

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace DataAccessLayer
8 {
9     public class CourseRepository : Repository<Course>, ICourseRepository
10     {
11         public CourseRepository() : base(new SchoolDBEntities())
12         {
13
14         }
15     }
16 }
17
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using DataAccessLayer;
7
8 namespace BusinessLayer
9 {
10     public interface IBusinessLayer
11     {
12         IList<Standard> getAllStandards();
13         Standard GetStandardByID(int id);
14         Standard GetStandardByName(string name);
15         void AddStandard(Standard standard);
16         void UpdateStandard(Standard standard);
17         void RemoveStandard(Standard standard);
18
19         IList<Student> getAllStudents();
20         Student GetStudentByID(int id);
21         Student GetStudentByName(string name);
22         void AddStudent(Student student);
23         void UpdateStudent(Student student);
24         void RemoveStudent(Student student);
25
26         IList<Teacher> getAllTeachers();
27         Teacher GetTeacherByID(int id);
28         Teacher GetTeacherByName(string name);
29         void AddTeacher(Teacher teacher);
30         void UpdateTeacher(Teacher teacher);
31         void RemoveTeacher(Teacher teacher);
32
33         IList<Course> getAllCourses();
34         Course GetCourseByID(int id);
35         Course GetCourseByName(string name);
36         void AddCourse(Course course);
37         void UpdateCourse(Course course);
38         void RemoveCourse(Course course);
39
40         IList<Course> GetCoursesByTeacherID(int id);
41         IList<Course> GetCoursesByTeacherName(string name);
42
43         IList<Student> GetStudentsByStandardID(int id);
44     }
45 }
46
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace DataAccessLayer
8 {
9     public interface ICourseRepository : IRepository<Course>
10     {
11
12     }
13 }
14
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Linq.Expressions;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace DataAccessLayer
9 {
10     public interface IRepository<T> : IDisposable
11     {
12         void Insert(T entity);
13
14         void Delete(T entity);
15
16         void Update(T entity);
17
18         T GetById(int id);
19
20         IQueryable<T> SearchFor(Expression<Func<T, bool>> predicate);
21
22         IEnumerable<T> GetAll();
23
24         T GetSingle(Func<T, bool> where, params Expression<Func<T, object>>[]
           navigationProperties);
25     }
26 }
27
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace DataAccessLayer
8 {
9     public interface IStandardRepository : IRepository<Standard>
10    {
11
12    }
13 }
14
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace DataAccessLayer
8 {
9     public interface IStudentRepository : IRepository<Student>
10    {
11
12    }
13 }
14
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace DataAccessLayer
8 {
9     public interface ITeacherRepository : IRepository<Teacher>
10    {
11
12    }
13 }
14
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using System.Linq.Expressions;
7 using System.Data.Entity;
8 using SqlServerTypes;
9 using System.Data.Entity.SqlServer;
10 using System.Data.Entity.Infrastructure;
11
12 namespace DataAccessLayer
13 {
14     public class Repository<T> : IRepository<T> where T : class
15     {
16         protected DbContext context;
17         protected DbSet<T> dbset;
18         public Repository(DbContext datacontext)
19         {
20             //You can use the cpmt
21             this.context = datacontext;
22             dbset = context.Set<T>();
23             SqlProviderServices.SqlServerTypesAssemblyName =
24                 "Microsoft.SqlServer.Types, Version=14.0.0.0, Culture=neutral,
25                 PublicKeyToken=89845dcd8080cc91";
26         }
27
28         public void Insert(T entity)
29         {
30             //Use the context object and entity state to save the entity
31             try
32             {
33                 context.Entry(entity).State = EntityState.Added;
34                 context.SaveChanges();
35             }
36             catch (DbUpdateConcurrencyException e)
37             {
38                 e.Entries.Single().Reload();
39             }
40             catch (InvalidOperationException e2) { }
41         }
42
43         public void Delete(T entity)
44         {
45             //Use the context object and entity state to delete the entity
46             //SqlProviderServices.SqlServerTypesAssemblyName =
47                 "Microsoft.SqlServer.Types, Version=14.0.0.0, Culture=neutral,
48                 PublicKeyToken=89845dcd8080cc91";
49             //SqlServerTypes.Utilities.LoadNativeAssemblies
```



```
(AppDomain.CurrentDomain.BaseDirectory);

46
47     try
48     {
49         context.Entry(entity).State = EntityState.Deleted;
50         context.SaveChanges();
51     }
52     catch (DbUpdateConcurrencyException e)
53     {
54         e.Entries.Single().Reload();
55     }
56
57 }
58
59 public void Update(T entity)
60 {
61     //Use the context object and entity state to update the entity
62     //SqlServerTypes.Utilities.LoadNativeAssemblies
63     (AppDomain.CurrentDomain.BaseDirectory);
64     try
65     {
66         SqlProviderServices.SqlServerTypesAssemblyName =
67             "Microsoft.SqlServer.Types, Version=14.0.0.0,
68             Culture=neutral, PublicKeyToken=89845dcd8080cc91";
69         SqlServerTypes.Utilities.LoadNativeAssemblies
70         (AppDomain.CurrentDomain.BaseDirectory);
71         context.Entry(entity).State = EntityState.Modified;
72         context.SaveChanges();
73     }
74     catch (DbUpdateConcurrencyException e)
75     {
76         e.Entries.Single().Reload();
77     }
78     catch (InvalidOperationException e2)
79     {
80     }
81 }
82
83 public T GetById(int id)
84 {
85     try
86     {
87         return dbset.Find(id);
88     }
89     catch (InvalidOperationException e)
90     {
91         return null;
92     }
93 }
```

