C# assignments

Guidelines:

1. Create only single solution file. Add multiple projects to the same solution file if needed.
2. Solution should build without any errors.
3. C# coding guidelines should be strictly followed.
4. There should be a single CLI to navigate and run each exercise individually. Any exercise which cannot be run from the CLI will not be graded.
5. Provide a ReadMe file with steps to execute your code.
6. Use Visual studio (and not VSCode) for these attempting these assignments.
7. Do **NOT** submit *bin, obj, and .vs* folders.

Basic C#

Exercise 1 Get inputs from user and convert it into:

1. Integer
   a. Use *int.parse*
   b. Use *Convert.ToInt*
   c. Use *int.TryParse*
2. Float - Try different methods to do it
3. Boolean - Try different methods to do it

Exercise 2 Demonstrate the difference between "==" operator, *object.Equals* methods and *object.ReferenceEquals* method. Relevant code examples and comments should be provided.

Exercise 3 Write a program to print all prime numbers between two given numbers. Ask the user to input two positive non-equal integers (between 2 and 1000), first number entered should be smaller than larger number and if that's not the case, ask the user to re-enter both the numbers. Until user enters valid input, show appropriate message, and keep asking for the input again.

OOPS with C#

Exercise 4 Our client is an equipment owner company. It needs a system to manage its equipment inventory (CRUD on equipment).

Equipment has following details in common -

1. Name
2. Description
3. Distance Moved Till Date (Defaults to 0 km)
4. Maintenance Cost (Defaults to 0)
5. Type of equipment – Use *enum* to represent this value

Equipment should have THE following functionality in common:

1. Move By – This functionality is to move equipment by certain distance (in km). It should take the distance to move as parameter. This in turn cause the following
    a. Increase the "distance moved till date" by the distance equipment is made to move.
    b. Increases the "maintenance cost" – This depends on the kind of equipment. (Think Inheritance & Polymorphism)

All equipment is one of the following two kinds -

1. Mobile – This equipment can move around by itself (it has wheels) e.g., a JCB, a Jeep, a tractor etc.
   Mobile equipment has following specific details
    a. Number of wheels
    b. When they are moved around (via Move By method), the maintenance cost increases by (numberOfWheels * distanceMoved)

2. Immobile – This equipment must be carried around e.g., a trolley, a ladder etc.
   Immobile equipment has following specific details
    a. Weight
    b. When they are moved around (via Move By method), the maintenance cost increases by (weight * distance moved)

The application being created should fulfill the following requirements.

1. Create an equipment – mobile and immobile
2. Move an equipment – mobile and immobile (this is going to update certain properties in equipment)
3. Show details of an equipment. (All details – including distance moved till date and maintenance cost)

Use following keywords: virtual, override, abstract etc

User following concepts: class, abstract, inheritance, polymorphism etc.

Exercise 5 This assignment is supposed to make you practice your skills on interfaces, classes, and OOPs.

Story - The system is a duck simulation game. There are ducks, each having weight and number of wings. All ducks can fly and quack, but different type of ducks fly and quack differently. For instance, let us consider the following–

1. Rubber ducks don't fly and squeak.
2. Mallard ducks fly fast and quack loud.
3. Redhead ducks fly slow and quack mild.

All ducks have following common property:

1. Type of Duck – Use enum to represent this value

Design classes and interfaces for the above story to meet the following requirements

1. Create a duck
2. Show details of a duck, i.e. when you call for e.g. ShowDetails() method of a duck, duck should print its traits.

User following concepts: class, interfaces, polymorphism etc.

