|  |  |
| --- | --- |
| Student Name: | Roman Kovtun |
| Course: | CIS339 |
| Session (month, year): | November 2018 |

Object-Oriented System Analysis and Design

The School of Prosperity  
Student Records System (SRS)

[Week 1—System Request 3](#_Toc360108365)

[Week 2—Use Case Diagram AND Use Cases Descriptions 5](#_Toc360108366)

[Week 3—Class Diagram AND CRC Cards 9](#_Toc360108367)

[Week 4—Sequence, Communication, and State Machine Diagrams 15](#_Toc360108368)

[Week 5—Package Diagram 19](#_Toc360108369)

[Week 6—Method Contract AND Method Specification 21](#_Toc360108370)

[Week 7—Object-Oriented Application Coding 24](#_Toc360108371)

# Week 1—System Request

|  |  |
| --- | --- |
| 1 | Use this system request template and complete the SRS system request.  **System Request—** Schedule Online System (SOS)  **Project sponsor:**  The School of Prosperity (SOP) located in the national capital of Washington, D.C.  **Business Need:**  Due to high demand the school need a schedule system for the students to eliminate a human error for the registration form from paper to computer. Each semester teacher requires to add/drop a student from main system in order to register for the classes. It takes a lot of time and it’s not officiant in our days to use a paper to register new students.  **Business Requirements:**  System requires a student to register for the class without fill out the paper. In addition, students can review their registered classes before the semester started. With a new system teacher can process the students add/drop per semester and will give them opportunity to save time when they fill out information about the student right away inside of the computer. Finally, teacher could access the schedule anywhere from outside the campus and do adjustments if needed.  **Business Value:**  The value of the business is to eliminate the human errors when taking information from paper to computer. Teacher will have opportunity to adjust their schedule if needed and save their time each semester.  **Special Issues or Constraints:**  The system doesn’t require extra cost to build, most of the functionality is to add/drop the students for the next semester and store students information of the server. If the student graduate or leave the facility, he will be removed from the class list and moved archive system in order to save the place for other students. Access anywhere from Internet. |
| 2 | Validate and verify that your system request addresses the SRS Preliminary Planning Overview.  The business need area explains what the SoP is requiring the system to operate. The business requirements consist of what the system must be able to handle and the expectations of the system by end of completion. The business value breaks down the details of information that the new system will save on both staff hours and finances for the school. |
| 3 | Explain how you completed your work, the decisions you made to arrive at your conclusions, and the lessons you learned.  My work was completed by reading the preliminary planning overview to get a good understanding of the requirements for the system, what the current system does, and what the system is supposed to implement. I arrived at my conclusion by evaluating the different possibilities that were needed and required. This system is going to take a lot of work from the start to complete. I also learned that mistakes are bound to happen because there are a lot of requirements for the system and sometimes things get overlooked due to human error. Having any type of error will be detriment to the entire system, which might cause the system to not work properly. |

