

## Apu's Animal Park – Version 3

### 1 Objectives

Collections are widely used to save groups of objects of the same or different types. Data as lists of objects occur in all types of information systems. Generics are another powerful feature that not only allows programmer to create type-safe data structures without worrying about the exact type of data, they also make use of collections extraordinarily effective. Using these two concepts within a polymorphic framework with the support of .NET adds a power to your programming and makes your code very flexible.

The main objectives of this assignment are:

- Create generic types.
- Use collections with generic objects.

.NET offers a huge number of collection types that are ready for programmers to use. In this assignment, we are going to continue working with the type **List<T>** (which you are already familiar with) and **Dictionary** (for higher grades).

#### Notes:

- This assignment will 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.
- You can continue with the same GUI system you used in the previous assignment, Windows Forms, WPF or MAUI. It is of course allowed to change the GUI or the application platform that you used in the previous assignment and experiment with a new one.
- You do not have to follow the instructions given here step by step and you may change the GUI and implement your own solution, provided you keep a good programming level and follow the OOP rules.

### 2 Description

In our daily life, we use lists and tables to group data in various forms. We need to prepare lists of products, customers, friends, study schema, timetables and other such collections of data to organize our systems.

We have containers such as drawers, boxes, shelves, buckets and all types of holders that store our object. In much the same way we are going to create a container class to store and handle a list of objects.

In most cases when working a list of objects, we have similar actions to take, e.g., adding a new object to the list, changing an existing object, removing an object from the list and saving or retrieving the list to and from a data source (file, or database).

A typical solution for handling a list of objects is to create a container class, define a collection in the class as a field and then write methods to perform the above-mentioned operations on the collection. We need to write methods to perform the following common operations.

- Add an object to the list.
- Remove an object from the list.
- Replace an existing object with a new object (or change its attributes).
- Insert an object at certain position in the list.
- Return an object from the list.
- Get info for an object.
- Search for an object.
- Provide a list with some information on all of the elements of the list.
- Saving or retrieving data to and from a data source. However, we let this feature be a part of one of the next assignments.

Each time we have a collection, we need to program the above methods. With generics, we can program these operations using a class with data and methods that would work with any type of an object. Then, we reuse it for lists containing different type of objects, for example `List<Animal>`, `List<FoodItem>` or `List<BankAccount>`, without writing new code.

### 3    **ToDo:**

Create a generic **ListManager** class containing a list of T where T can be an `Animal` or a `FoodItem` or any other object type. The generic **ListManager** class will be capable of maintaining a collection of the type `List<T>` without knowing the actual type of the objects it will contain. The symbol "<T>" can be replaced by `<Animal>`, `<Employee>`, `<Student>`, `<Staff>`, `<int>`, `<bool>` or any other type. The code then will work for all the types.

The letter T (type parameter) is used as symbol for "Type" meaning any type, but it is not a keyword. You may use any other letter or words like `<TKey>` or `<Elem>`.

To make the model even more flexible, we write an interface **ICollectionManager** in which we define all the necessary operations as methods of the interface. We let **ListManager** implement this interface.

In this assignment, we are going to create a new interface, and a number of new classes:

