```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer
{
    public class CourseRepository : Repository<Course>, ICourseRepository
    {
        public CourseRepository() : base(new SchoolDBEntities())
        {

        }
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DataAccessLayer;

namespace BusinessLayer
{
    public interface IBusinessLayer
    {
        IList<Standard> getAllStandards();
        Standard GetStandardByID(int id);
        Standard GetStandardByName(string name);
        void AddStandard(Standard standard);
        void UpdateStandard(Standard standard);
        void RemoveStandard(Standard standard);

        IList<Student> getAllStudents();
        Student GetStudentByID(int id);
        Student GetStudentByName(string name);
        void AddStudent(Student student);
        void UpdateStudent(Student student);
        void RemoveStudent(Student student);

        IList<Teacher> getAllTeachers();
        Teacher GetTeacherByID(int id);
        Teacher GetTeacherByName(string name);
        void AddTeacher(Teacher teacher);
        void UpdateTeacher(Teacher teacher);
        void RemoveTeacher(Teacher teacher);

        IList<Course> getAllCourses();
        Course GetCourseByID(int id);
        Course GetCourseByName(string name);
        void AddCourse(Course course);
        void UpdateCourse(Course course);
        void RemoveCourse(Course course);

        IList<Course> GetCoursesByTeacherID(int id);
        IList<Course> GetCoursesByTeacherName(string name);

        IList<Student> GetStudentsByStandardID(int id);
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer
{
    public interface ICourseRepository : IRepository<Course>
    {

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer
{
    public interface IRepository<T> : IDisposable
    {
        void Insert(T entity);

        void Delete(T entity);

        void Update(T entity);

        T GetById(int id);

        IQueryable<T> SearchFor(Expression<Func<T, bool>> predicate);

        IEnumerable<T> GetAll();

        T GetSingle(Func<T, bool> where, params Expression<Func<T, object>>[]
          navigationProperties);
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer
{
    public interface IStandardRepository : IRepository<Standard>
    {

    }
}

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer
{
    public interface IStudentRepository : IRepository<Student>
    {

    }
}

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer
{
    public interface ITeacherRepository : IRepository<Teacher>
    {

    }
}

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Linq.Expressions;
using System.Data.Entity;
using SqlServerTypes;
using System.Data.Entity.SqlServer;
using System.Data.Entity.Infrastructure;

namespace DataAccessLayer
{
    public class Repository<T> : IRepository<T> where T : class
    {
        protected DbContext context;
        protected DbSet<T> dbset;
        public Repository(DbContext datacontext)
        {
            //You can use the cpmt
            this.context = datacontext;
            dbset = context.Set< T>();
            SqlProviderServices.SqlServerTypesAssemblyName =
              "Microsoft.SqlServer.Types, Version=14.0.0.0, Culture=neutral,
              PublicKeyToken=89845dcd8080cc91";
        }

        public void Insert(T entity)
        {
            //Use the context object and entity state to save the entity
            try
            {
                context.Entry(entity).State = EntityState.Added;
                context.SaveChanges();
            }
            catch (DbUpdateConcurrencyException e)
            {
                e.Entries.Single().Reload();
            }
            catch (InvalidOperationException e2) { }
        }

        public void Delete(T entity)
        {
            //Use the context object and entity state to delete the entity
            //SqlProviderServices.SqlServerTypesAssemblyName =
              "Microsoft.SqlServer.Types, Version=14.0.0.0, Culture=neutral,
              PublicKeyToken=89845dcd8080cc91";
            //SqlServerTypes.Utilities.LoadNativeAssemblies
```

```
              (AppDomain.CurrentDomain.BaseDirectory);

            try
            {
                context.Entry(entity).State = EntityState.Deleted;
                context.SaveChanges();
            }
            catch (DbUpdateConcurrencyException e)
            {
                e.Entries.Single().Reload();
            }

        }

        public void Update(T entity)
        {
            //Use the context object and entity state to update the entity
            //SqlServerTypes.Utilities.LoadNativeAssemblies(AppDomain.CurrentDomain.BaseDirectory);
            try
            {
                SqlProviderServices.SqlServerTypesAssemblyName = "Microsoft.SqlServer.Types, Version=14.0.0.0, Culture=neutral, PublicKeyToken=89845dcd8080cc91";
                SqlServerTypes.Utilities.LoadNativeAssemblies(AppDomain.CurrentDomain.BaseDirectory);
                context.Entry(entity).State = EntityState.Modified;
                context.SaveChanges();
            }
            catch (DbUpdateConcurrencyException e)
            {
                e.Entries.Single().Reload();
            }
            catch (InvalidOperationException e2)
            {

            }

        }

        public T GetById(int id)
        {
            try
            {
                return dbset.Find(id);
            }
            catch (InvalidOperationException e)
            {
                return null;
            }
        }

        public IQueryable<T> SearchFor(Expression<Func<T, bool>> predicate)
        {
            return dbset.Where(predicate);
            //return context.Where(predicate);
        }

        public IEnumerable<T> GetAll()
        {
            return dbset.ToList();
        }

        //This method will find the related records by passing two argument
        //First argument: lambda expression to search a record such as d => d.StandardName.Equals(standardName) to search am record by standard name
        //Second argument: navigation property that leads to the related records such as d => d.Students
        //The method returns the related records that met the condition in the first argument.
