Winter 2019 – 2nd In-semester Examination

IT602 – Object Oriented Programming

ID: ………………………………………………….. Name: …………………………………………….

Max Marks: 20 Time: 90 mins

Instructions:

1. All questions are compulsory.
2. Marks carried by each question are mentioned alongside the question.
3. The answers are to be written on the question paper itself in space provided below every question.
4. Supplementary sheet provided is only for rough work and should NOT be attached to the question paper.

Q1. (7)

A Photo Studio provides many photo services, like, converting a set of photos to gray scale album. The following code implements a class PhotoStudio which takes the order (through function PlaceOrder) where it takes set of photos. And, then it can convert the photo set into gray scale. See the following implementation:

```
enum PhotoType { Jpeg, Png };
class Jpeg { } /* represents a JPEG file */
class JpegFilter
{
    public Jpeg GrayScale(Jpeg jpeg) { /*convert to grayscale*/ return jpeg; }
}
class Png { } /* represents a PNG file */
class PngFilter
{
    public Png GrayScale(Png png) { /*convert to grayscale*/ return png; }
}

class PhotoStudio
{
    private JpegFilter jpegFilter;
    private PngFilter pngFilter;
    private List<Jpeg> jpegs;
    private List<Png> pngs;

    public void PlaceOrder(List<Jpeg> jpegPhotos, List<Png> pngPhotos, PhotoType
image)
    {
        switch(image)
        {
            case PhotoType.Jpeg:
                jpegFilter = new JpegFilter();
                jpegs = jpegPhotos;
                break;
            case PhotoType.Png:
                pngFilter = new PngFilter();
                pngs = pngPhotos;
                break;
        }
    }
    public List<Jpeg> GetGrayScaleAlbumForJpeg()
    {
        List<Jpeg> album = new List<Jpeg>();
        foreach (var photo in jpegs)
            album.Add(jpegFilter.GrayScale(photo));

        return album;
    }
    public List<Png> GetGrayScaleAlbumForPng()
    {
        List<Png> album = new List<Png>();
        foreach (var photo in pngs)
            album.Add(pngFilter.GrayScale(photo));

        return album;
    }
}
```

There are several problems with the above implementation. For example, both gray scale conversion functions do pretty much the same thing, other than working on different type of image format, violating the DRY principle. Also, whole PhotoStudio class gets affected if new type of photo (like bmp) is introduced.

Use guidance on SOLID principles and OOP concepts to come up with modified code that is a much better design, while achieving the same purpose – to convert a set of photos into gray scale set. In your answer **re-write complete code.** Goals of your solutions should be:

1. DRY principle is not violated i.e. there is only one function to process the album into gray scale for any photo type
2. If 3rd type of photo is introduced in the system, nothing in class PhotoStudio changes other than addition of switch case for the new type of photo in PlaceOrder function. (Don't introduce a 3rd type, only imagine and test where all your code changes)

/* your answer goes here */

Q2. (7)

Following code simulates a visa approval for an applicant at an embassy. No change to the code in class Applicant is allowed. In class Embassy also, no change is allowed other than filling in blank. You need to implement callback or pub/sub mechanism in the class VisaProcessor such that class Embassy can utilize it. In the VisaProcessor class that you author, you can assume that it approves the visa for every applicant when its ProcessVisa method is called – and it informs the registered entities about the status in this call itself via calling the callback function set by the entity (Embassy here).

Complete the code so that when it is run, the output is exactly the following:

```
Applicant ID: 1, Status: Approved
Applicant ID: 2, Status: Approved
```

```
    class Embassy
    {
        public static void Main(string[] args)
        {
            List<Applicant> applicants = new List<Applicant>()
            { new Applicant(1), new Applicant(2)};
            VisaProcessor visaProcessor = new VisaProcessor();
            visaProcessor.SetCallbackForVisaStatus(

/* write a lambda expression to provide the callback function to VisaProcessor */



            );

            foreach (var applicant in applicants)
                visaProcessor.ProcessVisa(applicant);
        }
    }

    class Applicant
    {
        public Applicant(int ID) { this.ID = ID; }
        public int ID { get; }
    }

    /* Write code for class VisaProcessor */
```

Q3. (2)

Fill in the blanks with the line number(s) where boxing and unboxing is happening in the code below.

```
1    class TestBoxing

2    {

3        static void Main()

4       {

5           int i = 123;

6           object o = i;

7           i = 456;

8           int j = (int)o;

9           System.Console.WriteLine(i);

10          System.Console.WriteLine(j);
        }
    }
```

Boxing:…………………………………………… Unboxing:…………………………………………………

Q4. (4)

Consider the code below. The comment in the implementation function can be thought of as the intended behavior of the object for that method. The comment /* no impl */ means that there is no easy way for the object to implement the method.
Which of the SOLID principles the following code primarily violates? And, re-write the code to show how will you fix it.

```
interface IAnimal
{
    void Breathe();
    void Walk();
    void Swim();
}


class Fish : IAnimal
{
    public void Breathe() {/* breathe */}
    public void Swim() { /* swim */}
    public void Walk() { /* no impl */}
}

class Bull : IAnimal
{
    public void Breathe() { /* breathe */ }
    public void Swim() { /* no impl */}
    public void Walk() { /* walk */}
}
```

```
/* your answer goes here */
```