**ICollectionManager**, a generic interface

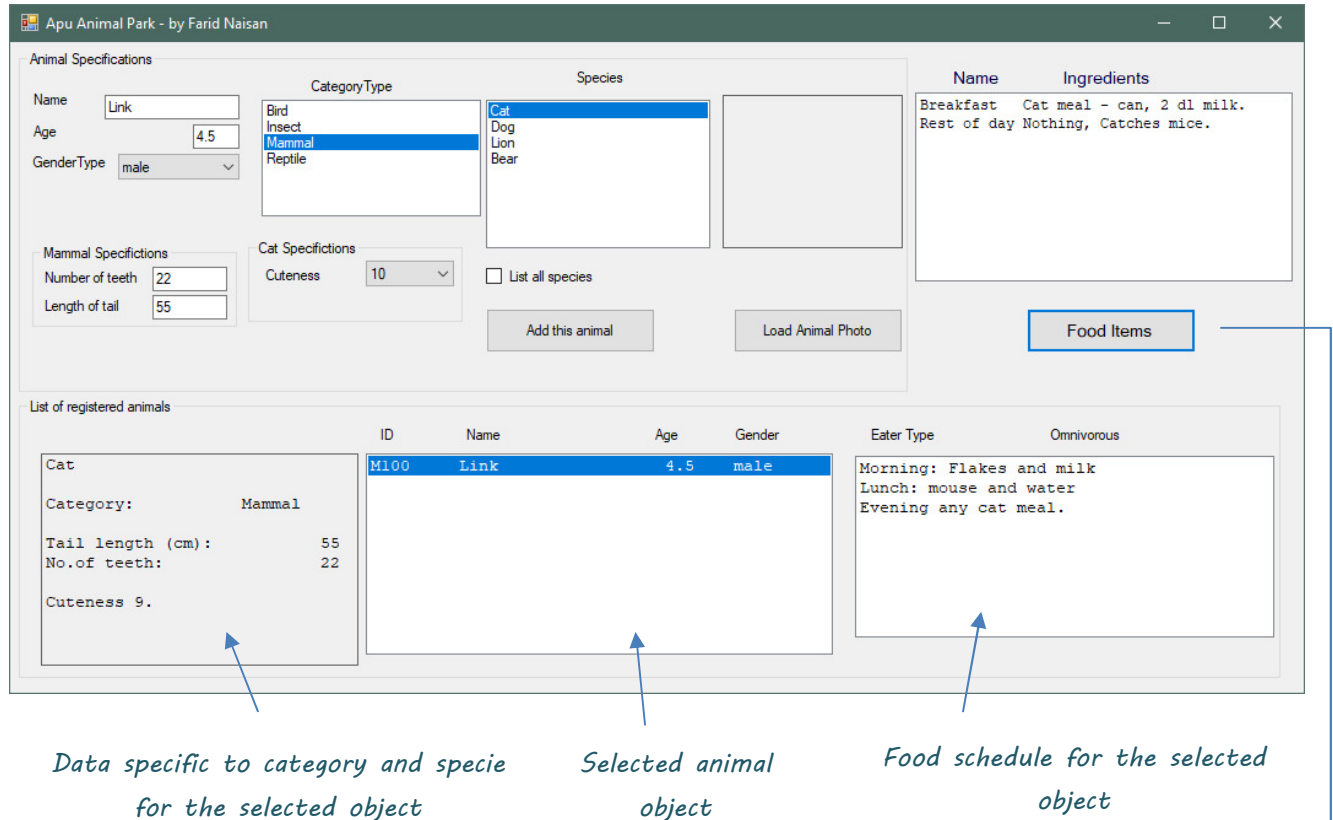
**listManager**, generic class, implementing **ICollectionManager**

**FoodItem**, class with ID, name and ingredients as fields. For ingredients, use an instance of **ListManager**.

**FoodForm**, a GUI class, providing GUI for user input related to **FoodItem**.

Furthermore, the application should allow the user to change or delete an existing item from the list of animals and the list of food items..

A run-time example of the program is presented below. A more detailed description of the above follows.



*Data specific to category and specie for the selected object*

*Selected animal object*

*Food schedule for the selected object*

3.1 When an item (an animal) in the middle ListBox is highlighted (selected), the data for that animal should updated in the left and right components.

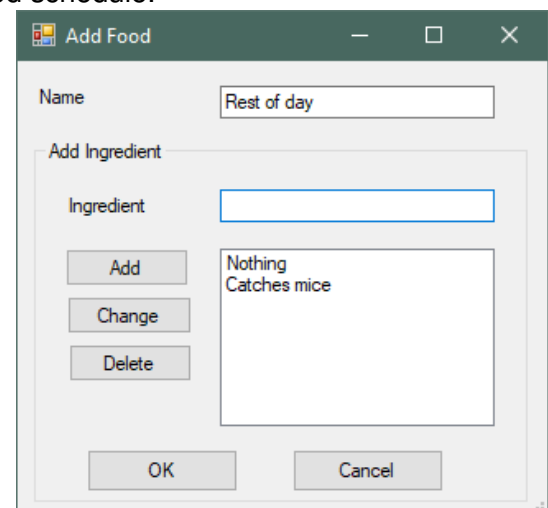
3.1.1 The left-side component (label, ListBox or ListView) is to display data saved in the category class of the selected item.