        //An example of the method GetStandardByName(string standardName)
        //public Standard GetStandardByName(string standardName)
        //{
        //return _standardRepository.GetSingle(d => d.StandardName.Equals(standardName), d => d.Students);
        //}
        public T GetSingle(Func<T, bool> where, params Expression<Func<T, object>>[] navigationProperties)
        {
            T item = null;
            IQueryable<T> dbQuery = context.Set<T>();
            foreach (Expression<Func<T, object>> navigationProperty in navigationProperties)
                dbQuery = dbQuery.Include<T, object>(navigationProperty);
            item = dbQuery.AsNoTracking().FirstOrDefault(where);
            return item;

        }

        public void Dispose()
        {
            throw new NotImplementedException();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer
{
    public class StudentRepository : Repository<Student>, IStudentRepository
    {
        public StudentRepository() : base(new SchoolDBEntities())
        {

        }
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer
{
    public class StandardRepository : Repository<Standard>, IStandardRepository
    {
        public StandardRepository() : base(new SchoolDBEntities())
        {
        }
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer
{
    public class TeacherRepository : Repository<Teacher>, ITeacherRepository
    {
        public TeacherRepository() : base(new SchoolDBEntities())
        {
        }
    }
}

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DataAccessLayer;

namespace BusinessLayer
{
    public class BusinessLayer : IBusinessLayer
    {
        private readonly IStandardRepository standardRepository;
        private readonly IStudentRepository studentRepository;
        private readonly ITeacherRepository teacherRepository;
        private readonly ICourseRepository courseRepository;

        public BusinessLayer()
        {
            standardRepository = new StandardRepository();
            studentRepository = new StudentRepository();
            courseRepository = new CourseRepository();
            teacherRepository = new TeacherRepository();
        }

        public void AddStandard(Standard standard)
        {
            standardRepository.Insert(standard);
            //throw new NotImplementedException();
        }

        public void AddStudent(Student student)
        {
            studentRepository.Insert(student);
            //throw new NotImplementedException();
        }

        public IList<Standard> getAllStandards()
        {
            return standardRepository.GetAll().ToList();
        }

        public IList<Student> getAllStudents()
        {
            return studentRepository.GetAll().ToList();
            //throw new NotImplementedException();
        }

        public Standard GetStandardByID(int id)
        {
            return standardRepository.GetById(id);
            //throw new NotImplementedException();
        }

        public Standard GetStandardByName(string name)
        {
            return standardRepository.GetSingle(s => s.StandardName.Equals
              (name), s => s.Students, s => s.Teachers);
        }

        public Student GetStudentByID(int id)
        {
            return studentRepository.GetById(id);
            //throw new NotImplementedException();
        }

        public Student GetStudentByName(string name)
        {
            return studentRepository.GetSingle(s => s.StudentName.Equals(name),
               s => s.StudentAddress);
        }

        public void RemoveStandard(Standard standard)
        {
            standardRepository.Delete(standard);
            //throw new NotImplementedException();
        }

        public void RemoveStudent(Student student)
        {
            studentRepository.Delete(student);
            //throw new NotImplementedException();
        }

        public void UpdateStandard(Standard standard)
        {
            standardRepository.Update(standard);
            //throw new NotImplementedException();
        }

        public void UpdateStudent(Student student)
        {
            studentRepository.Update(student);
            //throw new NotImplementedException();
        }

        public IList<Teacher> getAllTeachers()
        {
            return teacherRepository.GetAll().ToList();
        }

        public Teacher GetTeacherByID(int id)
        {
            return teacherRepository.GetById(id);
        }

        public Teacher GetTeacherByName(string name)
        {
            return teacherRepository.GetSingle(t => t.TeacherName.Equals(name), 
              t => t.Standard);
        }

        public void AddTeacher(Teacher teacher)
        {
            teacherRepository.Insert(teacher);
        }

        public void UpdateTeacher(Teacher teacher)
        {
            teacherRepository.Update(teacher);
        }

        public void RemoveTeacher(Teacher teacher)
        {
            teacherRepository.Delete(teacher);
        }

        public IList<Course> getAllCourses()
        {
            return courseRepository.GetAll().ToList();
        }

        public Course GetCourseByID(int id)
        {
            return courseRepository.GetById(id);
        }

        public Course GetCourseByName(string name)
        {
            return courseRepository.GetSingle(c => c.CourseName.Equals(name), c 
              => c.Teacher);
        }

        public void AddCourse(Course course)
        {
            courseRepository.Insert(course);
        }

        public void UpdateCourse(Course course)
        {
            courseRepository.Update(course);
        }

        public void RemoveCourse(Course course)
        {
            courseRepository.Delete(course);
        }

        public IList<Course> GetCoursesByTeacherID(int id)
        {
            return courseRepository.SearchFor(c => c.TeacherId ==
              id).ToList<Course>();
        }

        public IList<Course> GetCoursesByTeacherName(string name)
        {
            return courseRepository.SearchFor(c => c.Teacher.TeacherName ==
              name).ToList<Course>();
        }

        public IList<Student> GetStudentsByStandardID(int id)
        {
            return studentRepository.SearchFor(c => c.StandardId ==
              id).ToList<Student>();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using BusinessLayer;
using DataAccessLayer;
namespace Client
{
    public class Program
    {
        static BusinessLayer.BusinessLayer b1 = new
          BusinessLayer.BusinessLayer();
        public static void Main(string[] args)
        {
            MainMenu();
        }

        static void MainMenu()
        {
            string options = "1. Table Teacher" +
                "\n2. Table Courses" +
                "\n3. Table Standard" +
                "\n4. Table Student" +
                "\n5. Exit Program";
            int entry;
            Console.WriteLine(options);
            Console.Write("\nSelect an option: ");
            entry = ValidInt();
            while (entry != 5)
            {
                switch (entry)
                {
                    case 1: TeacherMenu();
                        break;
                    case 2: CoursesMenu();
                        break;
                    case 3: StandardMenu();
                        break;
                    case 4: StudentMenu();
                        break;
                    case 5:
                        break;
                    default: Console.WriteLine("Invalid entry. Must be between
                     1 and 5");
                        break;
                }
                Console.WriteLine(options);
                Console.Write("\nSelect an option: ");
                entry = ValidInt();
            }

        }

        static void TeacherMenu()
        {
            string options = "1. Create teacher" +
                "\n2. Delete Teacher" +
                "\n3. Update teacher by searching id" +
                "\n4. Update teacher by searching by name" +
                "\n5. Show all teachers" +
                "\n6. Display courses that have a teacher ID" +