Collections

Exercise 6 Extend Assignment 4 (OOPS - Equipment Inventory assignment). Maintain a list of equipment in the application to do following operations:

1. Create an equipment – mobile and immobile
2. Delete an equipment
3. Move an equipment – mobile and immobile (this is going to update certain properties in equipment)
4. List all equipment. (Just the basic details – name and description)
5. Show details of an equipment. (All details – including distance moved till date and maintenance cost)
6. List all mobile equipment -> Use Lambda
7. List all Immobile equipment -> Use Lambda
8. List all equipment that have not been moved till now
9. Delete all equipment
10. Delete all immobile equipment
11. Delete all mobile equipment

Create a console application for the above scenario meeting all requirements. Use in-memory objects for storing data. Also make sure to validate user input and show appropriate messages for invalid inputs. (Try using Lambda, LINQ, Select, Where query etc. for this assignment)

Exercise 7 Extend exercise 5(OOPS – Duck assignment)

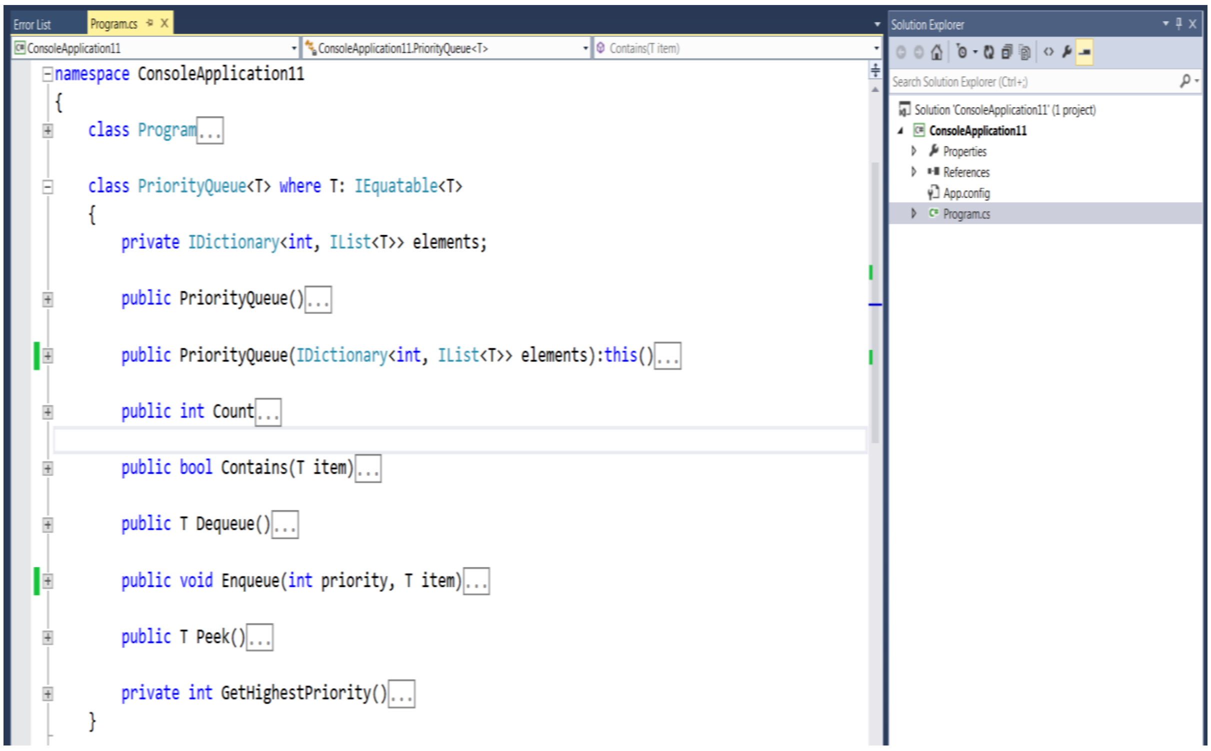
Maintain a list of ducks to do following operations

2. Add a duck
3. Remove a duck
4. Remove all ducks
5. Capability to iterate the duck collection in increasing order of their weights. This should be the collections default iteration behaviour
6. Capability to iterate the duck collection in increasing order of number of wings.

