Telerik Academy

C# OOP - Academy System

General description

You are given a task to recreate the Telerik Academy Integrated Learning System. Of course, it's a lot simplified. Instead of creating a web application, you need to create a console app by using your mighty OOP skills. The application already accepts commands and outputs text for each submitted command, you just need to write the OOP behind most of the command. You can create students, trainers, courses, lectures and so on. You can view the list of the commands below. Example inputs and outputs will be provided with each task, so that you can validate the code you are writing. Make sure to follow the all the Object Orientated Programming practices and conventions that we have talked about during the lectures and don't let the length of this description intimidate you, read it carefully and start hacking!

Commands

Here is the full list of commands that you can use inside the TAILS.

- **CreateSeason** <*startingYear*> <*endingYear*> <*initiative*> - Creates a new season that starts and ends at the given years and is a part of one of the 4 initiatives in Telerik Academy.
- **CreateStudent** <*username*> <*track*> - Creates a new student with that unique username and has the given track.
- **CreateTrainer** <*username*> <*technologiesList*> - Creates a new trainer with that unique username and a list of technologies, seperated by a comma (no spaces).
- **CreateCourse** <*seasonId*> <*name*> <*lecturesPerWeek*> <*startingDate*> - Creates a new lecture with the given name, lectures per week count and starting date, into the season with that ID.
- **CreateCourseResult** <*seasonId*> <*courseId*> <*examPoints*> <*coursePoints*> <*username*> - Creates a new course result for the course with that ID that is in a season with that ID with the given points into the given user.
- **CreateLecture** <*seasonId*> <*courseId*> <*name*> <*date*> <*trainerUsername*> - Creates a new lecture with the given name, date and trainer into the course with that ID that is in the season with that ID.
- **CreateLectureResource** <*seasonId*> <*courseId*> <*lectureId*> <*type*> <*name*> <*url*> - Creates a new lecture resource with the given type, name and url into the lecture with that ID that is in the course with that ID that is in the season with that ID.

- **AddTrainerToSeason** <*username*> <*seasonId*> - Adds the trainer with that username into the season with that ID.
- **AddStudentToSeason** <*username*> <*seasonId*> - Adds the student with that username into the season with that ID.
- **AddStudentToCourse** <*username*> <*seasonId*> <*courseId*> <*form*> - Adds the trainer with that username into the course with that ID that is in the season with that ID.

- **ListUsers** - Lists all registered students and trainers.
- **ListUsersInSeason** <*seasonId*> - Lists all registered students and trainers that have been added to that course.
- **ListCoursesInSeason** <*seasonId*> - Lists all coruses that are in that season.

- **Exit** - Terminates the program.

Architecture

Let's talk a bit about how the system works (you are already provided with all of this stuff, there is no need to implement it). There is an Engine located in the Core namespace that has a loop that cycles until the Exit command is submitted. With each cycle, it takes the input, passes it to the command parser that find the command with that name and executes it with those parameters. All commands are located in the Commands namespace. The commands themselves use the AcademyFactory located in the Core.Factories namespace to create the needed objects. After the command executes, it returns a result message to the Engine that prints it to the console and then the cycle beings again. In the Engine, there is a try-catch block that catches every possible exception type and prints the exception's message to the console. Do not bother reading those classes, your focus should be on the Models namespace, where you need to place the classes you create, using the provided interfaces in the Contracts namespace. The result from the execution of the commands is printed after executing the Exit command.

Restrictions

Violation of these restrictions will cause severe points penalties or exam disqualification.

- You are allowed create new **classes, interfaces, enumerations and namespaces** inside the Models namespace.
- You are allowed to modify the **AcademyFactory** in the Core.Factories namespace.
- You are allowed to modify the **ListUsersCommand** in the Commands.Listing namespace.
- You are allowed to modify the **Grade** enum.
- You are allowed to include a using to the **Track** enum in the interface.
- You are allowed to include usings in the **AcademyFactory**.
- Use the provided zero tests for the most accurate data
- **You are NOT allowed to modify any other existing interfaces!**
- **You are NOT allowed to modify any other existing classes, enumerations and namespaces!**

Section I - Tasks

All validation lenghts are exclusive (<, >)