                "\n7. Exit Teacher Table";
            Console.WriteLine(options);
            Console.Write("\nSelect an option: ");
            int entry;
            entry = ValidInt();
            while (entry != 7)
            {
                switch (entry)
                {
                    case 1: ClientCreatesTeacher();
                        break;
                    case 2: ClientDeletesTeacher();
                        break;
                    case 3: UpdateTeacherBySearchingID();
                        break;
                    case 4: UpdateTeacherBySearchingName();
                        break;
                    case 5: ShowAllTeachers();
                        break;
                    case 6: DisplayCoursesWithTeacherID();
                        break;
                    case 7:
                        break;
                    default: Console.WriteLine("ERROR: Invalid option. Entry 
                    must be between 1 and 7.");
                        break;
                }
                Console.WriteLine(options);
                Console.Write("\nSelect an option: ");
                entry = ValidInt();
            }
        }

        static void CoursesMenu()
        {
            string options = "1. Create a new course" +
                "\n2. Update a course by searching id" +
                "\n3. Update a course by searching course name" +
                "\n4. Delete a course" +
                "\n5. Display all courses" +
                "\n6. Exit Courses Table";
            int entry;
            Console.WriteLine(options);
            Console.Write("\nSelect an option: ");
            entry = ValidInt();
            while (entry != 6)
            {
                switch (entry)
                {
                    case 1: ClientCreatesCourse();
                        break;
                    case 2: UpdateCourseBySearchingID();
                        break;
                    case 3: UpdateCourseBySearchingName();
                        break;
                    case 4: ClientDeletesCourse();
                        break;
                    case 5: ShowAllCourses();
                        break;
                    case 6:
                        break;
                    default: Console.WriteLine("ERROR: Invalid option. Entry 
                    must be between 1 and 6.");
                        break;
                }
                Console.WriteLine(options);
                Console.Write("\nSelect an option: ");
                entry = ValidInt();
            }
        }

        static void StandardMenu()
        {
            string options = "1. Create Standard" +
                "\n2. Delete Standard" +
                "\n3. Update standard by searching id" +
                "\n4. Update standard by searching by name" +
                "\n5. Show all standard" +
                "\n6. Display students that have a standard ID" +
                "\n7. Exit Standard Table";
            Console.WriteLine(options);
            Console.Write("\nSelect an option: ");
            int entry;
            entry = ValidInt();
            while (entry != 7)
            {
                switch (entry)
                {
                    case 1: ClientCreatesStandard();
                        break;
                    case 2: ClientDeletesStandard();
                        break;
                    case 3: UpdateStandardBySearchingID();
                        break;
                    case 4: UpdateStandardBySearchingName();
                        break;
                    case 5: ShowAllStandards();
                        break;
                    case 6: ShowStudentsWithStandardID();
                        break;
                    case 7:
                        break;
                    default: Console.WriteLine("ERROR: Invalid option. Entry 
                    must be between 1 and 7.");
                        break;
                }
                Console.WriteLine(options);
                Console.Write("\nSelect an option: ");
                entry = ValidInt();
            }
        }

        static void StudentMenu()
        {
            string options = "1. Create a new student" +
                "\n2. Update a student by searching id" +
                "\n3. Update a student by searching name" +
                "\n4. Delete a student" +
                "\n5. Display all students" +
                "\n6. Exit Students Table";
            int entry;
            Console.WriteLine(options);
            Console.Write("\nSelect an option: ");
            entry = ValidInt();
            while (entry != 6)
            {
                switch (entry)
                {
                    case 1: ClientCreatesStudent();
                        break;
                    case 2: UpdateStudentBySearchingID();
                        break;
                    case 3: UpdateStudentBySearchingName();
                        break;
                    case 4: ClientDeletesStudent();
                        break;
                    case 5: DisplayAllStudents();
                        break;
                    case 6:
                        break;
                    default:
                        Console.WriteLine("ERROR: Invalid option. Entry must 
                    be between 1 and 6.");
                        break;
                }
                Console.WriteLine(options);
                Console.Write("\nSelect an option: ");
                entry = ValidInt();
            }
        }

        static void ShowAllTeachers()
        {
            List<Teacher> teacherList = b1.getAllTeachers().ToList();
            foreach(Teacher t in teacherList)
            {
                DisplayTeacher(t);
            }
        }

        static void ClientCreatesTeacher()
        {
            List<Teacher> tList = b1.getAllTeachers().ToList();
            int maxID = tList.Max(x => x.TeacherId);
            //Console.WriteLine("last id: " + maxID);
            int newTeacherID = maxID + 1;
            Console.Write("\nEnter the new teacher name: ");
            string newTeacherName = Console.ReadLine();
            Console.Write("\nEnter a standard ID for this teacher: ");
            int standardIDEntry = ValidInt();
            Standard selectedStandard = b1.GetStandardByID(standardIDEntry);
            if (selectedStandard != null)
            {
                Teacher teacher = new Teacher()
                {
                    TeacherId = newTeacherID,
                    TeacherName = newTeacherName,
                    StandardId = standardIDEntry,
                    Standard = null
                };
                //b1.AddTeacher(teacher);
                selectedStandard.Teachers.Add(teacher);
                b1.UpdateStandard(selectedStandard);
```

```
                    b1.AddTeacher(teacher);
                    DisplayTeacher(teacher);
                }
                else
                {
                    Teacher teacher = new Teacher()
                    {
                        TeacherId = newTeacherID,
                        TeacherName = newTeacherName,
                        StandardId = standardIDEntry,
                        Standard = null
                    };
                    DisplayTeacher(teacher);
                }
            }

            static void ClientDeletesTeacher()
            {
                ShowAllTeachers();
                Console.Write("\nEnter the id of the teacher you would like to
                  delete: ");
                int entry;
                entry = ValidInt();
                Teacher selectedTeacher = b1.GetTeacherByID(entry);
                if (selectedTeacher != null)
                {
                    DisplayTeacher(selectedTeacher);
                    int tStandardID = (int)selectedTeacher.StandardId;
                    Standard s = b1.GetStandardByID(tStandardID);
                    s.Teachers.Remove(selectedTeacher);
                    b1.UpdateStandard(s);
                    selectedTeacher.Courses.Clear();
                    b1.UpdateTeacher(selectedTeacher);
                    b1.RemoveTeacher(selectedTeacher);