```
90     }
91 }
92
93 public IQueryable<T> SearchFor(Expression<Func<T, bool>> predicate)
94 {
95     return dbset.Where(predicate);
96     //return context.Where(predicate);
97 }
98
99 public IEnumerable<T> GetAll()
100 {
101     return dbset.ToList();
102 }
103
104 //This method will find the related records by passing two argument
105 //First argument: lambda expression to search a record such as d => d.StandardName.Equals(standardName) to search am record by standard name
106 //Second argument: navigation property that leads to the related records such as d => d.Students
107 //The method returns the related records that met the condition in the first argument.
108 //An example of the method GetStandardByName(string standardName)
109 //public Standard GetStandardByName(string standardName)
110 //{
111 //return _standardRepository.GetSingle(d => d.StandardName.Equals(standardName), d => d.Students);
112 //}
113 public T GetSingle(Func<T, bool> where, params Expression<Func<T, bool>>[] navigationProperties)
114 {
115     T item = null;
116     IQueryable<T> dbQuery = context.Set<T>();
117     foreach (Expression<Func<T, bool>> navigationProperty in navigationProperties)
118         dbQuery = dbQuery.Include<T, object>(navigationProperty);
119     item = dbQuery.AsNoTracking().FirstOrDefault(where);
120     return item;
121 }
122
123
124 public void Dispose()
125 {
126     throw new NotImplementedException();
127 }
128 }
129 }
130
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace DataAccessLayer
8 {
9     public class StudentRepository : Repository<Student>, IStudentRepository
10    {
11        public StudentRepository() : base(new SchoolDBEntities())
12        {
13
14        }
15    }
16 }
17
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace DataAccessLayer
8 {
9     public class StandardRepository : Repository<Standard>, IStandardRepository
10    {
11        public StandardRepository() : base(new SchoolDBEntities())
12        {
13        }
14    }
15 }
16
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace DataAccessLayer
8 {
9     public class TeacherRepository : Repository<Teacher>, ITeacherRepository
10     {
11         public TeacherRepository() : base(new SchoolDBEntities())
12         {
13         }
14     }
15 }
16
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using DataAccessLayer;
7
8 namespace BusinessLayer
9 {
10     public class BusinessLayer : IBusinessLayer
11     {
12         private readonly IStandardRepository standardRepository;
13         private readonly IStudentRepository studentRepository;
14         private readonly ITeacherRepository teacherRepository;
15         private readonly ICourseRepository courseRepository;
16
17         public BusinessLayer()
18         {
19             standardRepository = new StandardRepository();
20             studentRepository = new StudentRepository();
21             courseRepository = new CourseRepository();
22             teacherRepository = new TeacherRepository();
23         }
24
25         public void AddStandard(Standard standard)
26         {
27             standardRepository.Insert(standard);
28             //throw new NotImplementedException();
29         }
30
31         public void AddStudent(Student student)
32         {
33             studentRepository.Insert(student);
34             //throw new NotImplementedException();
35         }
36
37         public IList<Standard> getAllStandards()
38         {
39             return standardRepository.GetAll().ToList();
40         }
41
42         public IList<Student> getAllStudents()
43         {
44             return studentRepository.GetAll().ToList();
45             //throw new NotImplementedException();
46         }
47
48         public Standard GetStandardByID(int id)
49         {
```

```
50         return standardRepository.GetById(id);
51         //throw new NotImplementedException();
52     }
53
54     public Standard GetStandardByName(string name)
55     {
56         return standardRepository.GetSingle(s => s.StandardName.Equals      ↗
57             (name), s => s.Students, s => s.Teachers);
58     }
59
60     public Student GetStudentByID(int id)
61     {
62         return studentRepository.GetById(id);
63         //throw new NotImplementedException();
64     }
65
66     public Student GetStudentByName(string name)
67     {
68         return studentRepository.GetSingle(s => s.StudentName.Equals(name), ↗
69             s => s.StudentAddress);
70     }
71
72     public void RemoveStandard(Standard standard)
73     {
74         standardRepository.Delete(standard);
75         //throw new NotImplementedException();
76     }
77
78     public void RemoveStudent(Student student)
79     {
80         studentRepository.Delete(student);
81         //throw new NotImplementedException();
82     }
83
84     public void UpdateStandard(Standard standard)
85     {
86         standardRepository.Update(standard);
87         //throw new NotImplementedException();
88     }
89
90     public void UpdateStudent(Student student)
91     {
92         studentRepository.Update(student);
93         //throw new NotImplementedException();
94     }
95
96     public IList<Teacher> getAllTeachers()
97     {
98         return teacherRepository.GetAll().ToList();
99     }
```

```
97     }
98
99     public Teacher GetTeacherByID(int id)
100    {
101        return teacherRepository.GetById(id);
102    }
103
104     public Teacher GetTeacherByName(string name)
105    {
106        return teacherRepository.GetSingle(t => t.TeacherName.Equals(name), ↗
            t => t.Standard);
107    }
108
109     public void AddTeacher(Teacher teacher)
110    {
111        teacherRepository.Insert(teacher);
112    }
113
114     public void UpdateTeacher(Teacher teacher)
115    {
116        teacherRepository.Update(teacher);
117    }
118
119     public void RemoveTeacher(Teacher teacher)
120    {
121        teacherRepository.Delete(teacher);
122    }
123
124     public IList<Course> getAllCourses()
125    {
126        return courseRepository.GetAll().ToList();
127    }
128
129     public Course GetCourseByID(int id)
130    {
131        return courseRepository.GetById(id);
132    }
133
134     public Course GetCourseByName(string name)
135    {
136        return courseRepository.GetSingle(c => c.CourseName.Equals(name), c ↗
            => c.Teacher);
137    }
138
139     public void AddCourse(Course course)
140    {
141        courseRepository.Insert(course);
142    }
143
```



```
144     public void UpdateCourse(Course course)
145     {
146         courseRepository.Update(course);
147     }
148
149     public void RemoveCourse(Course course)
150     {
151         courseRepository.Delete(course);
152     }
153
154     public IList<Course> GetCoursesByTeacherID(int id)
155     {
156         return courseRepository.SearchFor(c => c.TeacherId == id).ToList<Course>();
157     }
158
159     public IList<Course> GetCoursesByTeacherName(string name)
160     {
161         return courseRepository.SearchFor(c => c.Teacher.TeacherName == name).ToList<Course>();
162     }
163
164     public IList<Student> GetStudentsByStandardID(int id)
165     {
166         return studentRepository.SearchFor(c => c.StandardId == id).ToList<Student>();
167     }
168 }
169 }
170
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using BusinessLayer;
7 using DataAccessLayer;
8 namespace Client
9 {
10     public class Program
11     {
12         static BusinessLayer.BusinessLayer b1 = new BusinessLayer.BusinessLayer();
13         public static void Main(string[] args)
14         {
15             MainMenu();
16         }
17
18         static void MainMenu()
19         {
20             string options = "1. Table Teacher" +
21                             "\n2. Table Courses" +
22                             "\n3. Table Standard" +
23                             "\n4. Table Student" +
24                             "\n5. Exit Program";
25             int entry;
26             Console.WriteLine(options);
27             Console.Write("\nSelect an option: ");
28             entry = ValidInt();
29             while (entry != 5)
30             {
31                 switch (entry)
32                 {
33                     case 1: TeacherMenu();
34                         break;
35                     case 2: CoursesMenu();
36                         break;
37                     case 3: StandardMenu();
38                         break;
39                     case 4: StudentMenu();
40                         break;
41                     case 5:
42                         break;
43                     default: Console.WriteLine("Invalid entry. Must be between
44                                     1 and 5");
45                         break;
46                 }
47                 Console.WriteLine(options);
48                 Console.Write("\nSelect an option: ");
49             }
50         }
51     }
52 }
```

