Read();
                cust = new Customer(dr["Name"].ToString(), (int)dr["CustomerId"]);
                dr.Close();
            }
            catch (SqlException ex)
            {
                throw new Exception("Error Getting Customer", ex);
            }
            finally
```

```
        {
            cn.Close();
        }
        return cust;
    }
    public List<Task> GetListOfTasks(Guid projectId)
    {
        string sql;
        SqlConnection cn;
        SqlCommand cmd;
        SqlDataReader dr;
        List<Task> tasks;
        tasks = new List<Task>();
        sql = "SELECT * FROM Task WHERE ProjectId = '" + projectId + "'";
        cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
        cmd = new SqlCommand(sql, cn);
        try
        {
            cn.Open();
            dr = cmd.ExecuteReader();
            while (dr.Read())
            {
                Task t = new Task((Guid)dr["TaskId"], dr["Name"].ToString(),
          (Status)((int)dr["StatusId"]));
                tasks.Add(t);
            }
            dr.Close();
        }
        catch (SqlException ex)
        {
            throw new Exception("Error getting taks list", ex);
        }
        finally
        {
            cn.Close();
        }
        return tasks;
    }
  }
}
```

Please note that the visibility for this method is set to protected so that it can be accessed in the subclasses.

Objects necessary to access the DB are SqlConnection, SqlCommand, SqlDataReader.

ConnectionString is defined by the Settings.settings file and accessed using the following code: *Properties.Settings.Default.ConnectionString*

The **try** and **catch** statement should be used whenever a connection to the DB is opened. Note that the connection is closed in the **finally** block; so that the connection will always be closed even if an error occurs.

To keep the code as simple as possible we assume a customer is always returned. In real system you should cater for the possibility of an empty row being returned.

When the SQL command is executed it will populate the SqlDataReader with the returned data. We can then extract this data by referring to the names of the columns in the Customer table from the DB. Notice how the value of the *"Name"* is cast into a string and the value of the *"CustomerId"* is cast into an integer:

```
cust = new Customer(dr["Name"].ToString(), (int)dr["CustomerId"]);
```

This method is similar to the GetCUstomer one. The main difference is that we need to retrieve a list of tasks for a given project (passed as a parameter)

The instantiated Task object ***t*** is added to the ***tasks*** list. The List ***tasks*** will only accept objects of type Task because we are using generics.

Funny bit of code: **(Status)((int)dr["StatusId"]).** The StatusId is accessed from the DB and its value is cast into an integer. When we created the items in the Status enumeration, we ensured that they had the same number as the StatusId in the Status table. This is why we can take the StatusId returned and cast it straight into Status.

Complete code listing for the CustomerDAO class:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;
using System.Data;
namespace PTSLibrary.DAO
{
   class CustomerDAO : SuperDAO
   {
      public int Authenticate(string username, string password)
      {
         string sql;
         SqlConnection cn;
         SqlCommand cmd;
         SqlDataReader dr;
         sql = String.Format("SELECT CustomerId FROM Customer WHERE Username='{0}' AND
           Password='{1}'", username, password);
         cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
         cmd = new SqlCommand(sql, cn);
         int id = 0;
         try
         {
            cn.Open();
            dr = cmd.ExecuteReader(CommandBehavior.SingleRow);
            if (dr.Read())
            {
               id = (int)dr["CustomerId"];
```

```
        }
        dr.Close();
    }
    catch (SqlException ex)
    {
        throw new Exception("Error Accessing Database", ex);
    }
    finally
    {
        cn.Close();
    }
    return id;
}
public List<Project> GetListOfProjects(int customerId)
{
    string sql;
    SqlConnection cn, cn2;
    SqlCommand cmd, cmd2;
    SqlDataReader dr, dr2;
    List<Project> projects;
    projects = new List<Project>();
    sql = "SELECT * FROM Project WHERE CustomerId = " + customerId;
    cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
    cmd = new SqlCommand(sql, cn);
    try
    {
        cn.Open();
        dr = cmd.ExecuteReader();
        while (dr.Read())
        {
            List<Task> tasks = new List<Task>();
            sql = "SELECT * FROM Task WHERE ProjectId = '" + dr["ProjectId"].ToString() + "'";
            cn2 = new SqlConnection(Properties.Settings.Default.ConnectionString);
            cmd2 = new SqlCommand(sql, cn2);
            cn2.Open();
            dr2 = cmd2.ExecuteReader();
            while (dr2.Read())
            {
                Task t = new Task((Guid)dr2["TaskId"], dr2["Name"].ToString(),
      (Status)dr2["StatusId"]);
                tasks.Add(t);
            }
            dr2.Close();
            Project p = new Project(dr["Name"].ToString(), (DateTime)dr["ExpectedStartDate"],
                            (DateTime)dr["ExpectedEndDate"], (Guid)dr["ProjectId"], tasks);
            projects.Add(p);
        }
        dr.Close();
    }
    catch (SqlException ex)
    {
```

```
            throw new Exception("Error Getting list", ex);
        }
        finally
        {
            cn.Close();
        }
        return projects;
    }
  }
}
```

Note *: The Client refers to a Java or .NET Client; in this case is the TeamLeader. It does not refer to Customer.