Course class

Create a **Course** class that implements the ICourse interface. Each **Course** must have a valid **Name** that is a non-empty string, between 3 and 45 symbols long. If the **Name** is not valid, the error message "*The name of the course must be between 3 and 45 symbols!*" should be thrown. In addition, the **Course** must have **LecturesPerWeek** that is an integer between 1 and 7. If that is not valid, the error message "*The number of*

lectures per week must be between 1 and 7!" must be thrown. The **Course**'s end date is calculated by adding 30 days to the starting date. The **Course** must print in the following format, if it does not contain lectures:

```
* Course
 - Name: <name>
 - Lectures per week: <lecturesPerWeek>
 - Starting date: <startingDate>
 - Ending date: <endingDate>
 - Lectures:
  * There are no lectures in this course!
```

Trainer Class

Create a **Trainer** class that implements the ITrainer interface. Each Trainer must have a valid Username that is a non-empty string, between 3 and 16 symbols long. If the Username is not valid, the error message "*User's username should be between 3 and 16 symbols long!*" must be thrown. Each **Trainer** has a collection of **Technologies** that he knows. The **Trainer** must be convertable to a string in the following format:

```
* Trainer:
 - Username: <username>
 - Technologies: <technology>; <technology>; <technology>
```

Student class

Create a **Student** class that implements the IStudent interface. Each Student must have a valid Username that is a non-empty string between 3 and 16 symbols long. If the Username is not valid, the error message "*User's username should be between 3 and 16 symbols long!*" must be thrown. Each **Student** must also have a **Track** that is either Frontend, Dev or None and a collection of course results. If the **Track** is invalid, the error message "*The provided track is not valid!*" should be thorwn The **Student** must be convertable to a string in the following format if he has no course results:

```
* Student:
 - Username: <username>
 - Track: <track>
 - Course results:
  * User has no course results!
```

Section I - Zero test

Input

```
CreateSeason 2016 2017 SoftwareAcademy
```

```
CreateStudent Pesho Frontend
CreateTrainer Gosho Java,Spring
AddStudentToSeason Pesho 0
AddTrainerToSeason Gosho 0
CreateCourse 0 JavaScriptOOP 2 2017-01-24
AddStudentToCourse Pesho 0 0 onsite
ListUsersInSeason 0
ListCoursesInSeason 0
Exit
```

Output

```
Season with ID 0 was created.
Student with ID 0 was created.
Trainer with ID 0 was created.
Student Pesho was added to Season 0.
Trainer Gosho was assigned to Season 0.
Course with ID 0 was created in Season 0.
Student Pesho was added to Course 0.JavaScriptOOP.
* Trainer:
 - Username: Gosho
 - Technologies: Java; Spring
* Student:
 - Username: Pesho
 - Track: Frontend
 - Course results:
  * User has no course results!
* Course:
 - Name: JavaScriptOOP
 - Lectures per week: 2
 - Starting date: 1/24/2017 12:00:00 AM
 - Ending date: 2/23/2017 12:00:00 AM
 - Onsite students: 1
 - Online students: 0
 - Lectures:
  * There are no lectures in this course!
```

Section II - Tasks

CourseResult class

Create a **CourseResult** class that implements the ICourseResult interface. Each **CourseResult** should have a valid **Course**. The **ExamPoints** should be between 0 and 1000, if not, the error message "*Course result's exam points should be between 0 and 1000!*" should be thrown. The **CoursePoints** should be between 0 and 125 and their error message should be **"Course result's course points should be between 0 and 125!"**. The **Grade** can be Excellent, Passed or Failed and it is calculated based on the points. If the **ExamPoints** are equal or above 65 or the **CoursePoints** are equal or above 75, the grade should be Excellent. If the **ExamPoints** are

below 60 but equal or above 30, or the **CoursePoints** are below 75 but equal or above 45, the grade should be Passed. Any score below that should be Failed. The **Student** must be signed up for that course in order to create a **CourseResult**. The **CourseResult** must print to the following format:

```
  * <Course name>: Points - <Course points>, Grade - <Grade>
```

A student that contains course results should have the following format (Note that the inner class is indented with two spaces):

```
* Student:
 - Username: <username>
 - Track: <track>
 - Course results:
  * <Course name>: Points - <Course points>, Grade - <Grade>
  * <Course name>: Points - <Course points>, Grade - <Grade>
```

Lecture class

Create a **Lecture** class that implements the ILecture interface. Each **Lecture** has a **Name** that has a lenght between 5 and 30 symbols. If that is invalid, the error message "*Lecture's name should be between 5 and 30 symbols long!*" should be thrown. Each **Lecture** has a **Date**, **Trainer** and a **collection of resources**. The **Lecture** must be convertable to a string in the following format if it has no resources:

```
  * Lecture:
   - Name: <name>
   - Date: <date>
   - Trainer username: <username>
   - Resources:
    * There are no resources in this lecture.
```

