Programming 3 Assignment 3

Assignment Overview

- The specifications associated with this assignment will incorporate the Singleton and State design patterns into the overall project design.
- Work associated with this assignment will reinforce the student's learning in the following areas:
    - Understanding and implementing the Singleton Pattern
    - Understanding and implementing the State Pattern

Evaluation

- The evaluation of this assignment will be in the form of a Code Review and Run Time Demonstration.
- During the Code Review evaluation will be based on:
    - Comments
    - Documentation
    - Adherence to Standards – and Pattern Best Practices
    - Coding choices (i.e. refactoring, no repeated code, readability etc…)
    - Implementation of Requirements
    - Verbal discussion of code
- During the Run-Time Demonstration evaluation will be based on:
    - A variety of run-time tests

Specifications

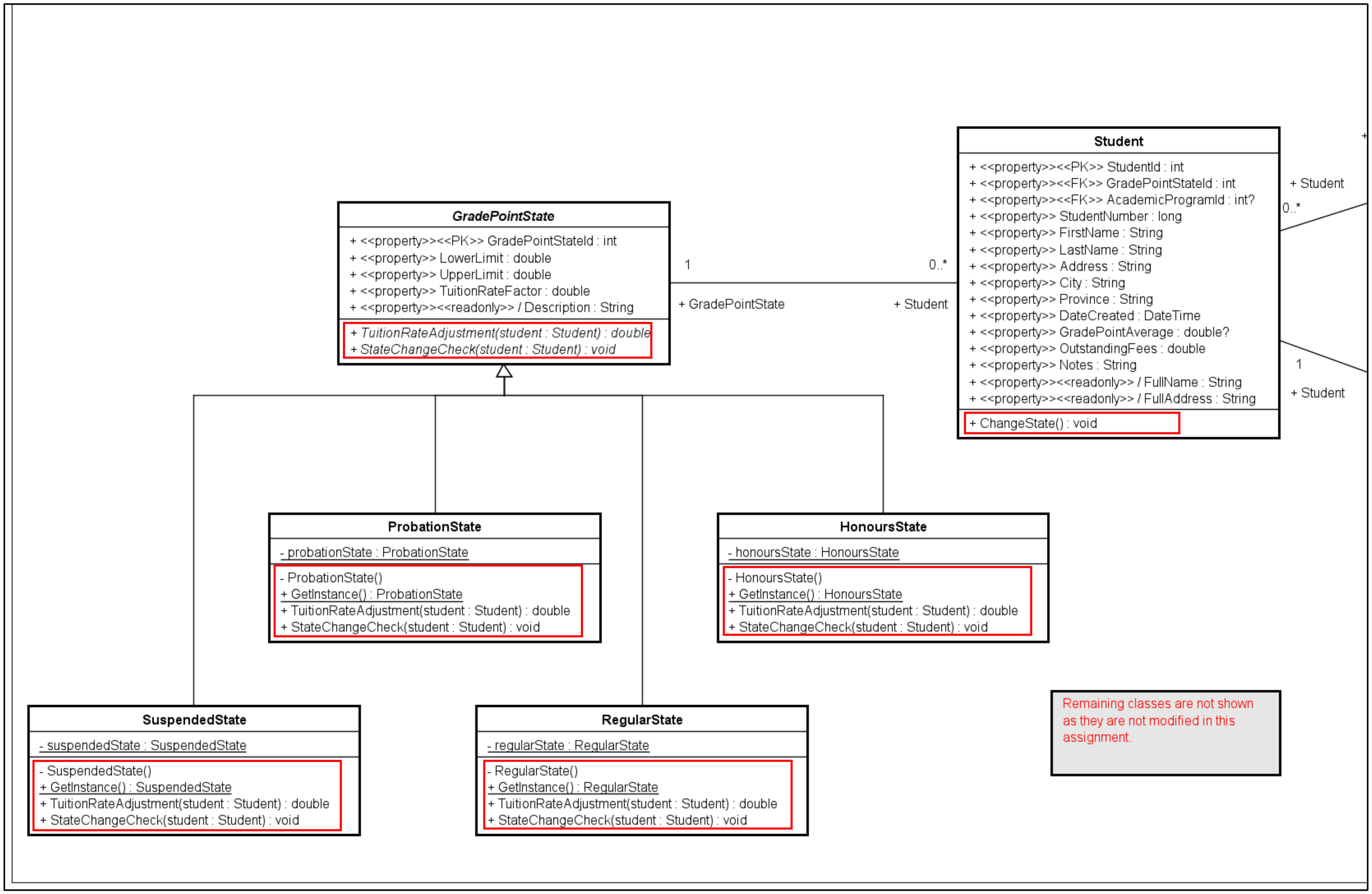
MVC Project

- Continue with the ASP.NET MVC project called **BITCollege_FL** based on the following specifications:

Class Diagram

- The class diagram used in Assignment 1 has been modified. Methods used to support the Singleton and State Design patterns have been added. Only those classes containing new methods are shown and are enclosed in red boxes.
- Declare the methods shown below in Model classes from Assignment 1. Implementation instructions will follow.
    - In cases in which a return value is required, for now, return the default value of the return value's data type. This will be modified in the implementation that follows.

Programming 3 Assignment 3



GradePointState
+ <<property>><<PK>> GradePointStateId : int
+ <<property>> LowerLimit : double
+ <<property>> UpperLimit : double
+ <<property>> TuitionRateFactor : double
+ <<property>><<readonly>> / Description : String
+ TuitionRateAdjustment(student : Student) : double
+ StateChangeCheck(student : Student) : void