3.1.2 The right-side component should show the food schedule.

3.2 When the user clicks on the Food Items button on the MainForm (above), a new form is to be opened for input which should be saved in an instance of FoodItem

3.3 The details of the FoodItem class is described later.

3.4 To limit the amount of work, the food items do not have to be associated to an animal object for those who are aiming a C grade. It is sufficient to display the items in a ListBox (or any other component) on the **Mainform**, as in the above figure (ListBox at the upper-right corner).

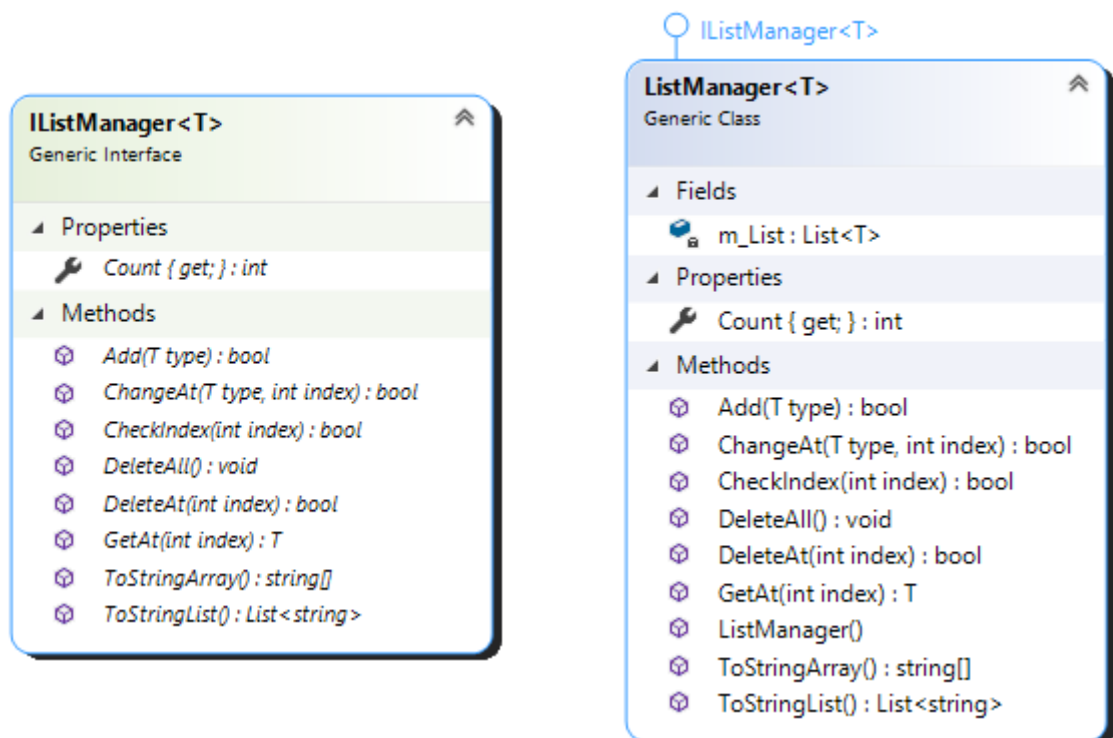


## Generic types: IListManager and ListManager

3.5 Create an interface **IListManager**. Define a number of methods to perform different operations on objects of collection of the type `List<T>`.

- 3.5.1 Add a new object to the list.
- 3.5.2 Remove an object from the list at a certain position
- 3.5.3 Replace (change) an object in a given position with a new object.
- 3.5.4 Return an object from a certain position in the list.
- 3.5.5 Return an array of strings where every string represents the object (calling the `ToString()` of the object).
- 3.5.6 Create a class **ListManager** that implements **IListManager**. The class should have a list of T object using the `List<T>` collection as an instance variable. It should of course implement the methods of the interface.
- 3.5.7 Do necessary validation of the arguments of the methods, i.e. objects should not be null and indexes should not go out of bounds.
- 3.5.8 Each method of the interface must be well-documented so it can be used in other projects and other programmers (an example is given later in this document).

The class diagrams below are given as a guidance. You do not have to define and implement the methods that are not listed in the requirements above.



