# Object-Oriented Programming – Practical Exam

## Problem 1 – Software Academy

A software academy teaches two types of **courses**: **local courses** that are held in some of the academy’s local labs and **offsite courses** held in some other town outside of the academy’s headquarters. Each course has a **name**, a **teacher** assigned to teach it and a course program (sequence of **topics**). Each **teacher** has a name and knows the courses he or she teaches. Both courses and teachers could be printed in human-readable text form.

### Design the Class Hierarchy

Your **first task** is to **design an object-oriented class hierarchy** to model the software academy, courses and teachers **using the best practices for object-oriented design (OOD) and object-oriented programming (OOP)**. Avoid duplicated code though abstraction, inheritance, and polymorphism and encapsulate correctly all fields.

You are given few C# **interfaces** that you should **obligatory** implement and use as a basis of your code:

|  |
| --- |
| namespace SoftwareAcademy  {  public interface ITeacher  {  string Name { get; set; }  void AddCourse(ICourse course);  string ToString();  }  public interface ICourse  {  string Name { get; set; }  ITeacher Teacher { get; set; }  void AddTopic(string topic);  string ToString();  }  public interface ILocalCourse : ICourse  {  string Lab { get; set; }  }  public interface IOffsiteCourse : ICourse  {  string Town { get; set; }  }  public interface ICourseFactory  {  ITeacher CreateTeacher(string name);  ILocalCourse CreateLocalCourse(string name, ITeacher teacher, string lab);  IOffsiteCourse CreateOffsiteCourse(string name, ITeacher teacher, string town);  }  public class CourseFactory : ICourseFactory  {  // TODO: implement the interface  }  } |

All your courses should implement ICourse. Teachers should implement ITeacher. Local and offsite courses should implement ILocalCourse and IOffsiteCourse respectively. Courses and teachers should be created only through the ICourseFactory interface implemented by a class named CourseFactory.

The ITeacher.ToString() method returns the information about a teacher in the following form:

|  |
| --- |
| **Teacher: Name=*(teacher name)*; Courses=[*(course names – comma separated)*]** |

The courses added to a certain teacher (though the AddCourse(…) method) should appear in the order of their addition. If the teacher has no courses added, don’t print them. It is allowed to add the same course more than once. Some teachers might have no courses. Course names are separated by “, “ (comma + space).

The ICourse.ToString() method returns the information about a course in the following form:

|  |
| --- |
| ***(course type)*: Name=*(course name)*; Teacher=*(teacher name)*; Topics=[*(course topics – comma separated)*]; Lab=*(lab name – when applicable)*; Town=*(town name – when applicable)*;** |

The *(course* *type)* is either “LocalCourse“ or “OffsiteCourse”. The course lab and town are only shown when applicable. If the course has no teacher, it is not printed. The course topics added to a certain course (though the AddTopic(…) method) should appear in the order of their addition. It is allowed to add the same topic more than once. If the course has no topics added to it, do not print them. Do not print ; at the line end.

All properties in the above interfaces are mandatory (cannot be null or empty) except ICourse.Teacher, which is optional (a null value can be passed to the course factory).

If a null value is passed to some mandatory property, your program should throw ArgumentNullException.

### Write a Program to Execute C# Statements Using Your Classes

Your **second task** is to write a program that **executes a standard C# code block** (sequence of C# statements) using your classes and interfaces and prints the results on the console. A sample C# code block is given below:

|  |
| --- |
| **CourseFactory factory = new CourseFactory();**  **ITeacher nakov = factory.CreateTeacher("Svetlin Nakov");**  **Console.WriteLine(nakov);** |

The statements are guaranteed to be **valid C# sequences of statements** that will compile correctly if you implement correctly the interfaces and classes described above. The statements will be no more than **500**. Each statement will be less than **100 characters** long. The statements end with an empty line. The code block will not throw any exceptions at runtime and will not get into an endless loop.

### Additional Notes

To simplify your work you are given a C# code block execution engine that compiles and executes a sequence of C# statements read from the console using the classes and interfaces in your project (see the file SoftwareAcademy-Skeleton.rar). Please put all your code directly in the namespace “SoftwareAcademy”.

You are only **allowed to write classes**. You are **not allowed to modify the existing interfaces**.

### Sample Input

|  |
| --- |
| **CourseFactory factory = new CourseFactory();**  **ITeacher nakov = factory.CreateTeacher("Nakov");**  **Console.WriteLine(nakov);**  **nakov.Name = "Svetlin Nakov";**  **ICourse oop = factory.CreateLocalCourse("OOP", nakov, "Light");**  **oop.AddTopic("Using Classes and Objects");**  **oop.AddTopic("Defining Classes");**  **oop.AddTopic("OOP Principles");**  **oop.AddTopic("Teamwork");**  **oop.AddTopic("Exam Preparation");**  **Console.WriteLine(oop);**  **ICourse html = factory.CreateOffsiteCourse("HTML", nakov, "Plovdiv");**  **html.AddTopic("Using Classes and Objects");**  **html.AddTopic("Defining Classes");**  **html.AddTopic("OOP Principles");**  **html.AddTopic("Teamwork");**  **html.AddTopic("Exam Preparation");**  **Console.WriteLine(html);**  **nakov.AddCourse(oop);**  **nakov.AddCourse(html);**  **Console.WriteLine(nakov);**  **oop.Name = "Object-Oriented Programming";**  **(oop as ILocalCourse).Lab = "Enterprise";**  **oop.Teacher = factory.CreateTeacher("George Georgiev");**  **oop.AddTopic("Practical Exam");**  **Console.WriteLine(oop);**  **html.Name = "HTML Basics";**  **(html as IOffsiteCourse).Town = "Varna";**  **html.Teacher = oop.Teacher;**  **html.AddTopic("Practical Exam");**  **Console.WriteLine(html);**  **ICourse css = factory.CreateLocalCourse("CSS", null, "Ultimate");**  **Console.WriteLine(css);**  ***(empty line)*** |

### Sample Output

|  |
| --- |
| **Teacher: Name=Nakov**  **LocalCourse: Name=OOP; Teacher=Svetlin Nakov; Topics=[Using Classes and Objects, Defining Classes, OOP Principles, Teamwork, Exam Preparation]; Lab=Light**  **OffsiteCourse: Name=HTML; Teacher=Svetlin Nakov; Topics=[Using Classes and Objects, Defining Classes, OOP Principles, Teamwork, Exam Preparation]; Town=Plovdiv**  **Teacher: Name=Svetlin Nakov; Courses=[OOP, HTML]**  **LocalCourse: Name=Object-Oriented Programming; Teacher=George Georgiev; Topics=[Using Classes and Objects, Defining Classes, OOP Principles, Teamwork, Exam Preparation, Practical Exam]; Lab=Enterprise**  **OffsiteCourse: Name=HTML Basics; Teacher=George Georgiev; Topics=[Using Classes and Objects, Defining Classes, OOP Principles, Teamwork, Exam Preparation, Practical Exam]; Town=Varna**  **LocalCourse: Name=CSS; Lab=Ultimate** |