1
0..*
+ GradePointState
+ Student
Student
+ <<property>><<PK>> StudentId : int
+ <<property>><<FK>> GradePointStateId : int
+ <<property>><<FK>> AcademicProgramId : int?
+ <<property>> StudentNumber : long
+ <<property>> FirstName : String
+ <<property>> LastName : String
+ <<property>> Address : String
+ <<property>> City : String
+ <<property>> Province : String
+ <<property>> DateCreated : DateTime
+ <<property>> GradePointAverage : double?
+ <<property>> OutstandingFees : double
+ <<property>> Notes : String
+ <<property>><<readonly>> / FullName : String
+ <<property>><<readonly>> / FullAddress : String
+ ChangeState() : void
+ Student
0..*
1
+ Student
+
ProbationState
- probationState : ProbationState
- ProbationState()
+ GetInstance() : ProbationState
+ TuitionRateAdjustment(student : Student) : double
+ StateChangeCheck(student : Student) : void
HonoursState
- honoursState : HonoursState
- HonoursState()
+ GetInstance() : HonoursState
+ TuitionRateAdjustment(student : Student) : double
+ StateChangeCheck(student : Student) : void
SuspendedState
- suspendedState : SuspendedState
- SuspendedState()
+ GetInstance() : SuspendedState
+ TuitionRateAdjustment(student : Student) : double
+ StateChangeCheck(student : Student) : void
RegularState
- regularState : RegularState
- RegularState()
+ GetInstance() : RegularState
+ TuitionRateAdjustment(student : Student) : double
+ StateChangeCheck(student : Student) : void
Remaining classes are not shown as they are not modified in this assignment.

Figure 1 (electronic version in the Assignment 3 Files folder)

The Singleton Pattern

- The GradePointState sub classes will implement the Singleton Pattern. That is, for each GradePointState sub class, the pattern restricts the instantiation of the class to a single object.
- Feel free to incorporate private constants into each of the GradePointState sub classes to allow the code that follows to be more readable and maintainable.

