Homework: Inheritance and Abstraction

This document defines the homework assignments from the "OOP" Course @ Software University. Please submit as homework a single zip / rar / 7z archive holding the solutions (source code) of all below described problems. The solutions should be written in C#.

Problem 1. School

We are given a school. In the school, there are classes of students.

- Each class has a set of teachers.
- Each **teacher** teaches a **set of disciplines**.
- Students have name and unique class number. Classes have unique text identifier. Teachers have name. Disciplines have name, number of lectures and students. Both teachers and students are people. Students, classes, teachers and disciplines have **details** (optional field).

Your task is to identify the classes (in terms of OOP) and their members, encapsulate their fields, define the class hierarchy (inherit shared data and functionality) and create a class diagram with Visual Studio.

Problem 2. Human, Student and Worker

Define an abstract class Human holding a first name and a last name.

- Define a class **Student** derived from **Human** that has a field **faulty number** (5-10 digits / letters).
- Define a class Worker derived from Human with fields WeekSalary and WorkHoursPerDay and method **MoneyPerHour()** that returns the payment earned by hour by the worker.
- Define the proper constructors and properties for the classes in your class hierarchy.
- Initialize a list of 10 students and sort them by faculty number in ascending order (use LINQ or OrderBy() extension method). Initialize a list of 10 workers and sort them by payment per hour in descending order.
- Merge the lists and then sort them by first name and last name.

Problem 3. Animals

Create an abstract class Animal holding name, age and gender.

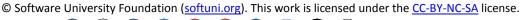
- Create a hierarchy with classes Dog, Frog, Cat, Kitten and Tomcat. Dogs, frogs and cats are animals. Kittens are female cats and Tomcats are male cats. Define useful constructors and methods.
- Define an interface **ISound** which implements the method **ProduceSound()**. All animals should implement this interface. The ProduceSound() method should produce a specific sound according to the animal (e.g. the dog should bark).
- Create arrays of different kinds of animals and calculate the average age of each kind of animal using LINQ.

Problem 4. Company Hierarchy

Create the following OOP class hierarchy:

- **Person** general class for anyone, holding **id**, **first name** and **last name**.
 - Employee general class for all employees, holding the field salary and department. The department can only be one of the following: Production, Accounting, Sales or Marketing.
 - Manager holds a set of employees under his command.
 - RegularEmployee
 - SalesEmployee holds a set of sales. A sale holds product name, date and price.





















- Developer holds a set of projects. A project holds project name, project start date, details and a state (open or closed). A project can be closed through the method CloseProject().
- o **Customer** holds the **net purchase amount** (total amount of money the customer has spent).

Extract **interfaces** for each class. (e.g. **IPerson**, **IEmployee**, **IManager**, etc.) The interfaces should hold their public properties and methods (e.g. **IPerson** should hold **id**, **first name** and **last name**). Each class should implement its respective interface.

Define proper constructors. Avoid code duplication through abstraction. Encapsulate all data and validate the input. Throw exceptions where necessary. Override **ToString()** in all classes to print detailed information about the object.

Create several employees of type Manager, SalesEmployee and Developer and add them in a **single** collection. Finally, print them in a for-each loop.

Problem 5. *** Business Report Application

Create a Windows desktop application that creates reports about the employees from the previous problem. The application should support the following functionality:

- **Import Data** button for importing the data to work with. When pressed, it loads the employee collection from the previous problem.
- Reports panel for displaying all regular employees and selecting a specific employee for report details.
- **Report Details** panel for visualizing data about the currently selected employee. Use the overridden **ToString()** methods to print the necessary information.
- **Export to Word** button for exporting all **checked** reports to word documents in the current directory. You are given a **class for generating Word document reports**, so you only need to figure out how to use it. (see the homework archive)
- **Export to DropBox** button that uploads all MS Word documents to a dropbox account on https://www.dropbox.com. Use an external library such as DropNet. Users are expected enter their DropBox credentials during the export to DropBox process.
- **About** button for displaying information about the application.

You are free to use any desktop UI system (Windows Forms / WPF / other). The screenshot below is merely a sample, so you can design your application differently.