```
48         entry = ValidInt();
49     }
50
51 }
52
53 static void TeacherMenu()
54 {
55     string options = "1. Create teacher" +
56         "\n2. Delete Teacher" +
57         "\n3. Update teacher by searching id" +
58         "\n4. Update teacher by searching by name" +
59         "\n5. Show all teachers" +
60         "\n6. Display courses that have a teacher ID" +
61         "\n7. Exit Teacher Table";
62     Console.WriteLine(options);
63     Console.Write("\nSelect an option: ");
64     int entry;
65     entry = ValidInt();
66     while (entry != 7)
67     {
68         switch (entry)
69         {
70             case 1: ClientCreatesTeacher();
71                 break;
72             case 2: ClientDeletesTeacher();
73                 break;
74             case 3: UpdateTeacherBySearchingID();
75                 break;
76             case 4: UpdateTeacherBySearchingName();
77                 break;
78             case 5: ShowAllTeachers();
79                 break;
80             case 6: DisplayCoursesWithTeacherID();
81                 break;
82             case 7:
83                 break;
84             default: Console.WriteLine("ERROR: Invalid option. Entry
85                 must be between 1 and 7.");
86                 break;
87         }
88         Console.WriteLine(options);
89         Console.Write("\nSelect an option: ");
90         entry = ValidInt();
91     }
92
93 static void CoursesMenu()
94 {
95     string options = "1. Create a new course" +
```

```
96         "\n2. Update a course by searching id" +
97         "\n3. Update a course by searching course name" +
98         "\n4. Delete a course" +
99         "\n5. Display all courses" +
100        "\n6. Exit Courses Table";
101    int entry;
102    Console.WriteLine(options);
103    Console.Write("\nSelect an option: ");
104    entry = ValidInt();
105    while (entry != 6)
106    {
107        switch (entry)
108        {
109            case 1: ClientCreatesCourse();
110                break;
111            case 2: UpdateCourseBySearchingID();
112                break;
113            case 3: UpdateCourseBySearchingName();
114                break;
115            case 4: ClientDeletesCourse();
116                break;
117            case 5: ShowAllCourses();
118                break;
119            case 6:
120                break;
121            default: Console.WriteLine("ERROR: Invalid option. Entry
122                must be between 1 and 6.");
123                break;
124        }
125        Console.WriteLine(options);
126        Console.Write("\nSelect an option: ");
127        entry = ValidInt();
128    }
129
130    static void StandardMenu()
131    {
132        string options = "1. Create Standard" +
133        "\n2. Delete Standard" +
134        "\n3. Update standard by searching id" +
135        "\n4. Update standard by searching by name" +
136        "\n5. Show all standard" +
137        "\n6. Display students that have a standard ID" +
138        "\n7. Exit Standard Table";
139        Console.WriteLine(options);
140        Console.Write("\nSelect an option: ");
141        int entry;
142        entry = ValidInt();
143        while (entry != 7)
```

```
144         {
145             switch (entry)
146             {
147                 case 1: ClientCreatesStandard();
148                     break;
149                 case 2: ClientDeletesStandard();
150                     break;
151                 case 3: UpdateStandardBySearchingID();
152                     break;
153                 case 4: UpdateStandardBySearchingName();
154                     break;
155                 case 5: ShowAllStandards();
156                     break;
157                 case 6: ShowStudentsWithStandardID();
158                     break;
159                 case 7:
160                     break;
161                 default: Console.WriteLine("ERROR: Invalid option. Entry
162                                     must be between 1 and 7.");
163                     break;
164             }
165             Console.WriteLine(options);
166             Console.Write("\nSelect an option: ");
167             entry = ValidInt();
168         }
169     }
170     static void StudentMenu()
171     {
172         string options = "1. Create a new student" +
173             "\n2. Update a student by searching id" +
174             "\n3. Update a student by searching name" +
175             "\n4. Delete a student" +
176             "\n5. Display all students" +
177             "\n6. Exit Students Table";
178         int entry;
179         Console.WriteLine(options);
180         Console.Write("\nSelect an option: ");
181         entry = ValidInt();
182         while (entry != 6)
183         {
184             switch (entry)
185             {
186                 case 1: ClientCreatesStudent();
187                     break;
188                 case 2: UpdateStudentBySearchingID();
189                     break;
190                 case 3: UpdateStudentBySearchingName();
191                     break;
```

```
192         case 4: ClientDeletesStudent();
193             break;
194         case 5: DisplayAllStudents();
195             break;
196         case 6:
197             break;
198         default:
199             Console.WriteLine("ERROR: Invalid option. Entry must
200             be between 1 and 6.");
201             break;
202     }
203     Console.WriteLine(options);
204     Console.Write("\nSelect an option: ");
205     entry = ValidInt();
206 }
207
208 static void ShowAllTeachers()
209 {
210     List<Teacher> teacherList = b1.getAllTeachers().ToList();
211     foreach(Teacher t in teacherList)
212     {
213         DisplayTeacher(t);
214     }
215 }
216
217 static void ClientCreatesTeacher()
218 {
219     List<Teacher> tList = b1.getAllTeachers().ToList();
220     int maxID = tList.Max(x => x.TeacherId);
221     //Console.WriteLine("last id: " + maxID);
222     int newTeacherID = maxID + 1;
223     Console.Write("\nEnter the new teacher name: ");
224     string newTeacherName = Console.ReadLine();
225     Console.Write("\nEnter a standard ID for this teacher: ");
226     int standardIDEntry = ValidInt();
227     Standard selectedStandard = b1.GetStandardByID(standardIDEntry);
228     if (selectedStandard != null)
229     {
230         Teacher teacher = new Teacher()
231         {
232             TeacherId = newTeacherID,
233             TeacherName = newTeacherName,
234             StandardId = standardIDEntry,
235             Standard = null
236         };
237         //b1.AddTeacher(teacher);
238         selectedStandard.Teachers.Add(teacher);
239         b1.UpdateStandard(selectedStandard);
```