Collections and Generics

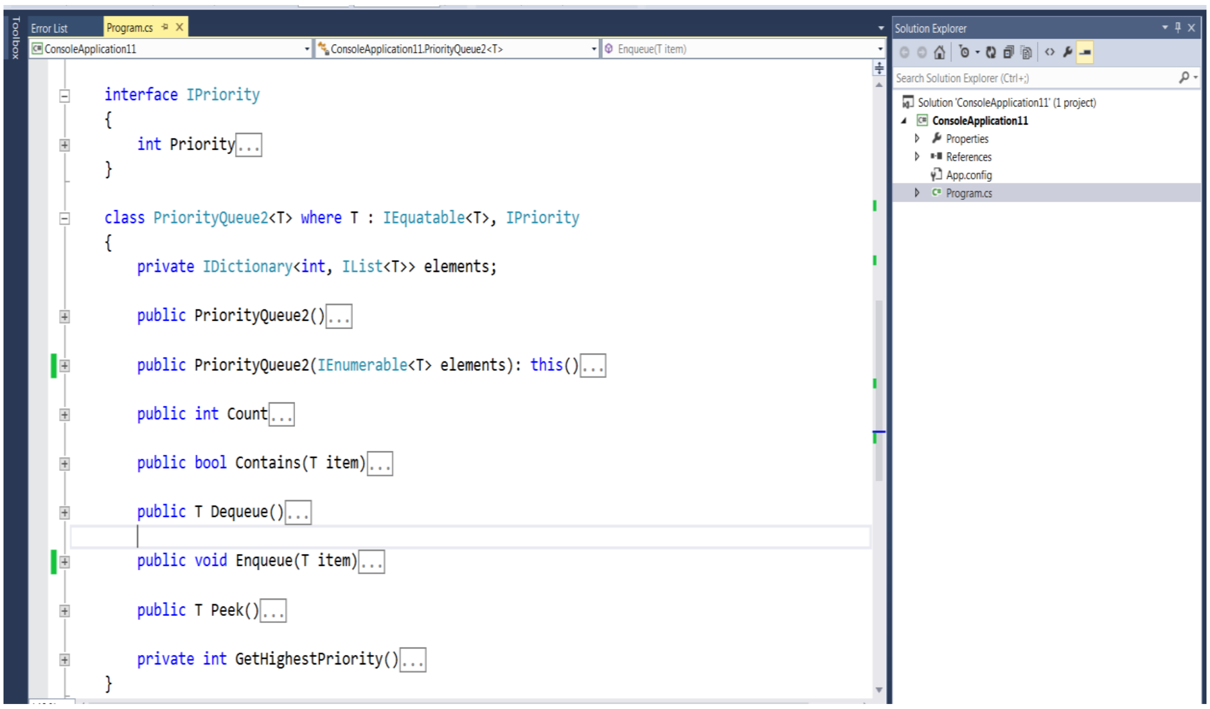
In this exercise, you'll implement a generic *PriorityQueue<T>* class that is parameterized by the type of object it contains. The priority queue will be used as a container of objects of type T. Implement Priority Queue in the following three ways. Create a small console application which provides a demo of various operations and its effect on the *PriorityQueue* object. There is no need for the application to be user interactive.

