# Apu Animal Park – Version 4

## 1 Objectives

The main goals of this assignment are:

- Use serialization to store and load data in different ways using text files, and serialization in Jason, binary and XML formats
- Handle exceptions
- Improve the graphical user interface with menus.

The main idea is to save the list of animals registered in the application to a file so it can be loaded and transferred to the application at a later time. The following table summarizes the requirement for different grades. Detailed information follows the table.

| | | |
|---|---|---|
| Grade D | • Saving and retrieving the list of the animals in the AnimalManager using text files<br>• Serialize and deserialize the list of animals using Json serialization.<br>• Handling exceptions | Text files: save and read data defined in the Animal class plus the Category, skipping data for the specie, e.g. for a Dog object, data specified in the Dog class can be skipped. |
| Grade C | In addition to requirements for Grade D:<br>• Serialize and deserialize the list of animals using binary serialization. | Save all data for a species. |
| Grade B | In addition to requirements for Grade C:<br>• Serialize the data in the FoodManager object using XML serialization. | |
| Grade C | Create a user-defined Exception class and use it in your code | |

## 2 Description and requirement for a grade C

The last version of the Animal Park application which you produced in the previous assignment meets most of the requirements of the customer. However, there is a final task that remains to be completed, and that is to save data to files and read data from files. Without this feature, the program is not very useful. Your mission in this assignment is to produce Version 4 of the application with the following general requirements:

2.1 The application should provide a feature to store the animal registry, i.e. the data saved in the **AnimalManager** object, to a file on disk and another feature to load the data saved by the application back into the program.

2.2 Include a main menu in your project with menu items for New, Open, Save, Save As and Exit. All of the menu items should be functioning well.

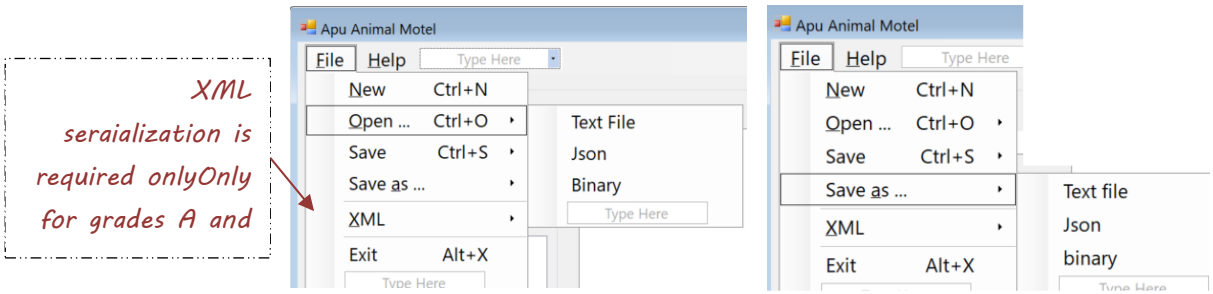
2.2.1 The File menu should have menu items to save and load animal data to and from a file in the following three different ways:

2.2.2 **Text file**, in any order you would like. In addition, to limit the amount of work, it is sufficient to store data for the category (Mammal, Bird, etc.) and data saved in the base class Animal (e.g. name, age, etc.). This is only valid for saving data to a text file.

2.2.3 **Json Serialization**, using either **JsonSerializer** or **JsonConvert** (JsonConvert is easier and rerecommended) to serialize and deserialize data in the Json format. All data, including the data specific to species, is to be saved.

2.2.4

2.2.5 **Binary** serialization with **BinaryFormatter,** to serialize and deserialize data.




Note: For a C grade, you do not have to include objects related to Food and Foomanager in saving data. This is requirement only for the A and B grades, as described below.

2.3 The user should be given the possibility of selecting a drive and a file on a directory on any drive when writing or reading files (use **OpenFileDialog** and **SaveFileDialog** controls).