                    Console.WriteLine("");
                }
                else
                {
                    Console.WriteLine("A teacher with that ID does not exist.");
                }
            }

            static void UpdateTeacherBySearchingID()
            {
                int entry;
                Console.Write("\nEnter the teacher id: ");
                entry = ValidInt();
                Teacher selectedTeacher = b1.GetTeacherByID(entry);
                if (selectedTeacher != null)
                {
                    DisplayTeacher(selectedTeacher);
                    string options = "1. Change standard" +
                        "\n2. Remove standard" +
                        "\n3. Change teacher name" +
                        "\n4. Return to teacher menu";
                    Console.WriteLine(options);
                    Console.Write("\nSelect an option: ");
                    entry = ValidInt();
                    string nameEntry;
                    int standardEntry;
                    Standard selectedStandard;
                    switch (entry)
                    {
                        case 1:
                            Console.Write("\nEnter the new standard id: ");
                            standardEntry = ValidInt();
                            selectedStandard = b1.GetStandardByID(standardEntry);
                            if (selectedStandard != null)
                            {
                                selectedStandard.Teachers.Add(selectedTeacher);
                                b1.UpdateStandard(selectedStandard);
                                selectedTeacher.StandardId = standardEntry;
                                b1.UpdateTeacher(selectedTeacher);
                            }
                            else
                            {
                                selectedTeacher.StandardId = standardEntry;
                                b1.UpdateTeacher(selectedTeacher);
                            }
                            break;
                        case 2:
                            selectedStandard = b1.GetStandardByID((int)
                        selectedTeacher.StandardId);
                            if (selectedStandard != null)
                            {
                                selectedStandard.Teachers.Remove(selectedTeacher);
                                b1.UpdateStandard(selectedStandard);
                                selectedTeacher.StandardId =  null;
                                b1.UpdateTeacher(selectedTeacher);
                            }
                            else
                            {
                                selectedTeacher.StandardId =  null;
                                b1.UpdateTeacher(selectedTeacher);
                            }
                            break;
                        case 3:
                            Console.Write("\nEnter the teacher's new name: ");
```

```csharp
                        nameEntry = Console.ReadLine();
                        selectedTeacher.TeacherName = nameEntry;
                        b1.UpdateTeacher(selectedTeacher);
                        foreach (Course c in selectedTeacher.Courses.ToList())
                        {
                            b1.UpdateCourse(c);
                        }
                        selectedStandard = b1.GetStandardByID((int)selectedTeacher.StandardId);
                        if (selectedStandard != null)
                        {
                            b1.UpdateStandard(selectedStandard);
                        }
                        break;
                    case 4:
                        break;
                    default: Console.WriteLine("ERROR: Invalid option. Make sure the entry is between 1 and 4");
                        break;
                }
                DisplayTeacher(selectedTeacher);
            }
            else
            {
                Console.WriteLine("A teacher with this ID does not exist.");
            }
        }

        static void UpdateTeacherBySearchingName()
        {
            int entry;
            string teacherNameEntry;
            Console.Write("\nEnter the teacher name: ");
            teacherNameEntry = Console.ReadLine();
            Teacher selectedTeacher = b1.GetTeacherByName(teacherNameEntry);
            if (selectedTeacher != null)
            {
                string options = "1. Change standard" +
                    "\n2. Remove standard" +
                    "\n3. Change teacher name" +
                    "\n4. Return to teacher menu";
                Console.WriteLine(options);
                Console.Write("\nSelect an option: ");
                entry = ValidInt();
                string nameEntry;
                int standardEntry;
                Standard selectedStandard;
                switch (entry)
                {
                    case 1:
                        Console.Write("\nEnter the new standard id: ");
                        standardEntry = ValidInt();
                        selectedStandard = b1.GetStandardByID(standardEntry);
                        if (selectedStandard != null)
                        {
                            selectedStandard.Teachers.Add(selectedTeacher);
                            b1.UpdateStandard(selectedStandard);
                            selectedTeacher.StandardId = standardEntry;
                            b1.UpdateTeacher(selectedTeacher);
                        }
                        else
                        {
                            selectedTeacher.StandardId = standardEntry;
                            b1.UpdateTeacher(selectedTeacher);
                        }
                        break;
                    case 2:
                        selectedStandard = b1.GetStandardByID((int)
                    selectedTeacher.StandardId);
                        if (selectedStandard != null)
                        {
                            selectedStandard.Teachers.Remove(selectedTeacher);
                            b1.UpdateStandard(selectedStandard);
                            selectedTeacher.StandardId = null;
                            b1.UpdateTeacher(selectedTeacher);
                        }
                        else
                        {
                            selectedTeacher.StandardId = null;
                            b1.UpdateTeacher(selectedTeacher);
                        }
                        break;
                    case 3:
                        Console.Write("\nEnter the teacher's new name: ");
                        nameEntry = Console.ReadLine();
                        selectedTeacher.TeacherName = nameEntry;
                        b1.UpdateTeacher(selectedTeacher);
                        foreach (Course c in selectedTeacher.Courses.ToList())
                        {
                            b1.UpdateCourse(c);
                        }
                        selectedStandard = b1.GetStandardByID((int)
                    selectedTeacher.StandardId);
                        if (selectedStandard != null)
                        {
                            b1.UpdateStandard(selectedStandard);
                        }
                        break;
                    case 4:
                        break;
                    default:
                        Console.WriteLine("ERROR: Invalid option. Make sure the entry is between 1 and 4");
                        break;
                }
                DisplayTeacher(selectedTeacher);
            }
            else
            {
                Console.WriteLine("A teacher with this ID does not exist.");
            }
        }

        static void DisplayCoursesWithTeacherID()
        {
            int entry;
            Console.Write("\nEnter the teacher ID: ");
            entry = ValidInt();
            List<Course> coursesList = b1.GetCoursesByTeacherID(entry).ToList();
            if (coursesList == null || coursesList.Count() == 0)
            {
                Console.WriteLine("There are no courses with that teacher ID");
            }
            else
            {
