Lab Assignment 7

Jerry Belmonte
Keira Wong

CODE

Program.cs | GenericDataRepository.cs | Standard.cs

ConsoleClient

```
        {
            Menu.displayMenu();
            input = Validator.getMenuInput();

            switch (input)
            {
                case 0:
                    repeat = false;
                    break;
                case 1:
                    Menu.clearMenu();
                    addTeacher();
                    break;
                case 2:
                    Menu.clearMenu();
                    updateTeacher();
                    break;
                case 3:
                    Menu.clearMenu();
                    removeTeacher();
                    break;
                case 4:
                    Menu.clearMenu();
                    listTeachers();
                    break;
                case 5:
                    Menu.clearMenu();
                    listTeacherCourses();
                    break;
                case 6:
                    Menu.clearMenu();
                    addCourse();
                    break;
                case 7:
                    Menu.clearMenu();
                    updateCourse();
                    break;
                case 8:
                    Menu.clearMenu();
                    removeCourse();
                    break;
                case 9:
                    Menu.clearMenu();
                    listCourses();
                    break;
            }
        } while (repeat);
    }
```

Program.cs | GenericDataRepository.cs | Standard.cs | BuinessLayer .cs | Entities.cs | IBusinessLayer.cs | Course.

ConsoleClient | ConsoleClient.Program | listTeacherCou

```
168 public static void listTeacherCourses()
169 {
170     listTeachers();
171     int id = Validator.getId();
172     Teacher teacher = businessLayer.GetTeacherById(id);
173     if (teacher != null)
174     {
175         Console.WriteLine("Listing courses for [ID: {0}, Name: {1}]:", teacher.TeacherId, teacher.TeacherName);
176         if (teacher.Courses.Count > 0)
177         {
178             foreach (Course course in teacher.Courses)
179                 Console.WriteLine("Course ID: {0}, Name: {1}", course.CourseId, course.CourseName);
180         }
181         else
182         {
183             Console.WriteLine("No courses for [ID: {0}, Name: {1}]:", teacher.TeacherId, teacher.TeacherName);
184         };
185
186     }
187     else
188     {
189         Console.WriteLine("Teacher does not exist.");
190     };
191 }
```

Program.cs | GenericDataRepository.cs | Standard.cs | BuinessLayer .cs | Entities.cs | IBusinessLayer.c

ConsoleClient | ConsoleClient.Program

```
158 public static void listTeachers()
159 {
160     IList<Teacher> teachers = businessLayer.GetAllTeachers();
161     foreach (Teacher teacher in teachers)
162         Console.WriteLine("Teacher ID: {0}, Name: {1}", teacher.TeacherId, teacher.TeacherName);
163 }
```

IBusinessLayer.cs | GenericDataRepository.cs | IGenericDataRepo

Mm.BusinessLayer | Mm.B

```
2  using System.Collections.Generic;
3
4  namespace Mm.BusinessLayer
5  {
       2 references
6      public interface IBusinessLayer
7      {
           2 references
8          IList<Teacher> GetAllTeachers();
           4 references
9          Teacher GetTeacherByName(string teacherName);
           5 references
10         Teacher GetTeacherById(int teacherID);
```

BuinessLayer .cs | Program.cs | GenericDataRepository.cs | Standard.cs | Entities.cs | IBusinessLay

Mm.BusinessLayer | Mm.BusinessLayer.BuinessLayer

```
using DomainModel;
using Mm.DataAccessLayer;
using System.Collections.Generic;

namespace Mm.BusinessLayer
{
    2 references
    public class BuinessLayer : IBusinessLayer
    {
        private readonly ITeacherRepository _teacherRepository;
        private readonly ICourseRepository _courseRepository;

        1 reference
        public BuinessLayer()
        {
            _teacherRepository = new TeacherRepository();
            _courseRepository = new CourseRepository();
        }

        0 references
        public BuinessLayer(ITeacherRepository teacherRepository, ICourseRepository courseRepository)
        {
            _teacherRepository = teacherRepository;
            _courseRepository = courseRepository;
        }

        //CRUD for teachers

        2 references
        public IList<Teacher> GetAllTeachers()
        {
            return _teacherRepository.GetAll();
        }
```