# Week 2—Use Case Diagram and Use Cases Descriptions

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Generate a use case diagram for all of your use cases, including their actors. The diagram must be generated by a UML drawing tool. Copy and paste your diagram here:  **SRS Use Case Diagram**            Use this use case description template and complete the two SRS use cases of (1) Maintain Class Records and (2) Register a Student for Classes.  **Maintain Class Records**   |  |  |  |  | | --- | --- | --- | --- | | Use Case Name: Maintain Class Records System | | ID:123-123 | Importance Level: Normal | | Primary Actor: Staff | Use Case Type: | | | | Stakeholders and Interests: School Staff – maintain (create, edit, modify, and delete) class records for a given course in Student Online System <<system>> Class – store all classes in the database. Actor (School Staff) can only update or delete class per one selection. Relation ship Many classes to one selected class. | | | | | Brief Description: This use case maintains the school records of the classes it offers. By class in course. It has an option to Create, Read, Update, and Delete. It provides the School Staff actor with complete control over the creation, deletion, modification and the reading of class records for a given course. | | | | | Trigger: A School Staff wanting to maintain the class records for a given course  Type: | | | | | Relationships:  Association: with the School Staff actor  Include: relationship to the Login into SRS use case  Extend: Internet connection and user verification, Search Course ID  Generalization: - | | | | | Normal Flow of Events:   * 1. User getting verification token   2. The school staff enters the course whose classes are to be maintained   3. The system responds with a list of all classes for the entered course   4. The School Staff then select one of the classes and commands the system to perform one of the following operations on the selected class:      1. Delete the selected class      2. Modify certain fields in the selected class   5. Alternative way, the School Staff actor can create a new class offering for the course, where he/she cam select between online or face-to-face class, and then enter the required infmation for all kinds of classes and for each classification as fallow      1. For all classes: Class ID, Class Begin Date and End Date      2. For Online classes: Class URL, Class Browser      3. For Face-to-Face classes: Class Building, Class Room | | | | | SubFlows: - | | | | | Alternate/Exceptional Flows:  Alternative flow: each one of the CRUD operations is considered an alternative flow through the use case.  Exceptional flow: when the course entered, it might not exist in the system. | | | |   **Register a Student for Classes**   |  |  |  |  | | --- | --- | --- | --- | | Use Case Name: Register Student for Classes System | | ID: 123 | Importance Level: Normal | | Primary Actor: School Staff, Student | Use Case Type: Detail, Essential | | | | Stakeholders and Interests:  Student – self-register for classes School Staff – register on behalf of students <<system>> Classes – store all classes in database <<system>> Courses – store all courses that related to the classes in database <<system>> Registration – store all registration that was made by School Staff or Student in the database | | | | | Brief Description: This use case enables the actor to select a class (online or face-to-face) and then registers a student for the selected class provided that the student meets the registration requirements for the class. | | | | | Trigger: an actor (Student, School Staff) who wants to register a student for a class.  Type: | | | | | Relationships:  Association: with an actor (Student, School staff)  Include: relationship to the Login into SRS use case, Validation of registration, list of available courses, confirmation registration, violation registration, select class of selected course.  Extend: Verification of the user, Internet connection,  Generalization: <system>> Course is set of <<system>> Classes, All students are related to the <<system>> Registration database. | | | | | Normal Flow of Events:   1. User getting verification token 2. The system identifies the student implicitly from the login (if the actor is a Student) or by explicitly entering the student ID (if the actor is a School Staff) and verifying it. 3. The actor enters the course for which classes registration is thought 4. The system responds with a list of classes (pulls the list from <<system>> Class database) offered of the selected course (<<system>> Course list from database) 5. The actor then selects a class from the list of classes 6. The actor then commands the system to register the identified student for the selected class 7. The system verifies the registration rules as applied to the student and class (<<system>> Registration verify all available course from being duplicated). 8. If registration rules are verified, then the student is registered for the class and the registration is confirmed (<<system>> Registration send record to save to database). If not, then an error message is printed to the actor with the reason why registration cannot be competed at this time. 9. The system allows the actor to re-start the use case again beginning with event to register for another class 10. System will automatically update registration after class is completed and all other records like grades will be send to the Student. 11. Student on his/her behalf will maintain the course dropping if it considers using in the future. | | | | | SubFlows: - | | | | | Alternate/Exceptional Flows:  Alternative flow: same as above normal flow except validation of registration rules fails and the system prints out a message for the reason why the registration cannot be completed.  Exceptional flow: then the entered student ID is might not found in the system | | | | |
| 2 | Validate and verify your use case diagram and use case descriptions against the SRS Requirement Definition and the SRS System Request.  The area of this system is to save time for actor (School Staff, Student) Each actor has a right to control the system, however, the limitations of the Student actor is only allowed register and drop classes, when School Staff has a full control over the Courses, Classes, Registration and Student Records. Each access requires a validation token in order to take control over the system. |
| 3 | Explain how you completed your work, the decisions you made to arrive at your conclusions, and the lessons you learned.  This week assignment is simple but use of SRS system took me a while to understand. This week I learn how to create UML Use Case diagram by only using Activity Diagram. Also, I learn how to create a subsystem items in order to control database and user behavior. Include and extract the use case for each behavior and associate the records between use case and actor. |