                foreach (Course c in coursesList)
                {
                    DisplayCourse(c);
                }
            }
        }

        static void ClientCreatesCourse()
        {
            string courseNameEntry;
            int teacherIDEntry;
            List<Course> cList = b1.getAllCourses().ToList();
            int maxID = cList.Max(x => x.CourseId);
            Console.Write("\nEnter a course name: ");
            courseNameEntry = Console.ReadLine();
            Console.Write("\nEnter the teacher id: ");
            teacherIDEntry = ValidInt();
            Teacher selectedTeacher = b1.GetTeacherByID(teacherIDEntry);
            Course course = new Course()
            {
                CourseId = maxID + 1,
                CourseName = courseNameEntry,
                TeacherId = teacherIDEntry
            };
            if (selectedTeacher != null)
            {
                selectedTeacher.Courses.Add(course);
                b1.UpdateTeacher(selectedTeacher);
                b1.AddCourse(course);
                DisplayCourse(course);
            }
            else
            {
                b1.AddCourse(course);
                DisplayCourse(course);
            }
        }

        static void ClientDeletesCourse()
        {
            int entry;
            Console.Write("\nEnter the ID of the course: ");
            entry = ValidInt();
            Course selectedCourse = b1.GetCourseByID(entry);
            if (selectedCourse != null)
            {
                DisplayCourse(selectedCourse);
                selectedCourse.Students.Clear();
                b1.UpdateCourse(selectedCourse);
                Teacher selectedTeacher = selectedCourse.Teacher;
                if (selectedTeacher != null)
                {
                    selectedTeacher.Courses.Remove(selectedCourse);
                    b1.UpdateTeacher(selectedTeacher);
                }
                b1.RemoveCourse(selectedCourse);
                Console.WriteLine("Course deleted\n");
            }
            else
            {
                Console.WriteLine("A course with that ID does not exist");
            }
        }

        static void UpdateCourseBySearchingID()
        {
            int entry;
            Console.Write("Enter the ID of the course: ");
            entry = ValidInt();
            Course selectedCourse = b1.GetCourseByID(entry);
            if (selectedCourse != null)
            {
                DisplayCourse(selectedCourse);
                string options = "1. Update course name" +
                    "\n2. Change teacher" +
                    "\n3. Remove teacher" +
                    "\n4. Exit update course";
                Console.WriteLine(options);
                Console.Write("\nEnter an option: ");
                entry = ValidInt();
                string nameEntry;
                int idEntry;
                Teacher selectedTeacher;
                switch (entry)
                {
                    case 1:
                        Console.Write("\nEnter the new course name: ");
                        nameEntry = Console.ReadLine();
                        selectedCourse.CourseName = nameEntry;
                        b1.UpdateCourse(selectedCourse);
                        break;
                    case 2:
                        Console.Write("\nEnter the new teacher id: ");
                        idEntry = ValidInt();
                        selectedTeacher = b1.GetTeacherByID(idEntry);
                        Teacher currentTeacher = selectedCourse.Teacher;
                        if (selectedTeacher != null)
                        {
                            if (currentTeacher != null)
                            {
                                currentTeacher.Courses.Remove(selectedCourse);
                                b1.UpdateTeacher(currentTeacher);
                            }
                            b1.UpdateCourse(selectedCourse);
                            selectedCourse.TeacherId = idEntry;
                            b1.UpdateCourse(selectedCourse);
                            selectedTeacher.Courses.Add(selectedCourse);
                            b1.UpdateTeacher(selectedTeacher);
                        }
                        else
                        {
                            if (currentTeacher != null)
                            {
                                currentTeacher.Courses.Remove(selectedCourse);
                                b1.UpdateTeacher(currentTeacher);
                            }
                            selectedCourse.TeacherId = idEntry;
                            selectedCourse.Teacher = null;
                            b1.UpdateCourse(selectedCourse);
                        }
                        break;
                    case 3:
                        selectedTeacher = selectedCourse.Teacher;
                        if (selectedTeacher != null)
                        {
                            selectedTeacher.Courses.Remove(selectedCourse);
                            b1.UpdateTeacher(selectedTeacher);
                        }
                        b1.UpdateCourse(selectedCourse);
                        break;
                    case 4:
                        break;
                    default: Console.WriteLine("ERROR: Invalid option. Entry must be between 1 and 4.");
                        break;
                }
                DisplayCourse(selectedCourse);
            }
            else
            {
                Console.WriteLine("A course with that ID does not exist.");
            }
        }

        static void UpdateCourseBySearchingName()
        {
            string courseNameEntry;
            int entry;
            Console.Write("Enter the name of the course: ");
            courseNameEntry = Console.ReadLine();
            Course selectedCourse = b1.GetCourseByName(courseNameEntry);
            if (selectedCourse != null)
            {
                DisplayCourse(selectedCourse);
                string options = "1. Update course name" +
                    "\n2. Change teacher" +
                    "\n3. Remove teacher" +
                    "\n4. Exit update course";
                Console.WriteLine(options);
                Console.Write("\nEnter an option: ");
                entry = ValidInt();
                string nameEntry;
                int idEntry;
                Teacher selectedTeacher;
                switch (entry)
                {
                    case 1:
                        Console.Write("\nEnter the new course name: ");
                        nameEntry = Console.ReadLine();
                        selectedCourse.CourseName = nameEntry;
                        b1.UpdateCourse(selectedCourse);
                        break;
                    case 2:
                        Console.Write("\nEnter the new teacher id: ");
                        idEntry = ValidInt();
                        selectedTeacher = b1.GetTeacherByID(idEntry);
                        Teacher currentTeacher = b1.GetTeacherByID((int)selectedCourse.TeacherId);
                        if (selectedTeacher != null)
                        {
                            if (currentTeacher != null)
                            {
                                currentTeacher.Courses.Remove(selectedCourse);
                                b1.UpdateTeacher(currentTeacher);
                            }
                            b1.UpdateCourse(selectedCourse);
                            selectedCourse.TeacherId = idEntry;
                            b1.UpdateCourse(selectedCourse);
                            selectedTeacher.Courses.Add(selectedCourse);
                            b1.UpdateTeacher(selectedTeacher);
                        }
                        else
                        {
                            if (currentTeacher != null)
                            {
                                currentTeacher.Courses.Remove(selectedCourse);
                                b1.UpdateTeacher(currentTeacher);
                            }
                            selectedCourse.TeacherId = idEntry;