Constructors

- For the SuspendedState private constructor, set the inherited auto-implemented properties as follows:

Lower Limit	**Upper Limit**	**Tuition Rate Factor**
0.00	1.00	1.1

- For the ProbationState private constructor, set the inherited auto-implemented properties as follows:

Lower Limit	**Upper Limit**	**Tuition Rate Factor**
1.00	2.00	1.075

- For the RegularState private constructor, set the inherited auto-implemented properties as follows:

Lower Limit	**Upper Limit**	**Tuition Rate Factor**
2.00	3.70	1.0

- For the HonoursState private constructor, set the inherited auto-implemented properties as follows:

Lower Limit	**Upper Limit**	**Tuition Rate Factor**
3.70	4.50	0.9

Programming 3 Assignment 3

GetInstance methods

- In the GradePointState base class, instantiate a protected static variable of your data context object (BITCollege_FLContext) so that the following can be achieved in each of the derived classes.
    - **<u>Note</u>**: This data context object is defined as:
        - Protected because it is intended to be used in the sub classes.
        - Static as it will be used in the static GetInstance method implementations below.
- The GetInstance methods of each of the GradePointState sub classes will return the one and only object of the GradePointState sub class
- For **EACH** of the GradePointState sub class code the GetInstance method as follows:
    - If the private static variable of the GradePointState sub class is null
        - Use the inherited data context object to determine whether a record of this GradePointState sub type exists in the database
            - If so populate the static variable with that record
                - **<u>Hint</u>**: look at the Index ActionResult method in the corresponding controller for appropriate syntax to accomplish this task - but use the .SingleOrDefault() method instead of the ToList() method as we are expecting one or no records.
            - If not, use the private constructor to populate the static variable with an object of the GradePointState sub class
                - Use the populated static variable to persist this record to the database
                - **<u>Hint</u>**: look at the [HTTPPost] Create ActionResult method in the corresponding controller for appropriate syntax to accomplish this task.
    - Return the populated object.

Programming 3 Assignment 3

Controller and Views

- To apply the implementation of the Singleton Pattern to the project:
    - Delete the Create Views for **EACH** of the GradePointState sub classes
    - Delete the Index Views for **EACH** of the GradePointState sub classes

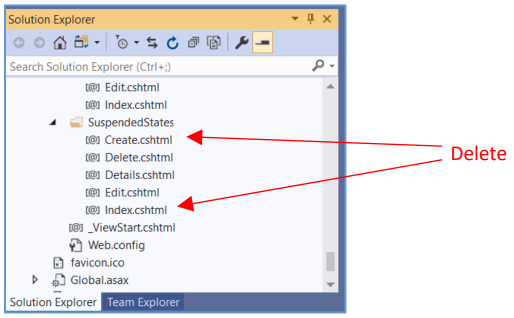


Figure 2

- Create a copy of the Details View in **EACH** of the GradePointState sub class view folders
    - Rename **EACH** of these copies to Index.cshtml
    - Remove paragraph block containing the HTML.ActionLinks from **EACH** of the new Index.cshtml as they will not be needed. The block will look like the following code:

```
<p>
  @Html.ActionLink("Edit", "Edit", new { id = Model.GradePointStateId })
  @Html.ActionLink("Back to List", "Index")
</p>
```

Code 1

- For **EACH** GradePointState sub class controller
    - Replace the code in the Index ActionResult method with the following:

```
return View(GradePointStateSubClass.GetInstance());
e.g.:
return View(SuspendedState.GetInstance());
```

Code 2

- Run the **EACH** of the GradePointState sub class Index views in the browser.
    - Run these in the following order:
        - SuspendedStates
        - ProbationStates
        - RegularStates
        - HonoursStates
    - Currently only the SuspendedState model has a corresponding record in the database.
    - When the other Index Views are run (e.g. Probation, Regular and Honours), the record will be created (based on the GetInstance logic) and then displayed.
    - The next time the Index views are run, the record will exist and will display – a new record should NOT be created.

The State Pattern

- The Grade Point State sub classes will also implement the State Pattern. You will see evidence of this pattern when updating and viewing Student records. A Student record cannot be assigned more than one state at a time, but the Student record's state can change from one to another.
- The change in state of a Student record will be based on the GradePointAverage value of the Student.
- Feel free to incorporate private constants into each of the Grade Point State subtypes to allow the code that follows to be more readable and maintainable.