BuinessLayer .cs | GenericDataRepository.cs

er | Mm.Busin

```
    }

    5 references
    public Teacher GetTeacherById(int teacherID)
    {
        return _teacherRepository.GetSingle(
            d => d.TeacherId.Equals(teacherID),
            d => d.Courses); //include related Courses
    }
```

IGenericDataRepository.cs | BuinessLayer .cs | Program.cs | Entities.cs | IBusinessLa

Mm.DataAccessLayer | Mm.DataAccessLayer.IGenericDataRepos

```
using DomainModel;
using System;
using System.Collections.Generic;
using System.Linq.Expressions;

namespace Mm.DataAccessLayer
{
    3 references
    public interface IGenericDataRepository<T> where T : class, IEntity
    {
        /* the return type of the two Get* methods is IList<T> rather than IQueryable<T>.
         * This means that the methods will be returning the actual already executed result
         * from the queries rather than executable queries themselves.
         */
        3 references
        IList<T> GetAll(params Expression<Func<T, object>>[] navigationProperties);
```

GenericDataRepository.cs | IGenericDataRepository.cs | BuinessLayer .cs | Program.cs | IBusinessLaye

Mm.DataAccessLayer | Mm.DataAccessLayer.GenericDataRepository<T>

```
using DomainModel;
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Linq.Expressions;

namespace Mm.DataAccessLayer
{
    2 references
    public class GenericDataRepository<T> : IGenericDataRepository<T> where T : class, IEntity
    {
        3 references
        public virtual IList<T> GetAll(params Expression<Func<T, object>>[] navigationProperties)
        {
            List<T> list;
            using (var context = new Entities())
            {
                IQueryable<T> dbQuery = context.Set<T>();

                //Apply eager loading
                foreach (Expression<Func<T, object>> navigationProperty in navigationProperties)
                    dbQuery = dbQuery.Include<T, object>(navigationProperty);

                list = dbQuery
                    .AsNoTracking()
                    .ToList<T>();
            }
            return list;
        }
```

BuinessLayer .cs | GenericDataRepository.cs | IGenericDataRepository.cs | Progran