2.4 Use error handling to secure other parts of your code where run-time error can be predictable.

2.5 **Exception**: Your application should work satisfactorily and should not fail for bugs, errors and unhandled exceptions thrown at run-time. Handle exceptions so the saving and loading files do not result in run-time errors. You may use the most related Exception class (such as FileNotFoundException) to catch error, but using the base class Exception is sufficient.

## For grades B:

2.6 In addition to the above requirements, the food registry (**FoodManager** object) should be saved to files in the XML format (using **xmlSerializer**).

2.7 Read an XML file saved by your application (XML deserialization).

**For grades A:**

2.8 Create a simple custom Exception to handle a particular situation in your application and test it by throwing the exception in a proper place in your code. If you can't think of a situation in which a custom exception can be suitable, you can write a custom Exception to make sure that each animal has some type of a food schema (or food schedule).

## To Do:

Copy the last version of your application to a new directory to do this assignment. You last version will be then a good back-up version. This assignment has two parts:

Part 1: Data persistence and menus

Part 2: Handling Exceptions and errors

# 3 Part 1: Data persistence and menus

**GUI Enhancements**

3.1 Provide a **File** menu with the menu items shown in the picture, using a **MenuStrip** component. For your convenience, name the **MenuStrip** control as **mnuFile** and use this as prefix to the names of the submenus. The **Open** submenu can be then called **mnuFileOpen**, and the submenu **Save As** may be named as **mnuFileSaveAs**.

3.2 The **New** submenu should initialize the program exactly as at start-up (but without restarting the application). If data has not been saved, allow the user (through a **MessageBox**) to confirm proceeding without saving current data or go back to the current session.

3.3 The **Open** submenu should open a text file or a binary file according to the user's option (text file, Json, Binary). The program must remember the type for the file so the menu **Save** can save the data correctly.

3.4 The **Save as** submenu lets the user to save data to either a text file, a Json fiel or a binary file with a name and location selected by the user. The program must remember the type for the file so the menu **Save** can save the data correctly.

3.5 The **Save** submenu saves the data using the current data file by binary serialization. When a data file has not yet been selected by the user, the application should call the **Save As** menu automatically. You can test the filename if it contains character or if it is empty or null.

3.6 Use the Windows Form controls, **OpenFileDialog** and **SaveFileDialog** for the **Open** and **Save As** submenus, respectively, to let the user choose the path and name of the data file.

**Structural and other requirements**

3.7 Make all animal objects contained in the **AnimalManager** class **serializable** so data can be saved to and opened from files.

3.8 Open the interface **IListManager** (from the previous assignment) and add the method definitions shown in the image below. You may use void instead of bool

```
//Each of the methods returns true if the operation
//completes successfully, and false otherwise.
//Use JsonConvert as it is more convenient
1 reference
bool JsonSerialize(string fileName);
//JsonDeserialize optional for all grades
1 reference
bool JsonDeserialize(string fileName);
1 reference
bool BinarySerialize(string fileName);
1 reference
bool BinaryDeSerialize(string fileName);
//XML - required only for grades A and B
1 reference
bool XMLSerialize(string fileName);
```

3.9 Implement the methods in the **ListManager** class.

3.10 **Important:** You may face a problem and get an exception deserializing a Json string from the file when opening a Json file that you have serialized before. This happens because the Animal class is abstract and Json cannot create instances of it. You do not have to fix this problem; it is sufficient that you have code for deserialization using a try-catch statement.

**HINT (optional)**: Create a UtilitiesLibrary, with two utility classes: **BinSerializerUtility**, and **XMLSerializerUtility** for hosting all general binary and xml serialization respectively. You can then call the methods of the classes from the **ListManager** class when implementing the interface methods.

3.11 Animal images do not need to be serialized.

3.12 **Do not hard code any file or directory paths**! All file paths are to be relative to the Project Directory (Application.StarupPath) or standard Windows folders (Users, My Documents). No hard-drive letters such as "C:\" should be used in your code. Use .NET objects to programatically access standard folders (if you will use such folders). The enumeration **Environment.SpecialFolder** has useful information about the system special folders.

