Apu Real Estate (ARE)

1 Objectives

In this assignment, you are to use the three principles of OOP, encapsulation, inheritance and polymorphism by implementing dynamic binding, interface, and abstract classes.

Notes:

- This assignment may be used again in the next module, so make sure to do a good job and keep your files for reuse. Do not forget to comment your code well and save your work quite often while you are writing code.

- This assignment is intended to be done using Windows Forms and controls. However, you may certainly try Windows Presentation Foundation (WPF) or MAUI

- If you would like to pick up and work on a project of your own, (for example a game or some order management system) you are free to do so provided that you are able to implement all the three principles of OOP as mentioned above. Using at least one interface, one abstract class and three concrete classes (subclasses) are mandatory. Consult your teacher before you start.

2 Description

Develop an application for a real estate company who can maintain a register of their objects currently on the market.

The real estate objects are grouped according the following criteria:

- Category – Residential, Commercial, and Institutional
- Type of residential buildings – villa, apartment, townhouse (row house).
- Type of commercial building – hotels, shops, warehouse, factories.
- Type of institutional buildings – hospitals, schools, universities
- Legal form– ownership, tenement or rental

Identify the object types, make a good design based on OOP and draw a class diagram, showing the classes and associations, before you start your programming. Determine specific attributes (fields) for each class and write the necessary methods. The class diagram is for your own use and does not have to be submitted.

Build an inheritance hierarchy wherever it is possible.

Interface ← Abstract classes ← Concrete classes

Use an interface to put some rules for the sub classes, e.g. all estate object must provide an ID (must have a get and a set property). Those classes that are not meant to be used by themselves instances or have some methods that cannot be implemented (and should be made abstract) should be declared as abstract.

Concrete classes are those which we need to create instances of and which store data (e.g. an apartment, a villa). You can have classes downward the hierarchy (not mandatory) that implement another interface.

3 Specifications and Requirements for a Pass grade (C)

3.1 The application must have a GUI. Create a Windows Forms (or a WPF) application using Visual Studio 2019 or higher. If you are working on a mac-computer, see the instructions given on the Course information module in Canvas.

3.2 For each class, determine the necessary fields and methods. Your design should have an interface **IEstate** and an abstract class **Estate** that implements **IEstate**. Create also three abstract classes **Commercial**, **Residential** and **Institutional** that inherit **Estate**.

3.3 Determine at least one abstract method in the **Estate** class that should be implemented by the subclasses, i.e. the classes Commercial, Residential and Institutional, or their subclasses.

3.4 Place the concrete classes, Apartment, Villa, Store (Shop), Warehouse and others, as listed in above description, as subclasses to their respective category class.

3.5 Hint: You can create two classes Rental (sw: hyresrätt) and Tenement (bostadsrätt) which inherit the Apartment class. A Rowhouse (Radhus) can inherit Villa.

3.6 All estate objects have an ID, an address consisting of data for street, zip code, city and country. The list of the countries is an enum and can be downloaded from the assignment page. Define these two requirements in the interface IEstate.

3.7 In this version, you have to create an application that is fully functional with features described below, for a **single** object (not a collection). As an example, the user should be able to add a Rental, or a Shop. The GUI should then be adjusted to transfer data from the user to the related objects.

3.7.1 **Add** (create) a new estate object of the types lowest in the hierarchy, e.g. a tenement or a rental, a warehouse or a shop. When the user selects a specific type of object, e.g. a Row House, the GUI should display the necessary controls (GUI components) for the fields of the estate type (e.g. number of rooms for an apartment), so that the user can give input for both the general and the specific attributes of that object. The general data is to be saved in the related base class and specific data are to be saved in the subclass.

3.7.2 **Change** the attributes of the object you have created.

3.7.3 **Delete** the current object.

3.7.4 It should also be possible to load and display an image from a file (one image is enough).

The code and GUI should be working well such that it should be easy be developed further in the next assign where we are going to store different estate objects using a collection.

3.8 When creating an object, use dynamic binding so the type of the Estate is determined at run-time depending on the user's choice. This means that you define a reference variable of the type Estate and then, depending on the type of object selected at run-time, make the variable refer to a concrete class. Refer to the reading material in the module to learn more about it.

```
Estate estate;  //referring to the base class

estate = new Apartment(…)
```

Note: You must show how and where in your code dynamic binding is implemented.

3.9 Test your application so it works as expected before submission. Projects that do not compile will be returned immediately for resubmission.

Optional: If you like to exercise more with interfaces, you can create an interface to be implemented by the Commercial class and another interface to be implemented by the Residential class.

Note: You may design the GUI, the solution and add or change functionalities as you find it necessary or more practical.

4 Specifications and Requirements for a higher (A, B) grade

In addition to the above requirements for a Pass grade, the following items are also to be implemented:

4.1 **For a B grade**: The application should have a feature to store some data on the seller (owner of the building) and the buyer of an estate. Create a base class Person and let buyer and seller inherit it. It is sufficient to include the name address of the seller and the buyer. The address should be an object, as explained above.

4.2 **For an A grade:** Include a payment system that include Bank, Western Union and PayPal. Create a class Payment (amount, options) serving as a base class to Bank (name and account number, WU (name, email), PayPal (email). The fields given in the parentheses are just examples. You may use other types of fields. The payment is not to be processed. It is enough to let the user select an option.

5 Some general quality considerations:

5.1 Convert the problem into as many classes as possible where each class has its own area of responsibility. Once you have a class, divide the tasks into methods. Write as many methods as required assigning one particular task to each method.

5.2 Some classes do not need to be a part of a hierarchy, e.g. the class Address. This class is neither a base class nor a sub class. It is an independent class by itself used by other classes through aggregation.

5.3 It is important that your solution is based on OOP and your code is well-structured and has good quality. Organize your classes in folders in VS.

5.4 Document your code by writing comments describing shortly the purpose of each class, methods of the class, and inside the code, wherever more clarification is necessary. Assume always that your code will be shared with or reviewed by others, and it should be easy to read and understand your thoughts.

5.5 Save your work as often as you can. Take always a backup of your good version so you can go back to an earlier version.

6 Grading and submission

The assignment is to be done and submitted individually. Compress your project to a zip, rar or 7z format and upload it to Canvas before showing it to your teacher. Be careful not to use any hard-coded file paths (for example path to an image file on your C-drive) in your source code, as it will not work on other computers.

Note: We are aware of the fact that solutions may be found on the Internet, and while you are allowed to study and learn from others' experience, it is strictly forbidden to copy code. All submissions in this course will be tested for plagiarism.

Good Luck!

Farid Naisan,
Course Responsible and Instructor