```
240         b1.AddTeacher(teacher);
241         DisplayTeacher(teacher);
242     }
243     else
244     {
245         Teacher teacher = new Teacher()
246         {
247             TeacherId = newTeacherID,
248             TeacherName = newTeacherName,
249             StandardId = standardIDEntry,
250             Standard = null
251         };
252         DisplayTeacher(teacher);
253     }
254 }
255
256 static void ClientDeletesTeacher()
257 {
258     ShowAllTeachers();
259     Console.Write("\nEnter the id of the teacher you would like to delete: ");
260     int entry;
261     entry = ValidInt();
262     Teacher selectedTeacher = b1.GetTeacherByID(entry);
263     if (selectedTeacher != null)
264     {
265         DisplayTeacher(selectedTeacher);
266         int tStandardID = (int)selectedTeacher.StandardId;
267         Standard s = b1.GetStandardByID(tStandardID);
268         s.Teachers.Remove(selectedTeacher);
269         b1.UpdateStandard(s);
270         selectedTeacher.Courses.Clear();
271         b1.UpdateTeacher(selectedTeacher);
272         b1.RemoveTeacher(selectedTeacher);
273         Console.WriteLine("");
274     }
275     else
276     {
277         Console.WriteLine("A teacher with that ID does not exist.");
278     }
279 }
280
281 static void UpdateTeacherBySearchingID()
282 {
283     int entry;
284     Console.Write("\nEnter the teacher id: ");
285     entry = ValidInt();
286     Teacher selectedTeacher = b1.GetTeacherByID(entry);
287     if (selectedTeacher != null)
```

```
288     {
289         DisplayTeacher(selectedTeacher);
290         string options = "1. Change standard" +
291             "\n2. Remove standard" +
292             "\n3. Change teacher name" +
293             "\n4. Return to teacher menu";
294         Console.WriteLine(options);
295         Console.Write("\nSelect an option: ");
296         entry = ValidInt();
297         string nameEntry;
298         int standardEntry;
299         Standard selectedStandard;
300         switch (entry)
301         {
302             case 1:
303                 Console.Write("\nEnter the new standard id: ");
304                 standardEntry = ValidInt();
305                 selectedStandard = b1.GetStandardByID(standardEntry);
306                 if (selectedStandard != null)
307                 {
308                     selectedStandard.Teachers.Add(selectedTeacher);
309                     b1.UpdateStandard(selectedStandard);
310                     selectedTeacher.StandardId = standardEntry;
311                     b1.UpdateTeacher(selectedTeacher);
312                 }
313                 else
314                 {
315                     selectedTeacher.StandardId = standardEntry;
316                     b1.UpdateTeacher(selectedTeacher);
317                 }
318                 break;
319             case 2:
320                 selectedStandard = b1.GetStandardByID((int)
321                 selectedTeacher.StandardId);
322                 if (selectedStandard != null)
323                 {
324                     selectedStandard.Teachers.Remove(selectedTeacher);
325                     b1.UpdateStandard(selectedStandard);
326                     selectedTeacher.StandardId = null;
327                     b1.UpdateTeacher(selectedTeacher);
328                 }
329                 else
330                 {
331                     selectedTeacher.StandardId = null;
332                     b1.UpdateTeacher(selectedTeacher);
333                 }
334                 break;
335             case 3:
336                 Console.Write("\nEnter the teacher's new name: ");
```



```
336         nameEntry = Console.ReadLine();
337         selectedTeacher.TeacherName = nameEntry;
338         b1.UpdateTeacher(selectedTeacher);
339         foreach (Course c in selectedTeacher.Courses.ToList())
340         {
341             b1.UpdateCourse(c);
342         }
343         selectedStandard = b1.GetStandardByID((int)
selectedTeacher.StandardId);
344         if (selectedStandard != null)
345         {
346             b1.UpdateStandard(selectedStandard);
347         }
348         break;
349     case 4:
350         break;
351     default: Console.WriteLine("ERROR: Invalid option. Make
sure the entry is between 1 and 4");
352         break;
353     }
354     DisplayTeacher(selectedTeacher);
355 }
356 else
357 {
358     Console.WriteLine("A teacher with this ID does not exist.");
359 }
360 }
361
362 static void UpdateTeacherBySearchingName()
363 {
364     int entry;
365     string teacherNameEntry;
366     Console.Write("\nEnter the teacher name: ");
367     teacherNameEntry = Console.ReadLine();
368     Teacher selectedTeacher = b1.GetTeacherByName(teacherNameEntry);
369     if (selectedTeacher != null)
370     {
371         string options = "1. Change standard" +
372             "\n2. Remove standard" +
373             "\n3. Change teacher name" +
374             "\n4. Return to teacher menu";
375         Console.WriteLine(options);
376         Console.Write("\nSelect an option: ");
377         entry = ValidInt();
378         string nameEntry;
379         int standardEntry;
380         Standard selectedStandard;
381         switch (entry)
382         {
```

```
383         case 1:
384             Console.WriteLine("\nEnter the new standard id: ");
385             standardEntry = ValidInt();
386             selectedStandard = b1.GetStandardByID(standardEntry);
387             if (selectedStandard != null)
388             {
389                 selectedStandard.Teachers.Add(selectedTeacher);
390                 b1.UpdateStandard(selectedStandard);
391                 selectedTeacher.StandardId = standardEntry;
392                 b1.UpdateTeacher(selectedTeacher);
393             }
394             else
395             {
396                 selectedTeacher.StandardId = standardEntry;
397                 b1.UpdateTeacher(selectedTeacher);
398             }
399             break;
400         case 2:
401             selectedStandard = b1.GetStandardByID((int)
402             selectedTeacher.StandardId);
403             if (selectedStandard != null)
404             {
405                 selectedStandard.Teachers.Remove(selectedTeacher);
406                 b1.UpdateStandard(selectedStandard);
407                 selectedTeacher.StandardId = null;
408                 b1.UpdateTeacher(selectedTeacher);
409             }
410             else
411             {
412                 selectedTeacher.StandardId = null;
413                 b1.UpdateTeacher(selectedTeacher);
414             }
415             break;
416         case 3:
417             Console.WriteLine("\nEnter the teacher's new name: ");
418             nameEntry = Console.ReadLine();
419             selectedTeacher.TeacherName = nameEntry;
420             b1.UpdateTeacher(selectedTeacher);
421             foreach (Course c in selectedTeacher.Courses.ToList())
422             {
423                 b1.UpdateCourse(c);
424             }
425             selectedStandard = b1.GetStandardByID((int)
426             selectedTeacher.StandardId);
427             if (selectedStandard != null)
428             {
429                 b1.UpdateStandard(selectedStandard);
430             }
431             break;
```