                            selectedCourse.Teacher = null;
                            b1.UpdateCourse(selectedCourse);
                        }
                        break;
                    case 3:
                        selectedTeacher = selectedCourse.Teacher;
                        if (selectedTeacher != null)
                        {
                            selectedTeacher.Courses.Remove(selectedCourse);
                            b1.UpdateTeacher(selectedTeacher);
                        }
                        selectedCourse.TeacherId = null;
                        b1.UpdateCourse(selectedCourse);
                        break;
                    case 4:
                        break;
                    default:
                        Console.WriteLine("ERROR: Invalid option. Entry must be between 1 and 4.");
                        break;
                }
                DisplayCourse(selectedCourse);
            }
            else
            {
                Console.WriteLine("A course with that name does not exist.");
            }
        }

        static void ShowAllCourses()
        {
            List<Course> courseList = b1.getAllCourses().ToList();
            if (courseList == null || courseList.Count() == 0)
            {
                Console.WriteLine("There are no courses.");
            }
            else
            {
                foreach (Course c in courseList)
                {
                    DisplayCourse(c);
                }
            }
        }

        static void ClientCreatesStandard()
        {
            List<Standard> sList = b1.getAllStandards().ToList();
            int maxID = sList.Max(x => x.StandardId);
            string nameEntry;
            string descriptionEntry;
            Console.Write("\nEnter the name of the standard: ");
            nameEntry = Console.ReadLine();
            Console.Write("\nEnter a description for the standard: ");
            descriptionEntry = Console.ReadLine();
            Standard standard = new Standard()
            {
                StandardId = maxID + 1,
                StandardName = nameEntry,
                Description = descriptionEntry
            };
            b1.AddStandard(standard);
            DisplayStandard(standard);
        }

        static void ClientDeletesStandard()
        {
            int entry;
            Console.Write("\nEnter the ID of the standard: ");
            entry = ValidInt();
            Standard selectedStandard = b1.GetStandardByID(entry);
            if (selectedStandard != null)
            {
                DisplayStandard(selectedStandard);
                b1.RemoveStandard(selectedStandard);
                List<Student> studentList = b1.getAllStudents().ToList();
                List<Teacher> teacherList = b1.getAllTeachers().ToList();
                foreach (Student s in studentList)
                {
                    if (s.StandardId == entry)
                    {
                        s.StandardId = null;
                    }
                    b1.UpdateStudent(s);
                }
                foreach (Teacher t in teacherList)
                {
                    if (t.StandardId == entry)
                    {
                        t.StandardId = null;
                    }
                    b1.UpdateTeacher(t);
                }
            }
            else
            {
                Console.WriteLine("A standard with that ID does not exist.");
            }
        }

        static void UpdateStandardBySearchingID()
        {
            string options = "1. Change standard name" +
                "\n2. Change description" +
                "\n3. Exit update standard";
            int entry;
            Console.Write("\nEnter the standard id: ");
            entry = ValidInt();
            Standard selectedStandard = b1.GetStandardByID(entry);
            if (selectedStandard != null)
            {
                DisplayStandard(selectedStandard);
                Console.WriteLine(options);
                Console.Write("\nSelect an option: ");
                entry = ValidInt();
                string nameEntry;
                string descriptionEntry;
                switch (entry)
                {
                    case 1:
                        Console.Write("\nEnter the new standard name: ");
                        nameEntry = Console.ReadLine();
                        selectedStandard.StandardName = nameEntry;
                        b1.UpdateStandard(selectedStandard);
                        foreach (Student s in selectedStandard.Students)
                        {
                            b1.UpdateStudent(s);
                        }
                        foreach (Teacher t in selectedStandard.Teachers)
                        {
                            b1.UpdateTeacher(t);
                        }
                        break;
                    case 2:
                        Console.WriteLine("Enter the new standard description: ");
                        descriptionEntry = Console.ReadLine();
                        selectedStandard.Description = descriptionEntry;
                        b1.UpdateStandard(selectedStandard);
                        foreach (Student s in selectedStandard.Students)
                        {
                            b1.UpdateStudent(s);
                        }
                        foreach (Teacher t in selectedStandard.Teachers)
                        {
                            b1.UpdateTeacher(t);
                        }
                        break;
                    case 3:
                        break;
                    default:Console.WriteLine("ERROR: Invalid option. Selection must be between 1 and 3.");
                        break;
                }
                DisplayStandard(selectedStandard);
            }
            else
            {
                Console.WriteLine("A standard with that ID does not exist");
            }
        }

        static void UpdateStandardBySearchingName()
        {
            string nameSearch;
            string options = "1. Change standard name" +
                "\n2. Change description" +
                "\n3. Exit update standard";
            int entry;
            Console.Write("\nEnter the standard name: ");
            nameSearch = Console.ReadLine();
            Standard selectedStandard = b1.GetStandardByName(nameSearch);
            if (selectedStandard != null)
            {
                DisplayStandard(selectedStandard);
                Console.WriteLine(options);
                Console.Write("\nSelect an option: ");
                entry = ValidInt();
                string nameEntry;
                string descriptionEntry;
                switch (entry)
                {
                    case 1:
                        Console.Write("\nEnter the new standard name: ");
                        nameEntry = Console.ReadLine();
                        selectedStandard.StandardName = nameEntry;
                        b1.UpdateStandard(selectedStandard);
                        foreach (Student s in selectedStandard.Students)
                        {
                            b1.UpdateStudent(s);
                        }
                        foreach (Teacher t in selectedStandard.Teachers)
                        {
                            b1.UpdateTeacher(t);
                        }
                        break;
                    case 2:
                        Console.WriteLine("Enter the new standard description: 
                     ");
                        descriptionEntry = Console.ReadLine();
                        selectedStandard.Description = descriptionEntry;
                        b1.UpdateStandard(selectedStandard);
                        foreach (Student s in selectedStandard.Students)
                        {
                            b1.UpdateStudent(s);
                        }
                        foreach (Teacher t in selectedStandard.Teachers)
                        {
                            b1.UpdateTeacher(t);
                        }
                        break;