ayer | Mm.DataAccessLayer.GenericDataRepository<T>

```
5 references
public virtual T GetSingle(Func<T, bool> where,
    params Expression<Func<T, object>>[] navigationProperties)
{
    T item = null;
    using (var context = new Entities())
    {
        IQueryable<T> dbQuery = context.Set<T>();

        //Apply eager loading
        foreach (Expression<Func<T, object>> navigationProperty in navigationProperties)
            dbQuery = dbQuery.Include<T, object>(navigationProperty);

        item = dbQuery
            .AsNoTracking() //Don't track any changes for the selected item
            .FirstOrDefault(where); //Apply where clause
    }
    return item;
}
```

```csharp
/// <summary>
/// List the courses of a specified teacher.
/// </summary>
public static void listTeacherCourses()
{
    listTeachers();
    int id = Validator.getId();
    Teacher teacher = businessLayer.GetTeacherById(id);
    if (teacher != null)
    {
        Console.WriteLine("Listing courses for [ID: {0}, Name: {1}]:", teacher.TeacherId, teacher.TeacherName);
        if (teacher.Courses.Count > 0)
        {
            foreach (Course course in teacher.Courses)
                Console.WriteLine("Course ID: {0}, Name: {1}", course.CourseId, course.CourseName);
        }
        else
        {
            Console.WriteLine("No courses for [ID: {0}, Name: {1}]", teacher.TeacherId, teacher.TeacherName);
        }
    }
    else
    {
        Console.WriteLine("Teacher does not exist.");
    }
}
```

```csharp
/// <summary>
/// List all teachers and courses in the database.
/// </summary>
public static void listAllTeachersAndCourses()
{
    IList<Teacher> teachers = businessLayer.GetAllTeachers();
    foreach (Teacher teacher in teachers)
    {
        Console.WriteLine("Listing courses for [ID: {0}, Name: {1}]:", teacher.TeacherId, teacher.TeacherName);
        int id = Convert.ToInt32(teacher.TeacherId);
        IList<Course> courses = businessLayer.GetCoursesByTeacherId(id);

        if (courses.Count > 0)
        {
            foreach (Course course in courses)
                Console.WriteLine("Course: [ID: {0}, Name: {1}]", course.CourseId, course.CourseName);
        }
        else
        {
            Console.WriteLine("No courses for [ID: {0}, Name: {1}]", teacher.TeacherId, teacher.TeacherName);
        }
    }
}
```

```
/// <summary>
/// Update the name of a course.
/// </summary>
public static void updateCourse()
{
    Menu.displaySearchOptions();
    int input = Validator.getOptionInput();
    listCourses();
    Course course = null;

    //find course by name
    if (input == 1)
    {
        Console.WriteLine("Enter a course's name: ");
        course = businessLayer.GetCourseByName(Console.ReadLine());
    }
    //find course by id
    else if (input == 2)
    {
        course = businessLayer.GetCourseById(Validator.getId());
    }

    // update the course if the course exists
    if (course != null)
    {
        Menu.displayUpdateCourseOptions();
        int ucoInput = Validator.getOptionInput();
        // [1] change name
        if (ucoInput == 1)
        {
            Console.WriteLine("Change this course's name to: ");
            course.CourseName = Console.ReadLine();
            course.EntityState = EntityState.Modified;
            businessLayer.UpdateCourse(course);
        }
        // [2] change teacher
        else if (ucoInput == 2)
        {
            // get the current teacher for the course
            int id = Convert.ToInt32(course.TeacherId);
            Teacher curTeacher = businessLayer.GetTeacherById(id);
            Console.WriteLine("Current teacher for the course: ");
            Console.WriteLine($"Teacher: [ID: {curTeacher.TeacherId}, Name: {curTeacher.TeacherName}]");

            // get the new teacher selection
            Console.WriteLine("Change this course's teacher to: ");
            foreach (Teacher teacher in businessLayer.GetAllTeachers())
            {
                // only list the teachers that are different that the current one
                if (teacher.TeacherId != id)
                {
                    Console.WriteLine("Teacher ID: {0}, Name: {1}", teacher.TeacherId, teacher.TeacherName);
                }
            }
            course.Teacher = businessLayer.GetTeacherById(Validator.getId());

            // change the teacher if the selection was valid
            if (course.Teacher != null && course.Teacher.TeacherId != id)
            {
                // update course
                Console.WriteLine("{0} has been updated.", course.CourseName);
                course.TeacherId = course.Teacher.TeacherId;
                course.EntityState = EntityState.Modified;
                businessLayer.UpdateCourse(course);
            }
            else
            {
                Console.WriteLine("Not a valid Teacher selection.");
            }
        }
    }
    else
    {
        Console.WriteLine("Course does not exist.");
    }
}
```

```
/// <summary>
/// Remove a course in the database.
/// </summary>
public static void removeCourse()
{
    listCourses();
    int id = Validator.getId();
    Course course = businessLayer.GetCourseById(id);
    if (course != null)
    {
        // TODO: remove course from teacher from database
        Console.WriteLine("{0} has been removed.", course.CourseName);
        course.EntityState = EntityState.Deleted;
        businessLayer.RemoveCourse(course);
    }
    else
    {
        Console.WriteLine("Course does not exist.");
    }
}
```

```
public class CourseRepository : GenericDataRepository<Course>, ICourseRepository
{
    public override void Remove(params Course[] items)
    {
        using (var contex = new Entities())
        {
            var dbSet = contex.Set<Course>();
            foreach (var item in items)
            {
                item.Students.Clear(); // remove any students relation
                item.Teacher = null; // remove any teacher relation
                dbSet.Add(item);
                foreach (var entry in contex.ChangeTracker.Entries<Course>())
                {
                    var entity = entry.Entity;
                    entry.State = GetEntityState(entity.EntityState);
                }
            }
            contex.SaveChanges();
        }
    }
}
```