Apu Animal Park – Version 2
FoodSchedule

1 Interface IAnimal

```
interface IAnimal
{
    string Name { get; set; }
    string Id { get; set; }
    GenderType Gender { get; set; }

    FoodSchedule GetFoodSchedule();
    string GetExtraInfo();
}
```

2 The Animal class

All methods of the interface must be implemented in the Animal class as the class implements the interface IAnimal. By implementing of methods, it is meant that the methods must be coded in the Animal class and must have the same method definition. If the Animal class does not provide an implementation (method body), because of not having the needed information that can be applicable for all subclasses, the method can be declared abstract. This way, the Animal class leaves the implementation to sub-classes. Every subclass can then implement the method in its own way.

Write the following line in the Animal class:

```
public abstract FoodSchedule GetFoodSchedule();
```

The Animal class does not know what food schedule is good for a certain animal such as dog, cat, lizard or butterfly. It therefore lets the subclasses provide the schedule. For this reason, the method is defined abstract with no implementation.

3 The Mammal class

The subclasses of the **Animal** class are the category-based classes **Mammal**, **Bird**, **Reptile** (etc.). Again, we have the same problem. Also, these classes are not capable of providing the correct food-schedule. They can, however, delegate the job to its subclasses.

Copy the abstract method definition to the classes Mammal, Bird and other category-classes n the hierarchy.

```
public abstract FoodSchedule GetFoodSchedule();
```

When an class contains at least one abstract method, the class must be declared abstract. Abstract classes cannot be instantiated and must be inherited.

Declare all the category classes, Mammal (and others) with the abstract keyword, because they now contain an abstract method:

```
abstract class Animal: IAnimal
```

The **GetFoodSchedule** method, as described above must be coded in each concrete class, Dog, Cat, ButterFly, etc.

4 The concrete classes – Dog as an example

We have now reached the point at which we can arrange a food schedule. Take the Dog class as an example and then proceed in the same way with other classes that are subclass to a category class like Mammal, Bird ,etc.

The Dog class:

```
private FoodSchedule foodSchedule;

//this method is called from the constructor
//for simplicity - the info is hard-coded
private void SetFoodSchedule()
{
    foodSchedule = new FoodSchedule();
    foodSchedule.EaterType = EaterType.Omnivorous;
    foodSchedule.Add("Morning: Flakes and milk");
    foodSchedule.Add("Lunch: bones and flakes");
    foodSchedule.Add("Evening any meat dish.");
}
public override FoodSchedule GetFoodSchedule()
{
    return foodSchedule;
}
```

5 MainForm

When the user selects an animal among the list of the registered animals, call the abstract method GetFoodSchedule and display the contents in the dedicated control.

Double-click on the ListBox showing int the list of animals (middle ListBox on the GUI), and complete the method that VS prepares for you with code shown in the clip below. Note that the name of the ListBox control is lstAnimal.

```
private void lstAnimals_SelectedIndexChanged(object sender, EventArgs e)
{
    int index = lstAnimals.SelectedIndex;
    if (index == -1)
        return;

    Animal animal = manager.GetAnimalAt(index);

    FoodSchedule foodSchedule = animal.GetFoodSchedule();

    lstFoodSchedule.Items.Clear();
    lstFoodSchedule.Items.Add(foodSchedule.Title());

    lblEaterType.Text = foodSchedule.EaterType.ToString();
    string[] foodList = foodSchedule.GetFoodListInfoStrings();

    lstFoodSchedule.Items.Clear();
    lstFoodSchedule.Items.AddRange(foodList);

    lblExtraInfo.Text = animal.GetExtraInfo();
}
```

6 Parts of the FoodSchedule class

```
class FoodSchedule
{
    private EaterType eaterType;
    private List<string> foodList; //morning: xxx, Lunch = xxx


    public FoodSchedule()
    {
        foodList = new List<string>();
    }

    public EaterType EaterType
    {
        get { return eaterType; }
        set { eaterType = value; }
    }


   /// <summary>
    /// Add method
    /// </summary>
    /// <param name="item"></param>
    public void Add(string item)
    {
        foodList.Add(item);
    }
```

```
public string[] GetFoodListInfoStrings()
{
    string[] infoStrings = foodList.ToArray();

    return infoStrings;
}
```

Good Luck!

Farid Naisan,
Course Responsible and Instructor