# Week 3—Class Diagram and CRC Cards

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Generate a class diagram for the SRS system. The diagram must be generated by a UML drawing tool. Copy and paste your diagram here:  **SRS Class Diagram**    Use this CRC template and complete a CRC card for each class you designed in your SRS class diagram.  **Class1 CRC Card**   |  |  |  |  | | --- | --- | --- | --- | | **Front:** | | | | | **Class Name:** School Staff | **ID:** 1 | | **Type:** Concrete, System | | **Description:** The School Staff that control over Students, Classes, Courses, and Registrations. | | | **Associated Use Cases:** School Staff | | **Responsibilities**  Register Student, Manage Student Records, register students for the classes, manage classes, Manage Courses | | **Collaborators**  School Staff | | | **Back:** | | | | | **Attributes:**  School Staff ID, School Staff Name | | | | | **Relationships:**  School Staff has control over Students One to Many. School Staff has control over Courses One to Many, School Staff has control over Class One to Many, School Staff has control over Registration One to many  **Generalization (a-kind-of): None**  **Aggregation (has-parts):** None  **Other Associations:**  Every association with each class is One to Many regards that School Staff has control over any System in the school | | | |   **Class2 CRC Card**   |  |  |  |  | | --- | --- | --- | --- | | **Front:** | | | | | **Class Name:** Student | **ID:** 1 | | **Type:** Concrete, System | | **Description:** Student has a right to register for the classes and check his information in the system | | | **Associated Use Cases:** Student | | **Responsibilities**  Providing Student Information, Getting Student GPA, (Updating Deleting and Adding) Student Information, register for the Class, Updating or Deleting the registration | | **Collaborators**  Student | | | **Back:** | | | | | **Attributes:**  First Name, Middle Initials, Last Name, Student ID, department (e.g. Grad Law, Undergrad IT), and Date of Birth, GPA (scale of 1.0 - 4.0) | | | | | **Relationships:**  Student can register for many classes when it associates with his GPA and requirements. Student has a right to search for the course and search for the class that he/she need to register for.  **Generalization (a-kind-of):** None  **Aggregation (has-parts):** None  **Other Associations:** Student has limited roles and he/she can only manage the registration and his/her own account or view personal information. | | | |   **Class3 CRC Card**   |  |  |  |  | | --- | --- | --- | --- | | **Front:** | | | | | **Class Name:** Course | **ID:** 1 | | **Type:** Concrete, System | | **Description:**  The collection of the classes in the course. Course contain the list of classes that student can take to accomplish his/her degree. | | | **Associated Use Cases:**  Course | | **Responsibilities**  School Staff can Update/Create/Delete Course, the course contains prerequisite courses, student/school staff can search for the course name | | **Collaborators**  Course, Student, School Staff | | | **Back:** | | | | | **Attributes:**  Course ID, Course Name, Credit Hours, Description, Prerequisite Course | | | | | **Relationships:** Each course can contain many classes. Each student/school staff can search for the courses. Each course can have multiple registrations with many classes. School staff can manage many courses  **Generalization (a-kind-of):** none  **Aggregation (has-parts):** none  **Other Associations:** none | | | |   **Class4 CRC Card**   |  |  |  |  | | --- | --- | --- | --- | | **Front:** | | | | | **Class Name:** Class | **ID:** 1 | | **Type:** Concrete, System | | **Description:**  The class that student can register for to proceed with their degree. | | | **Associated Use Cases:**  Class | | **Responsibilities**  Get type of the class and set it to be online class or face-to-face class. If the class is online then it will add class URL and class browser, if the class is face-to-face then it will add class building and class room. Each class can be deleted/added/updated. Each class display information about the class | | **Collaborators**  Student, School Staff, Class | | | **Back:** | | | | | **Attributes:**  Course ID, Class Begin Date, and Class End Date. In addition, online classes also should add or maintain this additional information: Class URL, Class Browser. Similarly, face-to-face classes should add or maintain this additional information about them: Class Building, Class Room. | | | | | **Relationships:**  Each class can maintain multiple registration, each class can be assigning to multiple courses because each course can be different but share same class. Classes can be register many to many. Class can be updated/deleted/added by School Staff. Student/School Staff can view a list of classes  **Generalization (a-kind-of):** none  **Aggregation (has-parts):** none  **Other Associations:** | | | |   **Class 5 CRC Card**   |  |  |  |  | | --- | --- | --- | --- | | **Front:** | | | | | **Class Name:** Registration | **ID:** 1 | | **Type:** Concrete, System | | **Description:**  Registration that associate with classes, course, and students | | | **Associated Use Cases:**  Registration | | **Responsibilities**  Each registration can contain the cours, class, and students list. School Staff can update/delete/add the registration. Student can view his/her registration | | **Collaborators**  Student, School Staff, Registration | | | **Back:** | | | | | **Attributes:**  Registration ID, Course ID, Class ID, Student ID | | | | | **Relationships:**  Each registration can contain multiple classes and course many to many. School staff can update/delete/add registration for the student many to many. Student can view his registrations many to many.  **Generalization (a-kind-of):** none  **Aggregation (has-parts):** none  **Other Associations:** | | | | |
| 2 | Validate and verify your class diagram and CRC cards against the SRS use case diagram and use case descriptions. Due to limitation of the student access, he/she can only register and search for course or class for their needs. On the other hand, School staff has a full control over the data like student, registration, class, and course. Each class has his own Setter and Getter which helps to define and get information for the School Staff/Student. On them behave of access they can delete/update/add the information for each class attributes. Each attribute follows with autogenerated ID which helps to create a unique field for the user. |
| 3 | Explain how you completed your work, the decisions you made to arrive at your conclusions, and the lessons you learned.  The following week for me was actually very interesting, I have learned how to create a simple UML class diagram and used to create my own system for example a SOS system that helps students to registers for needed class. On the other hand, it really helps to understand how the data will be displayed in the database to store or retrieve some of the information. Each class has a relationship which is associate with many to many or one to many relationship. |