                    case 3:
                        break;
                    default:
                        Console.WriteLine("ERROR: Invalid option. Selection must be between 1 and 3.");
                        break;
                }
                DisplayStandard(selectedStandard);
            }
            else
            {
                Console.WriteLine("A standard with that ID does not exist");
            }
        }

        static void ShowStudentsWithStandardID()
        {
            int entry;
            Console.Write("\nEnter the standard id: ");
            entry = ValidInt();
            List<Student> studentList = b1.GetStudentsByStandardID(entry).ToList();
            if (studentList == null || studentList.Count() == 0)
            {
                Console.WriteLine("There are no students with that standard ID");
            }
            else
            {
                foreach (Student s in studentList)
                {
                    DisplayStudent(s);
                }
            }
        }

        static void ShowAllStandards()
        {
            List<Standard> standardList = b1.getAllStandards().ToList();
            if (standardList == null || standardList.Count() == 0)
            {
                Console.WriteLine("There are no standards");
            }
            else
            {
                foreach (Standard s in standardList)
                {
                    DisplayStandard(s);
                }
            }
        }

        static void ClientCreatesStudent()
        {
            string nameEntry;
            int standardIDEntry;
            List<Student> sList = b1.getAllStudents().ToList();
            int maxID = sList.Max(x => x.StudentID);

            Console.Write("\nEnter the student name: ");
            nameEntry = Console.ReadLine();
            Console.Write("\nEnter the standard ID: ");
            standardIDEntry = ValidInt();
            Standard selectedStandard = b1.GetStandardByID(standardIDEntry);
            Student student = new Student()
            {
                StudentID = maxID + 1,
                StudentName = nameEntry,
                StandardId = standardIDEntry,
                Standard = null
            };
            b1.AddStudent(student);
            if (selectedStandard != null)
            {
                selectedStandard.Students.Add(student);
                b1.UpdateStandard(selectedStandard);
            }
            DisplayStudent(student);

        }

        static void ClientDeletesStudent()
        {
            int entry;
            Console.Write("\nEnter the student ID: ");
            entry = ValidInt();
            Student selectedStudent = b1.GetStudentByID(entry);
            if (selectedStudent != null)
            {
                Standard studentStandard = selectedStudent.Standard;
                studentStandard.Students.Remove(selectedStudent);
                b1.UpdateStandard(studentStandard);
                List<Course> cList = b1.getAllCourses().ToList();
                foreach (Course c in cList)
                {
                    if (c.Students.Contains(selectedStudent))
                    {
                        c.Students.Remove(selectedStudent);
                    }
                    b1.UpdateCourse(c);
                }
                b1.RemoveStudent(selectedStudent);
            }
            else
            {
                Console.WriteLine("There is no student with that ID.");
            }
        }

        static void UpdateStudentBySearchingID()
        {
            int entry;
            Console.Write("\nEnter the id of the student: ");
            entry = ValidInt();
            string options = "1. Change student name" +
                "\n2. Change standard" +
                "\n3. Remove standard" +
                "\n4. Exit update student";
            Student selectedStudent = b1.GetStudentByID(entry);

            if (selectedStudent != null)
            {
                DisplayStudent(selectedStudent);
                Console.WriteLine(options);
                Console.Write("Select an option: ");
                entry = ValidInt();
                string nameEntry;
                Standard selectedStandard;
                Standard currentStandard;
                switch (entry)
                {
                    case 1:
                        Console.Write("\nEnter the student's new name: ");
                        nameEntry = Console.ReadLine();
                        selectedStudent.StudentName = nameEntry;
                        b1.UpdateStudent(selectedStudent);
                        break;
                    case 2:
                        Console.Write("\nEnter the id of the new standard: ");
                        entry = ValidInt();
                        currentStandard = selectedStudent.Standard;
                        selectedStandard = b1.GetStandardByID(entry);
                        if (selectedStandard != null)
                        {
                            if (currentStandard != null)
                            {
                                currentStandard.Students.Remove
                    (selectedStudent);
                                b1.UpdateStandard(currentStandard);
                                b1.UpdateStudent(selectedStudent);
                            }
                            selectedStudent.StandardId = entry;
                            selectedStandard.Students.Add(selectedStudent);
                            b1.UpdateStandard(selectedStandard);
                            b1.UpdateStudent(selectedStudent);
                        }
                        else
                        {
                            if (currentStandard != null)
                            {
                                currentStandard.Students.Remove(selectedStudent);
                                b1.UpdateStandard(currentStandard);
                                b1.UpdateStudent(selectedStudent);
                            }
                            selectedStudent.StandardId = entry;
                            b1.UpdateStudent(selectedStudent);
                        }
                        break;
                    case 3:

                        currentStandard = selectedStudent.Standard;
                        currentStandard.Students.Remove(selectedStudent);
                        b1.UpdateStandard(currentStandard);
                        selectedStudent.StandardId =  null;
                        b1.UpdateStudent(selectedStudent);
                        break;
                    case 4:
                        break;
                    default:
                        Console.WriteLine("ERROR: Invalid input. Selection must be between 1 and 4");
                        break;
                }
                DisplayStudent(selectedStudent);
            }
            else
            {
                Console.WriteLine("A student with that ID does not exist.");
            }

        }

        static void UpdateStudentBySearchingName()
        {
            string nameSearchEntry;
            int entry;
            Console.Write("\nEnter the name of the student: ");
```

```
                nameSearchEntry = Console.ReadLine();
                string options = "1. Change student name" +
                    "\n2. Change standard" +
                    "\n3. Remove standard" +
                    "\n4. Exit update student";
                Student selectedStudent = b1.GetStudentByName(nameSearchEntry);

                if (selectedStudent != null)
                {
                    DisplayStudent(selectedStudent);
                    Console.WriteLine(options);
                    Console.Write("Select an option: ");
                    entry = ValidInt();
                    string nameEntry;
                    Standard selectedStandard;
                    Standard currentStandard;
                    switch (entry)
                    {
                        case 1:
                            Console.Write("\nEnter the student's new name: ");
                            nameEntry = Console.ReadLine();