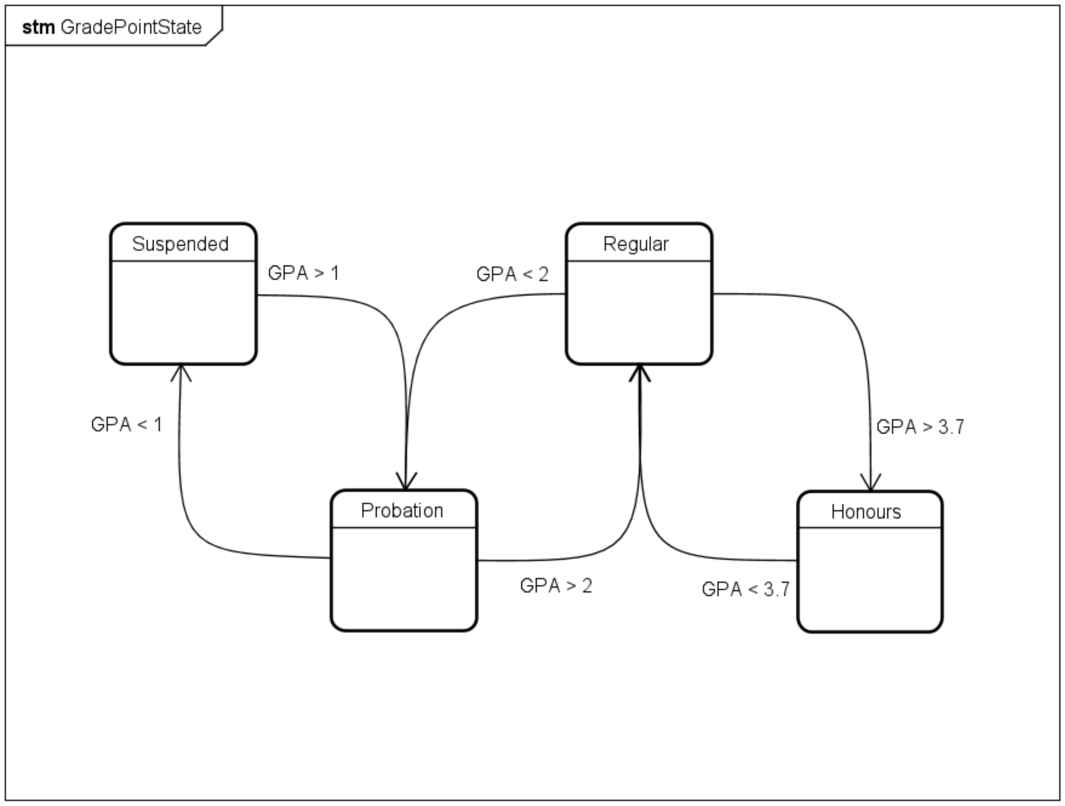


Figure 3

StateChangeCheck

- A change from one state to another (e.g. Suspended to Probation) will be realized in the StateChangeCheck method.
    - As such the StateChangeCheck method must be implemented for **EACH** of the Grade Point State subtypes
- If the GradePointAverage of the Student argument falls above the UpperLimit property of the current GradePointState sub class, the Student's GradePointStateId property must be set to the GradePointStateId of an instance of the next (neighbouring) State and this change must be persisted.
    - For example, if the upper limit of a Suspended Student's GradePointAverage is 1.00, and the actual GradePointAverage of the student is 1.01, that student's GradePointStateId property must be set to the GradePointStateId which corresponds to the Probation state.
- If the GradePointAverage of the Student argument falls below the LowerLimit property, the Student's GradePointStateId property must be set to the GradePointStateId of an instance of the next (neighbouring) State and this change must be persisted.

  - For example, if the lower limit of a Probation Student's GradePointAverage is 1.00, and the actual GradePointAverage of the student is 0.99, that student's GradePointStateId property must be set to the GradePointStateId which corresponds to the Suspended state.
- Persist any changes made to the Student record.