## The Class - AnimalManager

- 3.6 The **AnimalManager** class can now make use of the generic ListManager by inheriting the latter.

```
class AnimalManager : ListManager<Animal>
{
}

```

If it were not for the ID, this class could be just empty because all the operation it would need is provided by its base class, the **ListManager**.

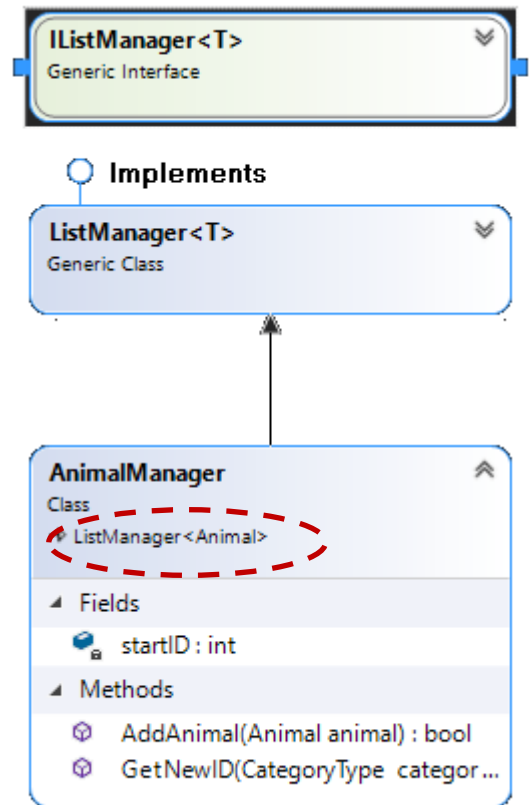
The method AddAnimal is called from MainForm. The methods assigns an ID and then it calls its base class Add method.

```
public bool AddAnimal(Animal animal)
{
    bool ok = false;

    if (animal != null)
    {
        animal.Id = GetNewID(animal.Category);
        Add(animal);
        ok = true;
    }

    return ok;
}

```



Other methods can directly be called from the MainForm. For example: **animalMngr.RemoveAt(index)** will automatically invoke the **RemoveAt** in the **ListManager** class through inheritance.

## The class MainForm

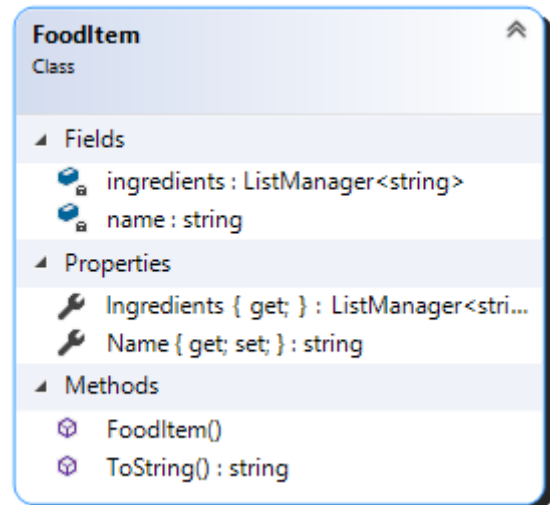
- 3.7 You have to provide a **Change** and a **Delete** button on the **MainForm** to allow the user to add a new Animal and remove an existing Animal object using **animalManager** (see the GUI example above). Make these features work.
- 3.8 Bring changes so **MainForm** does not call any other method than **AddAnimal** method from the manager class. As an example, to get an element from the manager, call **animalManager.GetAt** and to get an array of strings representing the elements of the list, you can use the following code:

```
string[] infoStrings = animalManager.ToStringArray();