Section II - Zero test

Input

```
CreateSeason 2016 2017 SoftwareAcademy
CreateStudent Pesho Frontend
CreateTrainer Cyku JS,C#,Haskell
AddStudentToSeason Pesho 0
AddTrainerToSeason Cyku 0
CreateCourse 0 JavaScriptOOP 2 2017-01-24
CreateLecture 0 0 Introduction 2017-01-25 Cyku
CreateLecture 0 0 Functions 2017-01-25 Cyku
AddStudentToCourse Pesho 0 0 onsite
```

```
CreateCourseResult 0 0 85 92 Pesho
ListUsersInSeason 0
ListCoursesInSeason 0
Exit
```

Output

```
SSeason with ID 0 was created.
Student with ID 0 was created.
Trainer with ID 0 was created.
Student Pesho was added to Season 0.
Trainer Cyku was assigned to Season 0.
Course with ID 0 was created in Season 0.
Lecture with ID 0 was created in course 0.JavaScriptOOP.
Lecture with ID 1 was created in course 0.JavaScriptOOP.
Student Pesho was added to Course 0.JavaScriptOOP.
Course result with ID 0 was created for Student Pesho.
* Trainer:
 - Username: Cyku
 - Technologies: JS; C#; Haskell
* Student:
 - Username: Pesho
 - Track: Frontend
 - Course results:
  * JavaScriptOOP: Points - 92, Grade - Excellent
* Course:
 - Name: JavaScriptOOP
 - Lectures per week: 2
 - Starting date: 1/24/2017 12:00:00 AM
 - Ending date: 2/23/2017 12:00:00 AM
 - Onsite students: 1
 - Online students: 0
 - Lectures:
  * Lecture:
   - Name: Introduction
   - Date: 1/25/2017 12:00:00 AM
   - Trainer username: Cyku
   - Resources:
    * There are no resources in this lecture.
  * Lecture:
   - Name: Functions
   - Date: 1/25/2017 12:00:00 AM
   - Trainer username: Cyku
   - Resources:
    * There are no resources in this lecture.
```

Section III - Tasks

DemoResource class

Create a **DemoResource** class that implements the ILectureResource interface. The **DemoResource** should have a **Name** that is a non-empty string, between 3 and 15 symbols long. If that is invalid, the error message "*Resource name should be between 3 and 15 symbols long!*" should be thrown. It also must have a **Url** that is a non-empty string between 5 and 150 symbols long. If that is invalid, the error message "*Resource url should be between 5 and 150 symbols long!*" should be thrown. The resource must print in the following format:

```
* Resource:
 - Name: <name>
 - Url: <url>
 - Type: Demo
```

HomeworkResource class

Create a **HomeworkResource** class that implements the ILectureResource interface. The **HomeworkResource** should have a **Name** that is a non-empty string, between 3 and 15 symbols long. If that is invalid, the error message "*Resource name should be between 3 and 15 symbols long!*" should be thrown. It also must have a **Url** that is a non-empty string between 5 and 150 symbols long. If that is invalid, the error message "*Resource url should be between 5 and 150 symbols long!*" should be thrown. It also has a **DueDate** that is automatically set to 7 days from now upon creation in the factory. The resource must print in the following format:

```
* Resource:
 - Name: <name>
 - Url: <url>
 - Type: Homework
 - Due date: <due date>
```

PresentationResource class

Create a **PresentationResource** class that implements the ILectureResource interface. The **PresentationResource** should have a **Name** that is a non-empty string, between 3 and 15 symbols long. If that is invalid, the error message "*Resource name should be between 3 and 15 symbols long!*" should be thrown. It also must have a **Url** that is a non-empty string between 5 and 150 symbols long. If that is invalid, the error message "*Resource url should be between 5 and 150 symbols long!*" should be thrown. The resource must print in the following format:

```
* Resource:
 - Name: <name>
 - Url: <url>
 - Type: Presentation
```