```
430         case 4:
431             break;
432         default:
433             Console.WriteLine("ERROR: Invalid option. Make sure the entry is between 1 and 4");
434             break;
435     }
436     DisplayTeacher(selectedTeacher);
437 }
438 else
439 {
440     Console.WriteLine("A teacher with this ID does not exist.");
441 }
442 }
443
444 static void DisplayCoursesWithTeacherID()
445 {
446     int entry;
447     Console.Write("\nEnter the teacher ID: ");
448     entry = ValidInt();
449     List<Course> coursesList = b1.GetCoursesByTeacherID(entry).ToList();
450     if (coursesList == null || coursesList.Count() == 0)
451     {
452         Console.WriteLine("There are no courses with that teacher ID");
453     }
454     else
455     {
456         foreach (Course c in coursesList)
457         {
458             DisplayCourse(c);
459         }
460     }
461 }
462
463 static void ClientCreatesCourse()
464 {
465     string courseNameEntry;
466     int teacherIDEntry;
467     List<Course> cList = b1.getAllCourses().ToList();
468     int maxID = cList.Max(x => x.CourseId);
469     Console.Write("\nEnter a course name: ");
470     courseNameEntry = Console.ReadLine();
471     Console.Write("\nEnter the teacher id: ");
472     teacherIDEntry = ValidInt();
473     Teacher selectedTeacher = b1.GetTeacherByID(teacherIDEntry);
474     Course course = new Course()
475     {
```

```
476         CourseId = maxID + 1,
477         CourseName = courseNameEntry,
478         TeacherId = teacherIDEntry
479     };
480     if (selectedTeacher != null)
481     {
482         selectedTeacher.Courses.Add(course);
483         b1.UpdateTeacher(selectedTeacher);
484         b1.AddCourse(course);
485         DisplayCourse(course);
486     }
487     else
488     {
489         b1.AddCourse(course);
490         DisplayCourse(course);
491     }
492 }
493
494 static void ClientDeletesCourse()
495 {
496     int entry;
497     Console.Write("\nEnter the ID of the course: ");
498     entry = ValidInt();
499     Course selectedCourse = b1.GetCourseByID(entry);
500     if (selectedCourse != null)
501     {
502         DisplayCourse(selectedCourse);
503         selectedCourse.Students.Clear();
504         b1.UpdateCourse(selectedCourse);
505         Teacher selectedTeacher = selectedCourse.Teacher;
506         if (selectedTeacher != null)
507         {
508             selectedTeacher.Courses.Remove(selectedCourse);
509             b1.UpdateTeacher(selectedTeacher);
510         }
511         b1.RemoveCourse(selectedCourse);
512         Console.WriteLine("Course deleted\n");
513     }
514     else
515     {
516         Console.WriteLine("A course with that ID does not exist");
517     }
518 }
519
520 static void UpdateCourseBySearchingID()
521 {
522     int entry;
523     Console.Write("Enter the ID of the course: ");
524     entry = ValidInt();
```

```
525     Course selectedCourse = b1.GetCourseByID(entry);
526     if (selectedCourse != null)
527     {
528         DisplayCourse(selectedCourse);
529         string options = "1. Update course name" +
530             "\n2. Change teacher" +
531             "\n3. Remove teacher" +
532             "\n4. Exit update course";
533         Console.WriteLine(options);
534         Console.Write("\nEnter an option: ");
535         entry = ValidInt();
536         string nameEntry;
537         int idEntry;
538         Teacher selectedTeacher;
539         switch (entry)
540         {
541             case 1:
542                 Console.Write("\nEnter the new course name: ");
543                 nameEntry = Console.ReadLine();
544                 selectedCourse.CourseName = nameEntry;
545                 b1.UpdateCourse(selectedCourse);
546                 break;
547             case 2:
548                 Console.Write("\nEnter the new teacher id: ");
549                 idEntry = ValidInt();
550                 selectedTeacher = b1.GetTeacherByID(idEntry);
551                 Teacher currentTeacher = selectedCourse.Teacher;
552                 if (selectedTeacher != null)
553                 {
554                     if (currentTeacher != null)
555                     {
556                         currentTeacher.Courses.Remove(selectedCourse);
557                         b1.UpdateTeacher(currentTeacher);
558                     }
559                     b1.UpdateCourse(selectedCourse);
560                     selectedCourse.TeacherId = idEntry;
561                     b1.UpdateCourse(selectedCourse);
562                     selectedTeacher.Courses.Add(selectedCourse);
563                     b1.UpdateTeacher(selectedTeacher);
564                 }
565             else
566             {
567                 if (currentTeacher != null)
568                 {
569                     currentTeacher.Courses.Remove(selectedCourse);
570                     b1.UpdateTeacher(currentTeacher);
571                 }
572                 selectedCourse.TeacherId = idEntry;
573                 selectedCourse.Teacher = null;
```

```
574         b1.UpdateCourse(selectedCourse);
575     }
576     break;
577     case 3:
578         selectedTeacher = selectedCourse.Teacher;
579         if (selectedTeacher != null)
580         {
581             selectedTeacher.Courses.Remove(selectedCourse);
582             b1.UpdateTeacher(selectedTeacher);
583         }
584         b1.UpdateCourse(selectedCourse);
585         break;
586     case 4:
587         break;
588     default: Console.WriteLine("ERROR: Invalid option. Entry must be between 1 and 4.");
589         break;
590     }
591     DisplayCourse(selectedCourse);
592 }
593 else
594 {
595     Console.WriteLine("A course with that ID does not exist.");
596 }
597 }
598
599 static void UpdateCourseBySearchingName()
600 {
601     string courseNameEntry;
602     int entry;
603     Console.Write("Enter the name of the course: ");
604     courseNameEntry = Console.ReadLine();
605     Course selectedCourse = b1.GetCourseByName(courseNameEntry);
606     if (selectedCourse != null)
607     {
608         DisplayCourse(selectedCourse);
609         string options = "1. Update course name" +
610             "\n2. Change teacher" +
611             "\n3. Remove teacher" +
612             "\n4. Exit update course";
613         Console.WriteLine(options);
614         Console.Write("\nEnter an option: ");
615         entry = ValidInt();
616         string nameEntry;
617         int idEntry;
618         Teacher selectedTeacher;
619         switch (entry)
620         {
621             case 1:
```