Exercise 8 Implementation 1 for Priority Queue



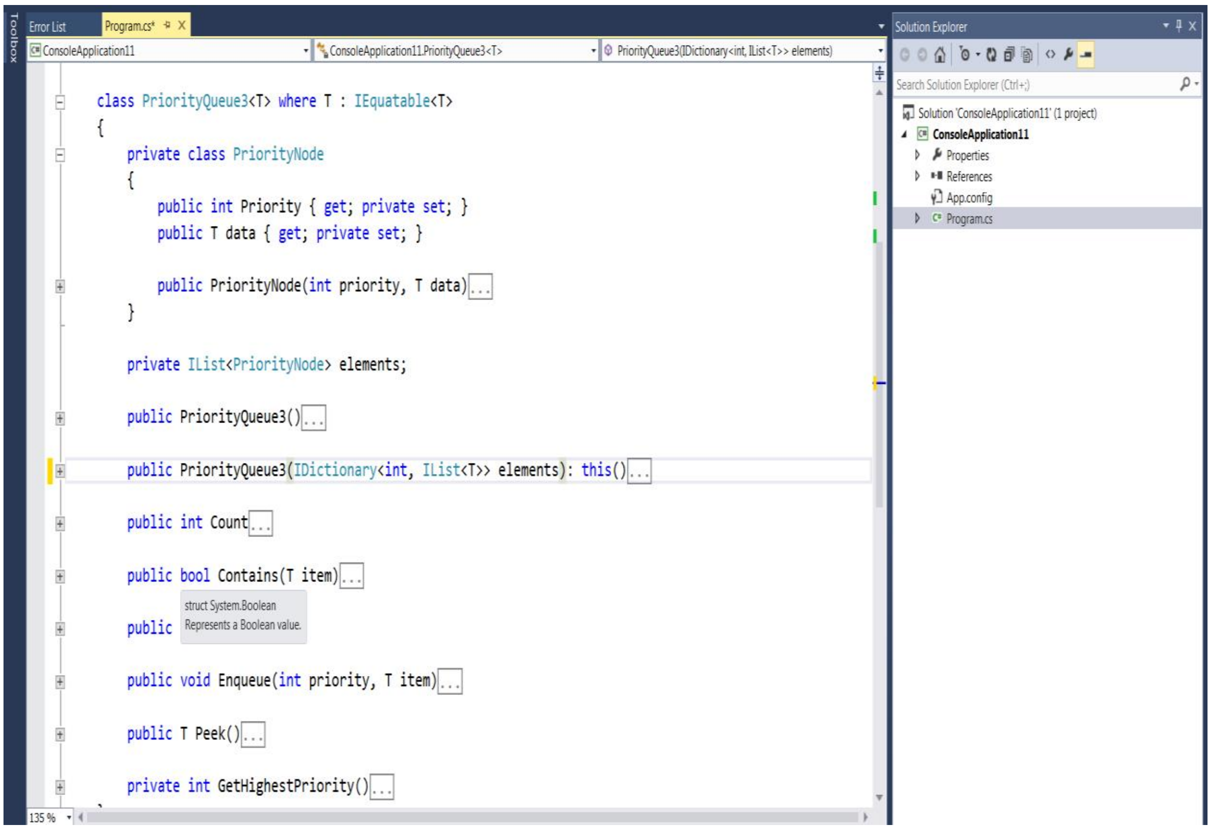
```
namespace ConsoleApplication11
{
    class Program...

    class PriorityQueue<T> where T: IEquatable<T>
    {
        private IDictionary<int, IList<T>> elements;

        public PriorityQueue()...

        public PriorityQueue(IDictionary<int, IList<T>> elements):this()...

        public int Count...

        public bool Contains(T item)...

        public T Dequeue()...

        public void Enqueue(int priority, T item)...

        public T Peek()...

        private int GetHighestPriority()...
    }
```

Exercise 9 Implementation 2 of Priority Queue



```
interface IPriority
{
    int Priority...
}

class PriorityQueue2<T> where T : IEquatable<T>, IPriority
{
    private IDictionary<int, IList<T>> elements;

    public PriorityQueue2()...

    public PriorityQueue2(IEnumerable<T> elements): this()...

    public int Count...

    public bool Contains(T item)...

    public T Dequeue()...

    public void Enqueue(T item)...

    public T Peek()...

    private int GetHighestPriority()...
}
```

Exercise 10 Implementation 3 of Priority Queue


Toolbox
Error List
Program.cs*
ConsoleApplication11
ConsoleApplication11.PriorityQueue3<T>
PriorityQueue3(IDictionary<int, IList<T>> elements)
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApplication11' (1 project)
ConsoleApplication11
Properties
References
App.config
Program.cs
struct System.Boolean
Represents a Boolean value.
135 %

```
class PriorityQueue3<T> where T : IEquatable<T>
{
    private class PriorityNode
    {
        public int Priority { get; private set; }
        public T data { get; private set; }

        public PriorityNode(int priority, T data)...
    }

    private IList<PriorityNode> elements;

    public PriorityQueue3()...

    public PriorityQueue3(IDictionary<int, IList<T>> elements): this()...

    public int Count...

    public bool Contains(T item)...

    public

    public void Enqueue(int priority, T item)...

    public T Peek()...

    private int GetHighestPriority()...
```