VideoResource class

Create a **VideoResource** class that implements the ILectureResource interface. The **VideoResource** should have a **Name** that is a non-empty string, between 3 and 15 symbols long. If that is invalid, the error message *"Resource name should be between 3 and 15 symbols long!"* should be thrown. It also must have a **Url** that is a non-empty string between 5 and 150 symbols long. If that is invalid, the error message *"Resource url should be between 5 and 150 symbols long!"* should be thrown. It also has a **UploadedOn** date that is automatically set to the date of creation in the factory. The resource must print in the following format:

```
* Resource:
 - Name: <name>
 - Url: <url>
 - Type: Video
 - Uploaded on: <uploaded date>
```

Section III - Zero Test

Input

```
CreateSeason 2016 2017 SoftwareAcademy
CreateStudent Pesho Frontend
CreateTrainer Cyku JS,C#,Haskell
AddStudentToSeason Pesho 0
AddTrainerToSeason Cyku 0
CreateCourse 0 JavaScriptOOP 2 2017-01-24
CreateLecture 0 0 Introduction 2017-01-25 Cyku
CreateLecture 0 0 Functions 2017-01-25 Cyku
CreateLectureResource 0 0 0 video Intro https://www.youtube.com/watch?
v=nhRJZ06JFxk
CreateLectureResource 0 0 1 homework Closures
https://github.com/TelerikAcademy/JavaScript-
OOP/tree/master/Topics/03.%20Closures-and-Scope/homework
AddStudentToCourse Pesho 0 0 onsite
CreateCourseResult 0 0 85 92 Pesho
ListCoursesInSeason 0
ListUsersInSeason 0
Exit
```

Output

```
Season with ID 0 was created.
Student with ID 0 was created.
Trainer with ID 0 was created.
Student Pesho was added to Season 0.
Trainer Cyku was assigned to Season 0.
```

```
Course with ID 0 was created in Season 0.
Lecture with ID 0 was created in course 0.JavaScriptOOP.
Lecture with ID 1 was created in course 0.JavaScriptOOP.
Lecture resource with ID 0 was created in Lecture
0.JavaScriptOOP.Introduction.
Lecture resource with ID 0 was created in Lecture 0.JavaScriptOOP.Functions.
Student Pesho was added to Course 0.JavaScriptOOP.
Course result with ID 0 was created for Student Pesho.
* Course:
 - Name: JavaScriptOOP
 - Lectures per week: 2
 - Starting date: 1/24/2017 12:00:00 AM
 - Ending date: 2/23/2017 12:00:00 AM
 - Onsite students: 1
 - Online students: 0
 - Lectures:
  * Lecture:
   - Name: Introduction
   - Date: 1/25/2017 12:00:00 AM
   - Trainer username: Cyku
   - Resources:
    * Resource:
     - Name: Intro
     - Url: https://www.youtube.com/watch?v=nhRJZ06JFxk
     - Type: Video
     - Uploaded on: 1/16/2017 12:00:00 AM
  * Lecture:
   - Name: Functions
   - Date: 1/25/2017 12:00:00 AM
   - Trainer username: Cyku
   - Resources:
    * Resource:
     - Name: Closures
     - Url: https://github.com/TelerikAcademy/JavaScript-
OOP/tree/master/Topics/03.%20Closures-and-Scope/homework
     - Type: Homework
     - Due date: 1/23/2017 12:00:00 AM
* Trainer:
 - Username: Cyku
 - Technologies: JS; C#; Haskell
* Student:
 - Username: Pesho
 - Track: Frontend
 - Course results:
  * JavaScriptOOP: Points - 92, Grade - Excellent
```

Section IV - Tasks

ListUsersCommand

Implement the **ListUsers** command in the **Commands.Listing** namespace. It should list all **Trainers** and

Students (in that order) from the **Engine**. If there are no users (no Trainers nor Students), the error message "*There are no registered users!*" should be thrown.

Section IV - Zero Test

Input

```
CreateStudent Pesho Frontend
CreateStudent Gosho Dev
CreateTrainer Mirto ASP.NET,Angular2
CreateTrainer Duncu ASP.NET,nodeJS
ListUsers
Exit
```

Output

```
Student with ID 0 was created.
Student with ID 1 was created.
Trainer with ID 0 was created.
Trainer with ID 1 was created.
* Trainer:
 - Username: Mirto
 - Technologies: ASP.NET; Angular2
* Trainer:
 - Username: Duncu
 - Technologies: ASP.NET; nodeJS
* Student:
 - Username: Pesho
 - Track: Frontend
 - Course results:
  * User has no course results!
* Student:
 - Username: Gosho
 - Track: Dev
 - Course results:
  * User has no course results!
```