```
622         Console.WriteLine("\nEnter the new course name: ");
623         nameEntry = Console.ReadLine();
624         selectedCourse.CourseName = nameEntry;
625         b1.UpdateCourse(selectedCourse);
626         break;
627     case 2:
628         Console.WriteLine("\nEnter the new teacher id: ");
629         idEntry = ValidInt();
630         selectedTeacher = b1.GetTeacherByID(idEntry);
631         Teacher currentTeacher = b1.GetTeacherByID((int) selectedCourse.TeacherId);
632         if (selectedTeacher != null)
633         {
634             if (currentTeacher != null)
635             {
636                 currentTeacher.Courses.Remove(selectedCourse);
637                 b1.UpdateTeacher(currentTeacher);
638             }
639             b1.UpdateCourse(selectedCourse);
640             selectedCourse.TeacherId = idEntry;
641             b1.UpdateCourse(selectedCourse);
642             selectedTeacher.Courses.Add(selectedCourse);
643             b1.UpdateTeacher(selectedTeacher);
644         }
645         else
646         {
647             if (currentTeacher != null)
648             {
649                 currentTeacher.Courses.Remove(selectedCourse);
650                 b1.UpdateTeacher(currentTeacher);
651             }
652             selectedCourse.TeacherId = idEntry;
653             selectedCourse.Teacher = null;
654             b1.UpdateCourse(selectedCourse);
655         }
656         break;
657     case 3:
658         selectedTeacher = selectedCourse.Teacher;
659         if (selectedTeacher != null)
660         {
661             selectedTeacher.Courses.Remove(selectedCourse);
662             b1.UpdateTeacher(selectedTeacher);
663         }
664         selectedCourse.TeacherId = null;
665         b1.UpdateCourse(selectedCourse);
666         break;
667     case 4:
668         break;
669     default:
```

```
670         Console.WriteLine("ERROR: Invalid option. Entry must
        be between 1 and 4.");
671         break;
672     }
673     DisplayCourse(selectedCourse);
674 }
675 else
676 {
677     Console.WriteLine("A course with that name does not exist.");
678 }
679 }
680
681 static void ShowAllCourses()
682 {
683     List<Course> courseList = b1.getAllCourses().ToList();
684     if (courseList == null || courseList.Count() == 0)
685     {
686         Console.WriteLine("There are no courses.");
687     }
688     else
689     {
690         foreach (Course c in courseList)
691         {
692             DisplayCourse(c);
693         }
694     }
695 }
696
697 static void ClientCreatesStandard()
698 {
699     List<Standard> sList = b1.getAllStandards().ToList();
700     int maxID = sList.Max(x => x.StandardId);
701     string nameEntry;
702     string descriptionEntry;
703     Console.Write("\nEnter the name of the standard: ");
704     nameEntry = Console.ReadLine();
705     Console.Write("\nEnter a description for the standard: ");
706     descriptionEntry = Console.ReadLine();
707     Standard standard = new Standard()
708     {
709         StandardId = maxID + 1,
710         StandardName = nameEntry,
711         Description = descriptionEntry
712     };
713     b1.AddStandard(standard);
714     DisplayStandard(standard);
715 }
716
717 static void ClientDeletesStandard()
```



```
718     {
719         int entry;
720         Console.Write("\nEnter the ID of the standard: ");
721         entry = ValidInt();
722         Standard selectedStandard = b1.GetStandardByID(entry);
723         if (selectedStandard != null)
724         {
725             DisplayStandard(selectedStandard);
726             b1.RemoveStandard(selectedStandard);
727             List<Student> studentList = b1.getAllStudents().ToList();
728             List<Teacher> teacherList = b1.getAllTeachers().ToList();
729             foreach (Student s in studentList)
730             {
731                 if (s.StandardId == entry)
732                 {
733                     s.StandardId = null;
734                 }
735                 b1.UpdateStudent(s);
736             }
737             foreach (Teacher t in teacherList)
738             {
739                 if (t.StandardId == entry)
740                 {
741                     t.StandardId = null;
742                 }
743                 b1.UpdateTeacher(t);
744             }
745         }
746         else
747         {
748             Console.WriteLine("A standard with that ID does not exist.");
749         }
750     }
751
752     static void UpdateStandardBySearchingID()
753     {
754         string options = "1. Change standard name" +
755             "\n2. Change description" +
756             "\n3. Exit update standard";
757         int entry;
758         Console.Write("\nEnter the standard id: ");
759         entry = ValidInt();
760         Standard selectedStandard = b1.GetStandardByID(entry);
761         if (selectedStandard != null)
762         {
763             DisplayStandard(selectedStandard);
764             Console.WriteLine(options);
765             Console.Write("\nSelect an option: ");
766             entry = ValidInt();
```

```
767         string nameEntry;
768         string descriptionEntry;
769         switch (entry)
770         {
771             case 1:
772                 Console.WriteLine("\nEnter the new standard name: ");
773                 nameEntry = Console.ReadLine();
774                 selectedStandard.StandardName = nameEntry;
775                 b1.UpdateStandard(selectedStandard);
776                 foreach (Student s in selectedStandard.Students)
777                 {
778                     b1.UpdateStudent(s);
779                 }
780                 foreach (Teacher t in selectedStandard.Teachers)
781                 {
782                     b1.UpdateTeacher(t);
783                 }
784                 break;
785             case 2:
786                 Console.WriteLine("Enter the new standard description: ");
787                 descriptionEntry = Console.ReadLine();
788                 selectedStandard.Description = descriptionEntry;
789                 b1.UpdateStandard(selectedStandard);
790                 foreach (Student s in selectedStandard.Students)
791                 {
792                     b1.UpdateStudent(s);
793                 }
794                 foreach (Teacher t in selectedStandard.Teachers)
795                 {
796                     b1.UpdateTeacher(t);
797                 }
798                 break;
799             case 3:
800                 break;
801             default: Console.WriteLine("ERROR: Invalid option.
802                 Selection must be between 1 and 3.");
803                 break;
804         }
805         DisplayStandard(selectedStandard);
806     else
807     {
808         Console.WriteLine("A standard with that ID does not exist");
809     }
810 }
811
812 static void UpdateStandardBySearchingName()
813 {
```