<u>Notes</u>:

- Reminder: The GradePointState sub classes have implemented the Singleton Pattern. As such, multiple instances cannot be created of these types. Use Singleton functionality as necessary to update the Student's GradePointStateId property.
  - Do not hard-code the value of the GradePointStateId.
  - Do not increment/decrement the value of the GradePointStateId.
    - Instead use the GetInstance method of the neighbouring state to obtain the GradePointStateId of the appropriate state.
- A GradePointAverage cannot drop below a Suspended State.
- A GradePointAverage cannot move above an Honours State.

TuitionRateAdjustment

- **Note**: The TuitionRateAdjustment methods will not be used until a later assignment. As such, it is important that you carefully desk-check your code. As well, when the TuitionRateAdjustment is eventually used, you may need to revisit your code in case your desk-checking did not catch all errors.
- As incentive for students at BIT College to maintain a good GradePointAverage, the tuition that a student pays is based, in part, on the Grade Point State of the Student. For Example, Students with a Suspended State will pay more for tuition than students with an Honours State. Each state has a TuitionRateFactor which will apply a discount or a premium to the actual cost of tuition for the individual course registrations. The actual discount or premium value will be further modified depending on specific scenarios for the various Grade Point States.
    - As such the implementation for the TuitionRateAdjustment method will be slightly different depending on the Grade Point State of the student.

SuspendedState

- For Students with a Suspended GradePointState, the TuitionRateFactor for each newly registered course has already been defined as 1.1. As such, all SuspendedState students will pay an additional 10% for each newly registered course.
- If the Student's GradePointAverage has dropped below 0.75, tuition for each newly registered course is increased by 2% above the SuspendedState's default premium.
- If the Student's GradePointAverage has dropped below 0.50, tuition for each newly registered course is increased by 5% above the SuspendedState's default premium.
- Do not modify the TuitionRateFactor, rather use the TuitionRateFactor to calculate the value that is to be returned from this method.

Examples:

- A student with a GradePointAverage 0.44 registering for a course that has a TuitionAmount of $1000 will be charged $1,150.
- A student with a GradePointAverage of 0.60 registering for a course that has a TuitionAmount of $1000 will be charged $1,120.
- A student with a GPA 0.80 registering for a course that has a TuitionAmount of $1000 will be charged $1,100.

Programming 3 Assignment 3

ProbationState

- For Students with a Probation GradePointState, the TuitionRateFactor for each newly registered course has already been defined as 1.075. As such, all ProbationState students will pay an additional 7.5% for each newly registered course.
- If the Student has completed 5 or more courses, tuition for each newly registered course is increased by only 3.5%.
    - <u>**Note**</u>: a completed course is defined as a Registration in which the Grade property is not NULL.
- Do not modify the TuitionRateFactor, rather use the TuitionRateFactor to calculate the value that is to be returned from this method.

<u>**Examples**</u>:

- A student with a GradePointAverage of 1.15 and has completed 3 courses, registering for a course that has a TuitionAmount of $1000 will be charged $1,075.
- A student with a GradePointAverage 1.15 who has completed 7 courses, registering for a course that has a TuitionAmount of $1000 will be charged $1,035.

RegularState

- For Students with a Regular GradePointState, the TuitionRateFactor for each newly registered course has already been defined as 1. As such, there will be no adjustments made to the cost of tuition for all RegularState students.

<u>**Example**</u>:

- A student with a GradePointAverage of 2.50, registering for a course that has a TuitionAmount of $1000 will be charged $1,000.

HonoursState

- For Students with an Honours GradePointState, the TuitionRateFactor for each newly registered course has already been defined as 0.90. As such, all Honours students will pay 10% less for each newly registered course.
- If the Student has achieved an Honours GradePointState after having completed 5 or more courses, tuition for each newly registered course is discounted by an <u>additional</u> 5%.