```

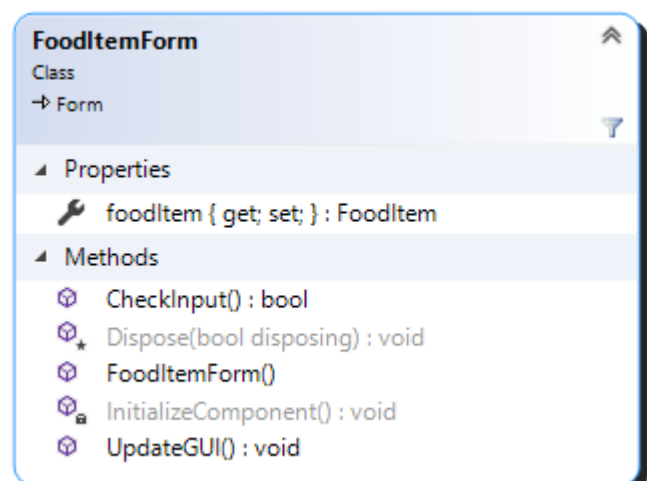
## The FoodItem class

- 3.9 The purpose of creating this class is to allow the user to provide a food schedule when running the program instead of hardcoding. As the class diagram shows, the class has two instance variables, name and ingredients.
- 3.10 Ingredients is a collection of the type ListManager. It directly can call the methods to add, remove or change elements without writing any manager class.



## The FoodItemForm

- 3.11 Create and design a form to add food items and display them on the MainForm.
- 3.12 You can use the following code connected to the Food Items button (see the GUI image given earlier):



```

private void btnAddStaff_Click(object sender, EventArgs e)
{
    FoodItemForm recipeForm = new FoodItemForm();
    if (recipeForm.ShowDialog() == DialogResult.OK)
    {
        lstRecipeList.Items.Add(recipeForm.foodItem.ToString());
    }
}
  
```

## 4 Specifications and Requirements for Grades (B and A)

### For Grade B

Do the following in addition to the all the requirements for a Grade C (except for the food items):

- 4.1 Design a solution to a Dictionary<TKey, TValue> where TKey is the ID (name) of a foodItem and TValue is a list of animal IDs (or names) that the foodItem is connected to.

A suggestion:

- Create a class **FoodManager**
- Define an array of FoodItem to store food items that are added using the FoodItemForm on the main GUI. Every FoodItem has then a name and a list of strings (e.g. Ingredients). You can use the ListManager class or an array of strings. for the ingredients
- The user should be able to connect an animal to a food item. The user selects an animal from the animal list and then selects a food item in the food item list. These two should be stored as a key-value pair in the Dictionary object. This object can be stored in the **FoodManager** class, or possibly in the **AnimalManager**. Make your own judgement.
- If more than one animal selects the same food item, it should not be a problem as the **Value** is an array.

The coding for the above (or a similar specification) is sufficient for a grade B. No GUI handling is required in this part as this will be a requirement for a grade A

### For Grade A

In addition to requirements for a B-grade, do the following:

- 4.2 Apply the above solution instead of the **FoodSchedule** from the previous assignment.

In both of the above two requirements, you are free to use your own judgement and decisions to come to a proper solution.

## 5 Some Help and Guidance

The interface should be documented using comments so it can easily be implemented in different cases for different object types.

Here is an example:

```

/// <summary>
/// Interface for implementation by manager classes hosting a collection
/// of the type List<T> where T can be any object type. In this documentation,
/// the collection is referred to as m_list.
/// IListManager can be implemented by different classes passing any type <T> at
/// declaration but then T MUST HAVE THE SAME TYPE IN ALL METHODS INCLUDED IN THIS
/// INTERFACE.
/// </summary>
/// <typeparam name="T">object type</typeparam>
interface IListManager<T>
{
    /// <summary>
    /// Return the number of items in the collection m_list
    /// </summary>
    int Count { get; }

    /// <summary>
    /// Add an object to the collection m_list.
    /// </summary>
    /// <param name="aType">A type.</param>
    /// <returns>True if successful, false otherwise.</returns>
    bool Add(T aType);

    /// <summary>
    /// Remove an object from the collection m_list at
    /// a given position.
    /// </summary>
    /// <param name="anIndex">Index to object that is to be removed.</param>
    /// <returns>True if successful, false otherwise.</returns>
    bool DeleteAt(int anIndex);
}

```

- 5.1 Create a class **ListManager** that implements the interface **IListManager**. Here is how you can begin the class. Complete the rest.

```

public class ListManager<T> : IListManager<T>
{
    private List<T> list;

    public ListManager()
    {
        list = new List<T>();
    }
}

```

## 6 Submission

Compress all the files, folders and subfolders into a **zip** or **rar** file, and then upload it via the Assignment page on Canvas as before.

Good Luck!

**Farid Naisan,**

Course Responsible and Instructor