Complete code listing for the CustomerDAO class:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;
using System.Data;
namespace PTSLibrary.DAO
{
  class ClientDAO : SuperDAO
  {
    public TeamLeader Authenticate(string username, string password)
    {
      string sql;
      SqlConnection cn;
      SqlCommand cmd;
      SqlDataReader dr;
      TeamLeader leader = null;
      sql = String.Format("SELECT DISTINCT Person.Name, UserId, TeamId FROM Person INNER
        JOIN Team ON (Team.TeamLeaderId = Person.UserId) WHERE  Username='{0}' AND
        Password='{1}'", username, password);
      cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
      cmd = new SqlCommand(sql, cn);
      try
      {
        cn.Open();
        dr = cmd.ExecuteReader(CommandBehavior.SingleRow);
        if (dr.Read())
        {
          leader = new TeamLeader(dr["Name"].ToString(), (int)dr["TeamId"],
        (int)dr["TeamId"]);
        }
        dr.Close();
      }
      catch (SqlException ex)
      {
```

```
            throw new Exception("Error Accessing Database", ex);
        }
        finally
        {
            cn.Close();
        }
        return leader;
    }
    public List<Project> GetListOfProjects(int teamId)
    {
        string sql;
        SqlConnection cn;
        SqlCommand cmd;
        SqlDataReader dr;
        List<Project> projects;
        projects = new List<Project>();
        sql = "SELECT P.* FROM Project AS P INNER JOIN Task AS T ON (P.ProjectId = T.ProjectId)
          WHERE T.TeamId = " + teamId;
        cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
        cmd = new SqlCommand(sql, cn);
        try
        {
            cn.Open();
            dr = cmd.ExecuteReader();
            while (dr.Read())
            {
                Customer cust = GetCustomer((int)dr["CustomerId"]);
                Project p = new Project(dr["Name"].ToString(), (DateTime)dr["ExpectedStartDate"],
          (DateTime)dr["ExpectedEndDate"], (Guid)dr["ProjectId"], cust);
                projects.Add(p);
            }
            dr.Close();
        }
        catch (SqlException ex)
        {
            throw new Exception("Error Getting list", ex);
        }
        finally
        {
            cn.Close();
        }
        return projects;
    }
  }
}
```

SQL ensures that only a person who is a TeamLeader can authenticate.

Complete code listing for the AdminDAO class:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;
using System.Data;
namespace PTSLibrary.DAO
{
   class AdminDAO : SuperDAO
   {
      public int Authenticate(string username, string password)
      {
         string sql;
         SqlConnection cn;
         SqlCommand cmd;
         SqlDataReader dr;
         sql = String.Format("SELECT UserId FROM Person WHERE IsAdministrator = 1 AND
           Username='{0}' AND Password='{1}'", username, password);
         cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
         cmd = new SqlCommand(sql, cn);
         int id = 0;
         try
         {
            cn.Open();
            dr = cmd.ExecuteReader(CommandBehavior.SingleRow);
            if (dr.Read())
            {
               id = (int)dr["UserId"];
            }
            dr.Close();
         }
         catch (SqlException ex)
         {
            throw new Exception("Error Accessing Database", ex);
         }
         finally
         {
            cn.Close();
         }
         return id;
      }
      public void CreateProject(string name, DateTime startDate, DateTime endDate, int
          customerId, int administratorId)
      {
         string sql;
         SqlConnection cn;
         SqlCommand cmd;
         Guid projectId = Guid.NewGuid();
         sql = "INSERT INTO Project (ProjectId, Name, ExpectedStartDate, ExpectedEndDate,
           CustomerId, AdministratorId)";
```

```
    sql += String.Format("VALUES ( '{0}', '{1}', '{2}', '{3}', {4}, {5})", projectId, name, startDate,
      endDate, customerId, administratorId);
    cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
    cmd = new SqlCommand(sql, cn);
    try
    {
      cn.Open();
      cmd.ExecuteNonQuery();
    }
    catch (SqlException ex)
    {
      throw new Exception("Error Inserting", ex);
    }
    finally
    {
      cn.Close();
    }
}
public List<Customer> GetListOfCustomers()
{
    string sql;
    SqlConnection cn;
    SqlCommand cmd;
    SqlDataReader dr;
    List<Customer> customers;
    customers = new List<Customer>();
    sql = "SELECT * FROM Customer";
    cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
    cmd = new SqlCommand(sql, cn);
    try
    {
      cn.Open();
      dr = cmd.ExecuteReader();
      while (dr.Read())
      {
        Customer c = new Customer(dr["Name"].ToString(), (int)dr["CustomerId"]);
        customers.Add(c);
      }
      dr.Close();
    }
    catch (SqlException ex)
    {
      throw new Exception("Error Getting list", ex);
    }
    finally
    {
      cn.Close();
    }
    return customers;
}
public List<Project> GetListOfProjects(int adminId)
```

```
{
    string sql;
    SqlConnection cn;
    SqlCommand cmd;
    SqlDataReader dr;
    List<Project> projects;
    projects = new List<Project>();
    sql = "SELECT * FROM Project WHERE AdministratorId = " + adminId;
    cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
    cmd = new SqlCommand(sql, cn);
    try
    {
        cn.Open();
        dr = cmd.ExecuteReader();
        while (dr.Read())
        {
            Customer cust = GetCustomer((int)dr["CustomerId"]);
            Project p = new Project(dr["Name"].ToString(), (DateTime)dr["ExpectedStartDate"],
      (DateTime)dr["ExpectedEndDate"], (Guid)dr["ProjectId"], cust);
            projects.Add(p);
        }
        dr.Close();
    }
    catch (SqlException ex)
    {
        throw new Exception("Error Getting list", ex);
    }
    finally
    {
        cn.Close();
    }
    return projects;
}
public List<Team> GetListOfTeams()
{
    string sql;
    SqlConnection cn;
    SqlCommand cmd;
    SqlDataReader dr;
    List<Team> teams;
    teams = new List<Team>();
    sql = "SELECT * FROM Team";
    cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
    cmd = new SqlCommand(sql, cn);
    try
    {
        cn.Open();
        dr = cmd.ExecuteReader();
        while (dr.Read())
        {
```

```
            Team t = new Team((int)dr["TeamId"], dr["Location"].ToString(),
          dr["Name"].ToString(), null);
            teams.Add(t);
          }
          dr.Close();
        }
        catch (SqlException ex)
        {
          throw new Exception("Error getting team list", ex);
        }
        finally
        {
          cn.Close();
        }
        return teams;
      }
      public void CreateTask(string name, DateTime startDate, DateTime endDate, int teamId,
          Guid projectId)
      {
        string sql;
        SqlConnection cn;
        SqlCommand cmd;
        Guid taskId = Guid.NewGuid();
        sql = "INSERT INTO Task (TaskId, Name, ExpectedDateStarted, ExpectedDateCompleted,
          ProjectId, TeamId, StatusId)";
        sql += String.Format("VALUES ( '{0}', '{1}', '{2}', '{3}', '{4}', {5}, {6})", taskId, name,
          startDate, endDate, projectId, teamId, 1);
        cn = new SqlConnection(Properties.Settings.Default.ConnectionString);
        cmd = new SqlCommand(sql, cn);
        try
        {
          cn.Open();
          cmd.ExecuteNonQuery();
        }
        catch (SqlException ex)
        {
          throw new Exception("Error Inserting", ex);
        }
        finally
        {
          cn.Close();
        }
      }
    }
}
```