    - **Note**: a completed course is defined as a Registration in which the Grade property is not NULL.
- If the Student's GradePointAverage is above 4.25, the student will receive an additional 2% discount.
- **Note**: The above scenarios are mutually exclusive. As such, a student can be eligible for both discounts.
- Do not modify the TuitionRateFactor, rather use the TuitionRateFactor to calculate the value that is to be returned from this method.

Examples:

- A student with a GradePointAverage 3.90 and has completed 3 courses, registering for a course that has a TuitionAmount of $1000 will be charged $900.
- A student with a GradePointAverage of 4.27 and has completed 4 courses, registering for a course that has a TuitionAmount of $1000 will be charged $880.
- A student with a GradePointAverage of 4.40 and has completed 7 courses, registering for a course that has a TuitionAmount of $1000 will be charged $830.
- A student with a GradePointAverage of 4.10 and has completed 7 courses, registering for a course that has a TuitionAmount of $1000 will be charged $850.

Student model

ChangeState method

Discussion

- The ChangeState method will initiate the process of ensuring that the Student is always associated with the correct state.
- This method will *eventually* be called whenever a Student record is updated.
- The StateChangeCheck methods implemented above only allow for movement to the next or previous state. However, a large change in GradePointAverage may necessitate a state changing beyond the next or previous state.
- Remember, when implementing this method, that you must remain true to the benefits of the State pattern
    - Reduced conditional statements
    - Increased cohesion
    - Potential extensibility

Implementation

- Instantiate a private instance of the Data Context class (BITCollege_FLContext) for use in this method.
    - Define this at the class level as it will be needed in additional methods in later assignments.
    - **Note**: Since this Data Context object is not being used in a static method, it does not need to be defined as static (as it had been in the GradePointState model)
    - **Note**: Since this Data Context object is not being used in sub classes, it does not need to be protected (as it had been in the GradePointState model)
- Use the Data Context instance to obtain a GradePointState object based on that Student's GradePointStateId
- Use the GradePointState object to call the StateChangeCheck method passing the current Student record
- The code above will need to execute repeatedly as state may change beyond the next/previous state. Repeat the above process as long as states keep changing.
- **Note**: You cannot use recursion in this method. That is, you cannot call ChangeState from within ChangeState.

StudentsController

- Modify the StudentsController as follows:
    - **Prior** to the db.SaveChanges() method call in **BOTH** the Edit and Create [HttpPost] ActionResult methods, call the ChangeState method using the argument supplied to the ActionResult method.

Students Edit View

- The following is done because we have automated the process of assigning the GradePointStateId. As such this field should no longer be editable.
- **Note**: A similar process *could* be applied to the Create View, however it adds an unnecessary level of complication to the project. As such, we will only be making the modifications below to the Edit view.
- Modify the Student Edit view by:
    - Adding the following HiddenFor statement (red) below the Student Id HiddenFor statement:

```
 @Html.HiddenFor(model => model.StudentId)
@Html.HiddenFor(model => model.GradePointStateId)
```

Code 3

- Removing the *div class="form-group"* (red) for the GradePointStateId field:

```
    <div class="form-group">
        @Html.LabelFor(model => model.GradePointStateId, "GradePointStateId", htmlAttributes: new { @class =
"control-label col-md-2" })
        <div class="col-md-10">
            @Html.DropDownList("GradePointStateId", null, htmlAttributes: new { @class = "form-control" })
            @Html.ValidationMessageFor(model => model.GradePointStateId, "", new { @class = "text-danger" })
        </div>
    </div>
```