# Week 4—Sequence, Communication, and State Machine Diagrams

|  |  |  |
| --- | --- | --- |
| 1 | Generate a sequence diagram for the SRS system Register a Student for Classes use case. The diagram must be generated by a UML drawing tool. Copy and paste your diagram here:  **Sequence Diagram for the Register a Student for Classes Use Case**     |  | | --- | | Generate a communication diagram for the SRS system Register a Student for Classes use case. The diagram must be generated by a UML drawing tool. Copy and paste your diagram here:  **Communication Diagram for the Register a Student for Classes Use Case** |   Generate a state machine diagram for the SRS system Registration Record object/class (the class that maintains the registration of a student in a class). The diagram must be generated by a UML drawing tool. Copy and paste your diagram here:  **State Machine Diagram for the Registration Record Object** |
| 2 | Validate and verify your behavioral diagrams against the SRS Register a Student for Classes use case description and the SRS class diagram.  For the each of the diagram that displayed above we can see that Sequence diagram is built on the school staff actor interaction where he/she login to the system with a right validation, then he/she search for course and class. Which then follow with registration of the student. The community diagram represents similar behavior as sequence diagram, but in the community diagram we must follow the steps to record the school staff interaction via SOS systems. Finally, State Machine diagram represent a one-way state with events that follows each state. Each event represents a proportion of the school staff anchor in the system. In conclusion, the diagrams that was represented above shares same type of Use Case that was build in week 1, and it represent a different type of messages in which school staff has an ability to register a student for the classes. |
| 3 | Explain how you completed your work, the decisions you made to arrive at your conclusions, and the lessons you learned. Due to number of diagrams I had to consider the same use case that I use in week 1 and compile it three different type of diagrams that shares similar behavior. In overall to complete the task I had to understand what each of those diagrams should be look like and how the behave to each situation when we have limited amount of the information. Even though, state machine diagram has different approach of displaying states and events. In conclusion, this week was interesting and well complicated to challenge us. |

