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. 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 **faculty 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 a LINQ query or the **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 2. 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 **ISoundProducible** which holds the method **ProduceSound()**. All animals should implement this interface. The **ProduceSound()** method should produce a specific sound depending on the animal invoking it (e.g. the dog should bark).
- Create an array of different kinds of animals and calculate the average age of each kind of animal using LINQ.

Problem 3. 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()**.
  - **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.