Tricky code: constructor of the class

In the PTSCustomeFacade we need to access all the CustomerDAO methods and not only the methods available in SuperDAO.

The following code does this:

```
        public PTSCustomerFacade() : base(new DAO.CustomerDAO())
    {
      dao = (DAO.CustomerDAO)base.dao;
    }
```

When an instance of PTSCustomerFacade is created, a call is made to the constructor of the super class (PTSSuperFacade). As an argument we pass in a new instance of CustomerDAO. This is acceptable because the constructor of PTSSuperFacade takes a parameter of type SuperDAO and CustomerDAO is a subclass of SUperDAO.

The instance of SuperDAO in the super (base) class is then cast back into an instance of CustomerDAO in the subclass. This is possible due to late binding.

Late biding: although the dao variable in the super class is currently of type SuperDAO, the object in memory it refers to is of type CustomerDAO (because this is what it was passed in the constructor). It is there for legal to cast it back to CUstomerDAO.

Complete code listing for the PTSClientFacade class:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace PTSLibrary
{
  class PTSAdminFacade : PTSSuperFacade
  {
    private DAO.AdminDAO dao;
    public PTSAdminFacade() : base(new DAO.AdminDAO())
    {
      dao = (DAO.AdminDAO)base.dao;
    }
    public int Authenticate(string username, string password)
    {
      if (username == "" || password == "")
      {
        throw new Exception("Missing Data");
      }
      return dao.Authenticate(username, password);
    }
    public void CreateProject(string name, DateTime startDate, DateTime endDate, int
        customerId, int administratorId)
    {
      if (name == null || name == "" || startDate == null || endDate == null)
      {
        throw new Exception("Missing Data");
      }
      dao.CreateProject(name, startDate, endDate, customerId, administratorId);
    }
    public Customer[] GetListOfCustomers()
    {
      return (dao.GetListOfCustomers()).ToArray();
    }
    public Project[] GetListOfProjects(int adminId)
    {
```

```
            return (dao.GetListOfProjects(adminId)).ToArray();
        }
        public Team[] GetListOfTeams()
        {
            return (dao.GetListOfTeams()).ToArray();
        }
        public void CreateTask(string name, DateTime startDate, DateTime endDate, int teamId,
            Guid projectId)
        {
            if (name == null || name == "" || startDate == null || endDate == null)
            {
                throw new Exception("Missing Data");
            }
            dao.CreateTask(name, startDate, endDate, teamId, projectId);
        }
    }
}
```