# Week 5—Package Diagram

|  |  |
| --- | --- |
| 1 | Generate a package diagram for the SRS system. The diagram must be generated by a UML drawing tool. Copy and paste your diagram here:  **SRS Package Diagram** |
| 2 | Validate and verify your package diagrams against the SRS class diagram and the SRS Register a Student for Classes use case sequence and communication diagrams. The following diagram has 7 packages that include a various of classes that related between each other. Due to specification of the SOS system we going to review each package on it own.School Staff package  * + Has control over the Student creation, updating, and deleting the student from the system   + It also has control over the registration package to register student for the classes   + In order to retrieve a list of courses or classes it must use a request search package which control over the search request that need to be passed through. * Student Package   + Student has control over his account and class which respond to student notifications like registered class or dropped class   + Student can search for the classes and course, but with limitations   + Student can Register for the classes on their behalf * Search Package   + Package respond to give a list of classes or courses for the user * Course Package   + With admin abilities can create, update, and delete the course if needed   + Also, each course that was search will give a list of courses that available. * Class Package   + Like the Course Package but has dependency of registration where school staff or student can look for the class if they need. * Registration Package   + Container Enrolment of all students that was working on the class   + With admin abilities can create, update, and delete the registration if needed for each student   + Student have a right to drop a class. * Validation Package   + Is one-way validation which validate if student is eligible to take classes that he/she requested for.   + In addition, student information should be validated too, before processing a full registration. |
| 3 | Explain how you completed your work, the decisions you made to arrive at your conclusions, and the lessons you learned.  While I was working on this diagram, I have found different way to approach the problem. However, in the package diagram it very similar to the diagram that I was working two weeks ago. Each package that I created has simple relationship which doesn’t confuse at all. Anyway, it was pretty interesting week to learn how to work with package diagrams and create them from a classes that already was existed from day 1. In addition, the main package that I think will depends on the anything else is search package, which is contain all possible information that user needs to find when he or she looking for the course or class. |

# Week 6—Method Contract and Method Specification

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Use this method contract template and complete a method contract for the GetCourseByCourseID() method of the CourseList class.  **CourseList.GetCourseByCourseID() Method Contract**   |  |  |  | | --- | --- | --- | | Method Name: GetCourseByCourseID() | Class Name: CourseList | ID: 1 | | Clients (Consumers): The stuff member calls this method when they want to choose and show more data for one course from the list of courses. This method is called with the coursed as a parameter in the web interface of the Course object as follows. | | | | Associated Use Cases: Maintain Course Records, Maintain Class Records, Register Class | | | | Description of Responsibilities: First, the method will search the course list for the gieven course ID. If the course ID matches then it returns the course, otherwise it will return NULL for the user. In the NULL case user had an option to repeat | | | | Arguments Received: ID of type Integer (string CourseID;) | | | | Type of Value Returned: Course object or null (if no course exists for the id) | | | | Pre-Conditions:   * courseList should exist and it cannot be empty. * The argument id should not be null | | | | Post-Conditions:   * The method returns the course matching CourseID with id, else return NULL (if there is no match | | |   Use this method specification template and complete a method specification for the GetCourseByCourseID() method of the CourseList class.  **CourseList.GetCourseByCourseID() Method Specification**   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Method Name: GetCourseByCourseID() | | | | Class Name: | | ID: 1 | | Contract ID: 1 | | | | Programmer: Roman Kovtun | | Date Due: 12/31/18 | | Programming Language:  Visual Basic Smalltalk C#  **Java** | | | | | | | | Triggers/Events:  When a staff member chooses a course from the course list to get more data of the selected course, then this method is called. | | | | | | | | Arguments Received:  Data Type: | | Notes: | | | | | String CourseID | | The Course ID of the chosen is used to filter the list of the available courses from the list of the courses. | | | | |  | |  | | | | |  | |  | | | | |  | |  | | | | | Messages Sent & Arguments Passed:  ClassName.MethodName: | | | Data Type: | Notes: | | | NONE | | |  |  | | | Argument Returned:  Data Type: | | Notes: | | | | | Course Object | | If course for ID doesn’t exit, then NULL is returned. | | | | | Algorithm Specification:  Loop throught the whole course list having list of courses.  If CourseID matches with give ID  Then Return this course with assigned classes  End If  End Loop | | | | | | Misc.Notes:  The course search cannot be combined. The only search user can perform is one course per search. Each course will always give a list of classes that was assigned by the course. | | | | | |
| 2 | Validate and verify your method contract and method specification against the CRC card for the CourseList class and the CRC card for the Course class. The following the Get Course By the course ID is performing only one-time search in the system to retrieve the course ID and class list of the assign course. In the following flow, the course can be NULL due to user error or poor communication to the SOS services. Due to limitation to the search, search cannot be performed combining with other courses. User always have a change to get another search he/she needs. In addition, NULL respond will notify the user that course that user was looking for is not exist or was typed wrong. |
| 3 | Explain how you completed your work, the decisions you made to arrive at your conclusions, and the lessons you learned. Due to luck of time I had to think about the project from different perspective. Even it took me extra time to understand what it really to build a class and how it should be used in the project. The actual class CourseList and it method GetCourseByCourseID, I have found the ways to perform a single method that will require to loop through the list of CourseID where it should match the ID of the search and CourseID and then return the course for the user. It really easy approach to follow, but it takes a lot of time to consider the course that will retrieve the class list. |