3.13 **Important**: Messages from catching serialiserings exceptions should **NOT** be given to the user from the **Utility** class (**NOT** from the **Manager** class either). The messages must be relayed to the GUI which is responsible for all interactions with the user. A

possible solution is not to catch exceptions in the **Utility** class and instead do it in the GUI-class. However, a **try-finally** or a **using { }** statement is necessary in the serialization methods to ensure closing of the stream. Here is an example of code for the GUI-class:

```
try
{
    myManager.OpenFile(fileName);
    UpdateGUI ( );
}
catch (Exception ex)
{
    MessageBox.Show ( "Show some informative text with or
without the use of ex!" );
}
finally
{
    filename.Close();
}
```

The exceptions falls through down the calling chain as long until it finds a **catch** block. To be on the safe side, passing the message from the Utility class to the caller is perhaps a better solution. Recall that ex.ToString() (in the catch block above) presents all details of the error to the user (Where it went wrong and Why). ex.Message is OK to use, but it usually says nothing useful. ex.ToString() is recommenden at least in debug mode, where you can find out why it's not working simply by looking at the exception!

3.14 **Include at least one demo data file of each serialization type** (Demo1.bin, Demo1.json, Demo1.xml) with your project when submitting your assignment so your teacher can use it for for testing your application. Make this a habbit whenever you release a version of an application. The end-users (as well as the testers) will undoubtedly appreciat one or more demo files that accompany the installation of your application.

## For Grade B : Persisting in XML format

In addition to the above requirements, the following must be done to qualify for a higher grade.

3.15 Write code to serialize the **FoodManager** objects when the user clicks on the sub-menu **Export to XML File**. To simply the problem, it is sufficient that only recipe names are written to XML file, as illustrated in the following listing. (The listing has been created as Demo1.xml in a test of the sample application).

**NOTE!** If you want to save the ingredients- (or qualifications) lists as well, CHANGE the collection from ListManager<string> to List<string> and it will work.

```
<?xml version="1.0"?>
<ArrayOfRecipe xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Recipe>
    <Name>Daily dog food</Name>
  </Recipe>
  <Recipe>
    <Name>Some other recipe</Name>
  </Recipe>
</ArrayOfRecipe>
```

**For Grade A : User-defined Exception**

3.16 **User-defined Exception** – this class needs not to be too complicated. For more information and examples, visit the following page:
https://learn.microsoft.com/en-us/dotnet/standard/exceptions/how-to-create-user-defined-exceptions

# 4 Part 2: Exceptions and error handling for all grades.

Your teachers will be testing your application and will return your application for resubmission at the first occurrence of an error due to careless coding. Make a good decision as to where you should use if-else, **Debug.Assert** (System.Diagnostics namespce) and where you should handle exceptions. Exception handling is costly (as far as performance is concerned) and therefore should not be used where you can use normal control statements.

4.1 Use a **try-catch** (or **using**) statement (or both if needed) to handle exceptions wherever you can anticipate system and run time errors.

4.2 All file handling and serializations (binary and XML) must contain handling of exceptions and the user should be alerted by a message about the type of error (the Message property of the Exception class is good enough).

4.3 Go through your code and ensure that your application will not crash for any reason. Test your application for all types of input to find bugs and errors.

# 5 Grading and submission

Make sure that you submit the correct version of your project and that you have compiled and tested your project before handing in. 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. It will not work on other computers. Projects that do not compile and run correctly will be directly returned for completion and resubmission.

Compress all the files, folders and subfolders into a **zip** or **rar** file, and then upload it via the Assignment page on Canvas. Click the button “Submit Answer” and attach your file. Do not send your project via mail!

Good Luck!

*Farid Naisan,*
Course Responsible and Instructor