```
814     string nameSearch;
815     string options = "1. Change standard name" +
816         "\n2. Change description" +
817         "\n3. Exit update standard";
818     int entry;
819     Console.Write("\nEnter the standard name: ");
820     nameSearch = Console.ReadLine();
821     Standard selectedStandard = b1.GetStandardByName(nameSearch);
822     if (selectedStandard != null)
823     {
824         DisplayStandard(selectedStandard);
825         Console.WriteLine(options);
826         Console.Write("\nSelect an option: ");
827         entry = ValidInt();
828         string nameEntry;
829         string descriptionEntry;
830         switch (entry)
831         {
832             case 1:
833                 Console.Write("\nEnter the new standard name: ");
834                 nameEntry = Console.ReadLine();
835                 selectedStandard.StandardName = nameEntry;
836                 b1.UpdateStandard(selectedStandard);
837                 foreach (Student s in selectedStandard.Students)
838                 {
839                     b1.UpdateStudent(s);
840                 }
841                 foreach (Teacher t in selectedStandard.Teachers)
842                 {
843                     b1.UpdateTeacher(t);
844                 }
845                 break;
846             case 2:
847                 Console.WriteLine("Enter the new standard description: ");
848                 descriptionEntry = Console.ReadLine();
849                 selectedStandard.Description = descriptionEntry;
850                 b1.UpdateStandard(selectedStandard);
851                 foreach (Student s in selectedStandard.Students)
852                 {
853                     b1.UpdateStudent(s);
854                 }
855                 foreach (Teacher t in selectedStandard.Teachers)
856                 {
857                     b1.UpdateTeacher(t);
858                 }
859                 break;
860             case 3:
861                 break;
```

```
862         default:
863             Console.WriteLine("ERROR: Invalid option. Selection
864                                 must be between 1 and 3.");
865             break;
866         }
867         DisplayStandard(selectedStandard);
868     }
869     else
870     {
871         Console.WriteLine("A standard with that ID does not exist");
872     }
873 }
874 static void ShowStudentsWithStandardID()
875 {
876     int entry;
877     Console.Write("\nEnter the standard id: ");
878     entry = ValidInt();
879     List<Student> studentList = b1.GetStudentsByStandardID
880                                     (entry).ToList();
881     if (studentList == null || studentList.Count() == 0)
882     {
883         Console.WriteLine("There are no students with that standard
884                             ID");
885     }
886     else
887     {
888         foreach (Student s in studentList)
889         {
890             DisplayStudent(s);
891         }
892     }
893 }
894 static void ShowAllStandards()
895 {
896     List<Standard> standardList = b1.getAllStandards().ToList();
897     if (standardList == null || standardList.Count() == 0)
898     {
899         Console.WriteLine("There are no standards");
900     }
901     else
902     {
903         foreach (Standard s in standardList)
904         {
905             DisplayStandard(s);
906         }
907     }
908 }
```

```
908
909     static void ClientCreatesStudent()
910     {
911         string nameEntry;
912         int standardIDEntry;
913         List<Student> sList = b1.getAllStudents().ToList();
914         int maxID = sList.Max(x => x.StudentID);
915
916         Console.Write("\nEnter the student name: ");
917         nameEntry = Console.ReadLine();
918         Console.Write("\nEnter the standard ID: ");
919         standardIDEntry = ValidInt();
920         Standard selectedStandard = b1.GetStandardByID(standardIDEntry);
921         Student student = new Student()
922         {
923             StudentID = maxID + 1,
924             StudentName = nameEntry,
925             StandardId = standardIDEntry,
926             Standard = null
927         };
928         b1.AddStudent(student);
929         if (selectedStandard != null)
930         {
931             selectedStandard.Students.Add(student);
932             b1.UpdateStandard(selectedStandard);
933         }
934         DisplayStudent(student);
935     }
936 }
937
938 static void ClientDeletesStudent()
939 {
940     int entry;
941     Console.Write("\nEnter the student ID: ");
942     entry = ValidInt();
943     Student selectedStudent = b1.GetStudentByID(entry);
944     if (selectedStudent != null)
945     {
946         Standard studentStandard = selectedStudent.Standard;
947         studentStandard.Students.Remove(selectedStudent);
948         b1.UpdateStandard(studentStandard);
949         List<Course> cList = b1.getAllCourses().ToList();
950         foreach (Course c in cList)
951         {
952             if (c.Students.Contains(selectedStudent))
953             {
954                 c.Students.Remove(selectedStudent);
955             }
956             b1.UpdateCourse(c);
957         }
958     }
959 }
```



```
1005         b1.UpdateStudent(selectedStudent);
1006     }
1007     selectedStudent.StandardId = entry;
1008     selectedStandard.Students.Add(selectedStudent);
1009     b1.UpdateStandard(selectedStandard);
1010     b1.UpdateStudent(selectedStudent);
1011 }
1012 else
1013 {
1014     if (currentStandard != null)
1015     {
1016         currentStandard.Students.Remove
1017         (selectedStudent);
1018         b1.UpdateStandard(currentStandard);
1019         b1.UpdateStudent(selectedStudent);
1020     }
1021     selectedStudent.StandardId = entry;
1022     b1.UpdateStudent(selectedStudent);
1023 }
1024 break;
1025 case 3:
1026     currentStandard = selectedStudent.Standard;
1027     currentStandard.Students.Remove(selectedStudent);
1028     b1.UpdateStandard(currentStandard);
1029     selectedStudent.StandardId = null;
1030     b1.UpdateStudent(selectedStudent);
1031     break;
1032 case 4:
1033     break;
1034 default:
1035     Console.WriteLine("ERROR: Invalid input. Selection
1036     must be between 1 and 4");
1037     break;
1038 }
1039 DisplayStudent(selectedStudent);
1040 }
1041 else
1042 {
1043     Console.WriteLine("A student with that ID does not exist.");
1044 }
1045 }
1046
1047 static void UpdateStudentBySearchingName()
1048 {
1049     string nameSearchEntry;
1050     int entry;
1051     Console.Write("\nEnter the name of the student: ");
```

```
1052     nameSearchEntry = Console.ReadLine();
1053     string options = "1. Change student name" +
1054         "\n2. Change standard" +
1055         "\n3. Remove standard" +
1056         "\n4. Exit update student";
1057     Student selectedStudent = b1.GetStudentByName(nameSearchEntry);
1058
1059     if (selectedStudent != null)
1060     {
1061         DisplayStudent(selectedStudent);
1062         Console.WriteLine(options);
1063         Console.Write("Select an option: ");
1064         entry = ValidInt();
1065         string nameEntry;
1066         Standard selectedStandard;
1067         Standard currentStandard;
1068         switch (entry)
1069         {
1070             case 1:
1071                 Console.Write("\nEnter the student's new name: ");
1072                 nameEntry = Console.ReadLine();
1073                 selectedStudent.StudentName = nameEntry;
1074                 b1.UpdateStudent(selectedStudent);
1075                 break;
1076             case 2:
1077                 Console.Write("\nEnter the id of the new standard: ");
1078                 entry = ValidInt();
1079                 currentStandard = selectedStudent.Standard;
1080                 selectedStandard = b1.GetStandardByID(entry);
1081                 if (selectedStandard != null)
1082                 {
1083                     if (currentStandard != null)
1084                     {
1085                         currentStandard.Students.Remove
1086                             (selectedStudent);
1087                         b1.UpdateStandard(currentStandard);
1088                         b1.UpdateStudent(selectedStudent);
1089                     }
1090                     selectedStudent.StandardId = entry;
1091                     selectedStandard.Students.Add(selectedStudent);
1092                     b1.UpdateStandard(selectedStandard);
1093                     b1.UpdateStudent(selectedStudent);
1094                 }
1095             else
1096             {
1097                 if (currentStandard != null)
1098                 {
1099                     currentStandard.Students.Remove
1100                         (selectedStudent);
```