# Week 7—Object-Oriented Application Coding

|  |  |
| --- | --- |
| 1 | Copy your code text of the CourseList.GetCourseByCourseID() method and paste it here:  **Code Text of Your CourseList.GetCourseByCourseID() Method**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SOSCourseList  {  class Sub  {  // Create a object which will have number, name, credit hours, summary, and prerequisites  public Sub(string classNumber, string className, int creditHours, string classSummary, string preSub)  {  this.SubID = classNumber;  this.SubTitle = className;  this.CreditHours = creditHours;  this.Summary = classSummary;  this.PreSub = preSub;  }  // make the object to be public accessible  public string SubID;  public string SubTitle;  public int CreditHours;  public string Summary;  public string PreSub;  }  class ListTester  {  public static void Main(string[] args)  {  GetCourseByIDCourseExist();  GetCourseByIDCourseDoesNotExist();  Console.WriteLine("Unit Test is Passed!");  Console.ReadKey();  }  // Check if the class is exist  public static void GetCourseByIDCourseExist()  {  Sublist list = new Sublist();  string lookUpCourse = "CIS 339";  Sub course = list.GetCourseByCourseID(lookUpCourse);  if (course.SubID != lookUpCourse)  System.Console.WriteLine("ERROR - GetCourseByIDCourseExist(): Returned ID does not matches to " + lookUpCourse);  }  // Check if the class does not exist  public static void GetCourseByIDCourseDoesNotExist()  {  Sublist list = new Sublist();  string lookUpCourse = "SEC 370";  Sub course = list.GetCourseByCourseID(lookUpCourse);  if (course != null)  System.Console.WriteLine("ERROR - GetCourseByIDCourseDoesNotExist(): must return null");  }  }  class Sublist  {  // Create object-array items  public Sub[] subArray =  {  new Sub ("CIS 339", "Object-Oriented Analysis and Design", 4, "Building on the foundation established in...", "CIS 321"),  new Sub ("SEC 370", "Web Security", 4, "This course examines issues involved in protecting web-based...", "SEC 280")  };  // Check each object-array item  public Sub GetCourseByCourseID(string courseNumber)  {  return subArray.Where(item => item.SubID == courseNumber).FirstOrDefault();  }  }  }  Copy a screenshot of running the unit test of the CourseList.GetCourseByCourseID() method and paste it here:  **Screenshot of running the Unit Test of CourseList.GetCourseByCourseID() Method**    Zip all the files you used in this coding project; copy and paste the \*.zip file here so that it can be unzipped and loaded into the IDE and run on another computer:  **A \*.zip File of all of the Coding Project Files** |
| 2 | Validate and verify your code and screenshot of the CourseList.GetCourseByCourseID() method against the method contract and the method specification of the same method.  I have validated my code during the unit test that I performed, I have made an additional tests that check if the course is related to the object-array. Due to luck of time, I only have made two course test units to check if the available course is related to the list of course, and if the course is not null, in which it will give me an console error that would say that this was not the same course as user was looking for. |
| 3 | Explain how you completed your work, the decisions you made to arrive at your conclusions, and the lessons you learned.  I have completed the work in no time, which gave me opportunity to work on my other homework and start preparing for the finals. The problem was in the first place is that I did not understand the role of unit test that I had to write, but when I saw a video that was explaining the aspects of the unit test and how it should be written, I have made a lot of interesting ideas to write, but I only come up for two unit test in which I testing the existence of the actual course in object-array. |