Extension methods and delegates

Exercise 11 Create the following extension methods on int

1. IsOdd
2. IsEven
3. IsPrime
4. IsDivisibleBy (taking one parameter excluding the first this parameter)

Exercise 12 Lambda and delegates

Create a list of integers and call where method on it and pass delegate to it in the following ways:

1. Find odd - Lambda Expression – without curly brackets
2. Find Even - Lambda Expression – with curly brackets
3. Find Prime – Anonymous Method
4. Find Prime Another – Lambda Expression
5. Find Elements Greater Than Five – Method Group Conversion Syntax
6. Find Less than Five – Delegate Object in Where – Method Group Conversion Syntax in Constructor of Object
7. Find 3k – Delegate Object in Where –Lambda Expression in Constructor of Object
8. Find 3k + 1 - Delegate Object in Where –Anonymous Method in Constructor of Object
9. Find 3k + 2 - Delegate Object in Where –Lambda Expression Assignment
10. Find anything - Delegate Object in Where – Anonymous Method Assignment
11. Find anything - Delegate Object in Where – Method Group Conversion Assignment

Exercise 13 Create extension methods on *IEnumerable<T>*. All of them taking a delegate

1. *CustomAll* - Should work as All operation of LINQ, with custom logic passed as delegate
2. *CustomAny* - Should work as Any operation of LINQ, with custom logic passed as delegate
3. *CustomMax* - Should work as Max operation of LINQ, with custom logic passed as delegate
4. *CustomMin* - Should work as Min operation of LINQ, with custom logic passed as delegate
5. *CustomWhere* - Should work as Where operation of LINQ, with custom logic passed as delegate
6. *CustomSelect* - Should work as Select operation of LINQ, with custom logic passed as delegate

Events

Exercise 14 Create a Product-Inventory software (No need of user Input or any other validations)

Product: (Override Equals or implement *IEquatable<T>*)

1. Id
2. Price
3. IsDefective

Inventory:

1. Dictionary containing all the products and their respective quantities in the inventory
2. Total value of the inventory

Methods

1. Add Product
2. Remove Product
3. Update Product Quantity

On change of Product's Price, Inventory total value should get updated. If a Product becomes defective, remove it from the inventory. (Handle them through *EventHandler*)

Exercise 15 Create an Observable Collection. Listen to its Collection Changed Event and display message on UI for each operation.

1. Addition: "Element 'x' is added in collection"
2. Removal: "Element 'x' is removed from collection"

Use *enum NotifyCollectionChangedAction* to differentiate the action

File Operations

Exercise 16 Get all the files from a given directory and perform the following actions

1. Return the number of text files in the directory (*.txt).
2. Return the number of files per extension type.
3. Return the top 5 largest files, along with their file size (use anonymous types).
4. Return the file with maximum length.

Exceptions

Exercise 17 You like teaching mathematics, and you are creating a small game for school kids. Game has multiple steps:

3. Display following message to user: “Enter any number from 1-5".
4. User enters an option from 1-5, show the exact message to user for the number selected.
    a. Enter even number.
    b. Enter odd number.
    c. Enter a prime number.
    d. Enter a negative number.
    e. Enter zero

for e.g., if user has selected 1, then show “Enter even number”,

If user does not enter correct number from 1-5 show error message. and then -> GOTO step 1. If user has entered correct number, then show success, else show error. -> after this GOTO step 1

To validate type of user input, there should be a validation method, which will return true if user input is correct, else validation method will throw different exception for different failure scenario.

Create a *CustomException* class. Also, when user has played this game for 5 times, show a message to user you have played this game for 5 times. Handle this also using exception.