```
1099         b1.UpdateStandard(currentStandard);
1100         b1.UpdateStudent(selectedStudent);
1101     }
1102     selectedStudent.StandardId = entry;
1103     b1.UpdateStudent(selectedStudent);
1104 }
1105     break;
1106 case 3:
1107
1108     currentStandard = selectedStudent.Standard;
1109     currentStandard.Students.Remove(selectedStudent);
1110     b1.UpdateStandard(currentStandard);
1111     selectedStudent.StandardId = null;
1112     b1.UpdateStudent(selectedStudent);
1113     break;
1114 case 4:
1115     break;
1116 default:
1117     Console.WriteLine("ERROR: Invalid input. Selection
1118                        must be between 1 and 4");
1119     break;
1120 }
1121 DisplayStudent(selectedStudent);
1122 }
1123 else
1124 {
1125     Console.WriteLine("A student with that ID does not exist.");
1126 }
1127
1128 static void DisplayAllStudents()
1129 {
1130     List<Student> studentList = b1.getAllStudents().ToList();
1131     if (studentList == null || studentList.Count() == 0)
1132     {
1133         Console.WriteLine("There are no students");
1134     }
1135     else
1136     {
1137         foreach (Student s in studentList)
1138         {
1139             DisplayStudent(s);
1140         }
1141     }
1142 }
1143
1144 public static void DisplayCourse(Course course)
1145 {
1146     string blankSpace = " ";
```

```
1147         if (course.Location == null && course.TeacherId == null)
1148         {
1149             Console.WriteLine("Course ID: {0} \t| Course Name: {1} \t|      ↗
1150                 Location: {2} \t| Teacher ID: {3} \t| Teacher: {4}",
1151                 course.CourseId,
1152                 course.CourseName,
1153                 blankSpace,
1154                 blankSpace,
1155                 blankSpace
1156             );
1157         }
1158         else if (course.Teacher == null)
1159         {
1160             Console.WriteLine("Course ID: {0} \t| Course Name: {1} \t|      ↗
1161                 Location: {2} \t| Teacher ID: {3} \t| Teacher: {4}",
1162                 course.CourseId,
1163                 course.CourseName,
1164                 course.Location,
1165                 course.TeacherId,
1166                 blankSpace
1167             );
1168         }
1169         else if (course.Location == null)
1170         {
1171             Console.WriteLine("Course ID: {0} \t| Course Name: {1} \t|      ↗
1172                 Location: {2} \t| Teacher ID: {3} \t| Teacher: {4}",
1173                 course.CourseId,
1174                 course.CourseName,
1175                 blankSpace,
1176                 course.TeacherId,
1177                 course.Teacher.TeacherName
1178             );
1179         }
1180         else
1181         {
1182             Console.WriteLine("Course ID: {0} \t| Course Name: {1} \t|      ↗
1183                 Location: {2} \t| Teacher ID: {3} \t| Teacher: {4}",
1184                 course.CourseId,
1185                 course.CourseName,
1186                 course.Location,
1187                 course.TeacherId,
1188                 course.Teacher.TeacherName
1189             );
1190         }
1191     }
1192 }
1193
1194 public static void DisplayTeacher(Teacher teacher)
1195 {
1196     if (teacher.Standard != null)
```

```
1192     {
1193         Console.WriteLine("Teacher ID: {0} \t| Name: {1} \t| Standard ID: {2} \t| Standard: {3}",
1194             teacher.TeacherId,
1195             teacher.TeacherName,
1196             teacher.StandardId,
1197             teacher.Standard.StandardName
1198         );
1199     }
1200     else if (teacher.StandardId == null)
1201     {
1202         Console.WriteLine("Teacher ID: {0} \t| Name: {1} \t| Standard ID: {2} \t| Standard: {3}",
1203             teacher.TeacherId,
1204             teacher.TeacherName,
1205             " ",
1206             " ");
1207     };
1208 }
1209 else
1210 {
1211     Console.WriteLine("Teacher ID: {0} \t| Name: {1} \t| Standard ID: {2} \t| Standard: {3}",
1212         teacher.TeacherId,
1213         teacher.TeacherName,
1214         teacher.StandardId,
1215         " ");
1216 };
1217 }
1218 }
1219
1220 public static void DisplayStandard(Standard standard)
1221 {
1222     if (standard.Description != null)
1223     {
1224         Console.WriteLine("Standard ID: {0} \t| Name: {1} \t| Description: {2}",
1225             standard.StandardId,
1226             standard.StandardName,
1227             standard.Description
1228         );
1229     }
1230     else
1231     {
1232         Console.WriteLine("Standard ID: {0} \t| Name: {1} \t| Description: {2}",
1233             standard.StandardId,
1234             standard.StandardName,
1235             " ")
1236     }
1237 }
```

```
1236         );
1237     }
1238 }
1239
1240 public static void DisplayStudent(Student student)
1241 {
1242     if (student.StandardId == null)
1243     {
1244         Console.WriteLine("Student ID: {0} \t| Name: {1} \t| Standard ID: {2}",
1245             student.StudentID,
1246             student.StudentName,
1247             " ");
1248     };
1249 }
1250 else
1251 {
1252     Console.WriteLine("Student ID: {0} \t| Name: {1} \t| Standard ID: {2}",
1253         student.StudentID,
1254         student.StudentName,
1255         student.StandardId);
1256 };
1257 }
1258 }
1259
1260 public static int ValidInt()
1261 {
1262     int input;
1263     while (int.TryParse(Console.ReadLine(), out input) == false)
1264     {
1265         Console.WriteLine("Invalid input. Must be an integer.");
1266     }
1267     return input;
1268 }
1269 }
1270 }
1271
```