                            selectedStudent.StudentName = nameEntry;
                            b1.UpdateStudent(selectedStudent);
                            break;
                        case 2:
                            Console.Write("\nEnter the id of the new standard: ");
                            entry = ValidInt();
                            currentStandard = selectedStudent.Standard;
                            selectedStandard = b1.GetStandardByID(entry);
                            if (selectedStandard != null)
                            {
                                if (currentStandard != null)
                                {
                                    currentStandard.Students.Remove(selectedStudent);
                                    b1.UpdateStandard(currentStandard);
                                    b1.UpdateStudent(selectedStudent);
                                }
                                selectedStudent.StandardId = entry;
                                selectedStandard.Students.Add(selectedStudent);
                                b1.UpdateStandard(selectedStandard);
                                b1.UpdateStudent(selectedStudent);
                            }
                            else
                            {
                                if (currentStandard != null)
                                {
                                    currentStandard.Students.Remove(selectedStudent);
                                    b1.UpdateStandard(currentStandard);
                                    b1.UpdateStudent(selectedStudent);
                                }
                                selectedStudent.StandardId = entry;
                                b1.UpdateStudent(selectedStudent);
                            }
                            break;
                        case 3:

                            currentStandard = selectedStudent.Standard;
                            currentStandard.Students.Remove(selectedStudent);
                            b1.UpdateStandard(currentStandard);
                            selectedStudent.StandardId =  null;
                            b1.UpdateStudent(selectedStudent);
                            break;
                        case 4:
                            break;
                        default:
                            Console.WriteLine("ERROR: Invalid input. Selection must be between 1 and 4");
                            break;
                    }
                    DisplayStudent(selectedStudent);
                }
                else
                {
                    Console.WriteLine("A student with that ID does not exist.");
                }
            }

            static void DisplayAllStudents()
            {
                List<Student> studentList = b1.getAllStudents().ToList();
                if (studentList == null || studentList.Count() == 0)
                {
                    Console.WriteLine("There are no students");
                }
                else
                {
                    foreach (Student s in studentList)
                    {
                        DisplayStudent(s);
                    }
                }
            }

            public static void DisplayCourse(Course course)
            {
                string blankSpace = " ";
                if (course.Location == null && course.TeacherId == null)
                {
                    Console.WriteLine("Course ID: {0} \t| Course Name: {1} \t| 
                      Location: {2} \t| Teacher ID: {3} \t| Teacher: {4}",
                        course.CourseId,
                        course.CourseName,
                        blankSpace,
                        blankSpace,
                        blankSpace
                        );
                }
                else if (course.Teacher == null)
                {
                    Console.WriteLine("Course ID: {0} \t| Course Name: {1} \t| 
                      Location: {2} \t| Teacher ID: {3} \t| Teacher: {4}",
                        course.CourseId,
                        course.CourseName,
                        course.Location,
                        course.TeacherId,
                        blankSpace
                        );
                }
                else if (course.Location == null)
                {
                    Console.WriteLine("Course ID: {0} \t| Course Name: {1} \t| 
                      Location: {2} \t| Teacher ID: {3} \t| Teacher: {4}",
                        course.CourseId,
                        course.CourseName,
                        blankSpace,
                        course.TeacherId,
                        course.Teacher.TeacherName
                        );
                }
                else
                {
                    Console.WriteLine("Course ID: {0} \t| Course Name: {1} \t| 
                      Location: {2} \t| Teacher ID: {3} \t| Teacher: {4}",
                        course.CourseId,
                        course.CourseName,
                        course.Location,
                        course.TeacherId,
                        course.Teacher.TeacherName
                        );
                }
            }

            public static void DisplayTeacher(Teacher teacher)
            {
                if (teacher.Standard != null)
                {
                    Console.WriteLine("Teacher ID: {0} \t| Name: {1} \t| Standard ID: {2} \t| Standard: {3}",
                        teacher.TeacherId,
                        teacher.TeacherName,
                        teacher.StandardId,
                        teacher.Standard.StandardName
                     );
                }
                else if (teacher.StandardId == null)
                {
                    Console.WriteLine("Teacher ID: {0} \t| Name: {1} \t| Standard ID: {2} \t| Standard: {3}",
                        teacher.TeacherId,
                        teacher.TeacherName,
                        " ",
                        " "
                     );
                }
                else
                {
                    Console.WriteLine("Teacher ID: {0} \t| Name: {1} \t| Standard ID: {2} \t| Standard: {3}",
                        teacher.TeacherId,
                        teacher.TeacherName,
                        teacher.StandardId,
                        " "
                     );
                }
            }

            public static void DisplayStandard(Standard standard)
            {
                if (standard.Description != null)
                {
                    Console.WriteLine("Standard ID: {0} \t| Name: {1} \t| Description: {2}",
                        standard.StandardId,
                        standard.StandardName,
                        standard.Description
                    );
                }
                else
                {
                    Console.WriteLine("Standard ID: {0} \t| Name: {1} \t| Description: {2}",
                        standard.StandardId,
                        standard.StandardName,
                        " "
                    );
                }
            }

            public static void DisplayStudent(Student student)
            {
                if (student.StandardId == null)
                {
                    Console.WriteLine("Student ID: {0} \t| Name: {1} \t| Standard ID: {2}",
                      student.StudentID,
                      student.StudentName,
                      " "
                      );
                }
                else
                {
                    Console.WriteLine("Student ID: {0} \t| Name: {1} \t| Standard ID: {2}",
                      student.StudentID,
                      student.StudentName,
                      student.StandardId
                    );
                }
            }

            public static int ValidInt()
            {
                int input;
                while (int.TryParse(Console.ReadLine(), out input) == false)
                {
                    Console.WriteLine("Invalid input. Must be an integer.");
                }
                return input;
            }
        }
    }
```