Code 4

Test

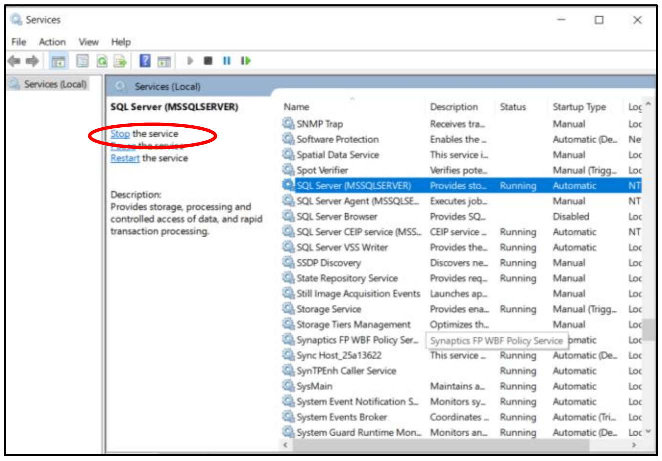
- To test the implementation of the State Pattern
    - Run the application and open the Student Index view.
    - Click the Edit link beside a Student record.
    - Modify the GradePointAverage value for the Student and update the record.
    - You should be redirected back to the Index page and the updated GradePointAverage AND State should display.

- To unit test the implementation of TuitionRateAdjustment (**optional, but recommended)**
    - Create a new console application (.NET Framework) in your BITCollege_FL solution
    - Set the console application as the startup project
    - Add project references to the BITCollege MVC and Utility projects
    - Add appropriate using statements to Program.cs
    - Install Entity Framework by navigating to Tools -> Nuget Package Manager -> Manage Nuget Packages -> Entity Framework 6.4.4
    - Copy the connection string from the web.config file (including the connection string tags) in BITCollege_FL and paste in the Console application's App.config file, below the closing </configSections> tag
    - Create a private static instance of the Context class outside of the main method of the Console application (Program.cs)
    - There are 10 possible unit tests that can be performed

Programming 3 Assignment 3

Backup

- Backup your Solution.
- Create a backup of your database and its corresponding log file at this point
- The database and log file can be found in the following directory:
- C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\DATA
    - **Note**: It may be necessary to stop the SQL Server Service to perform this backup.
    - **Note**: If you stop the service to perform the backup, make sure you restart the service.



SQL Server (MSSQLSERVER)

Stop the service
Restart the service

Description:
Provides storage, processing and controlled access of data, and rapid transaction processing.

Name	Description	Status	Startup Type	Log
SNMP Trap	Receives tra...		Manual	Loc
Software Protection	Enables the ...		Automatic (De...	Ne
Spatial Data Service	This service i...		Manual	Loc
Spot Verifier	Verifies pote...		Manual (Trigg...	Loc
SQL Server (MSSQLSERVER)	Provides sto...	Running	Automatic	NT
SQL Server Agent (MSSQLSE...	Executes job...		Manual	NT
SQL Server Browser	Provides SQ...		Disabled	Loc
SQL Server CEIP service (MSS...	CEIP service ...	Running	Automatic	NT
SQL Server VSS Writer	Provides the...	Running	Automatic	Loc
SSDP Discovery	Discovers ne...	Running	Manual	Loc
State Repository Service	Provides req...	Running	Manual	Loc
Still Image Acquisition Events	Launches ap...		Manual	Loc
Storage Service	Provides ena...	Running	Manual (Trigg...	Loc
Storage Tiers Management	Optimizes th...		Manual	Loc
Synaptics FP WBF Policy Ser...	Synaptics FP WBF Policy Service		omatic	Loc
Sync Host_25a13622	This service ...	Running	Automatic (De...	Loc
SynTPEnh Caller Service		Running	Automatic	Loc
SysMain	Maintains a...	Running	Automatic	Loc
System Event Notification S...	Monitors sy...	Running	Automatic	Loc
System Events Broker	Coordinates ...	Running	Automatic (Tri...	Loc
System Guard Runtime Mon...	Monitors an...	Running	Automatic (De...	Loc

Figure 4

Evaluation: In Person/Hybrid Delivery

- This assignment must be evaluated on or before the due date/time specified in Learn.
    - The assignment is due at the beginning of class on the due date specified. Your instructor will evaluate the assignment on the students' machine in a random order. Remote evaluations will be based on a first come, first served basis.