Question 1:

Print out is 0

Reason: The SomeSubscriber instance has referenced SomePublisher instance. Even though SomeSubscriber instance is set to null, SomePublisher instance is not null, it is treated to be in use. So garbage collector will still treat the SomeSubscriber instances in use. Thus the ~SomeSubscriber will not be put into the garbage collection queue. Even when GC.collect() is executed, it will not call SomeSubscriber desctructor.

In order to make the printout to be 10, we can have 2 solutions below.

The second one is better than the first one :

Solution 1: right after for-loop and before GC.Collect, we add publisher = null in order to let Garbage collector know the publisher is unused.

Solution 2: we can use using statement for publisher and IDispose. So when using statement exits, it will call dispose method of SomePublisher method. In order to implement this, we need to change the code below

1). using statement for SomePublisher instance initialization

2). Class SomePublisher has to inherit from IDispose

3). add Dispose method in SomePublisher, in Dispose method, set the EventHanlder to be null

Notes on the solutions:

(1)The garbage collection is forced to run by using GC.Collect(). This will make destructor ~SomeSubscriber is called in the garbage collection queue.

(2). GC.WaitForPendingaFinalizers() is called. This will wait for all finializers to complete before continue to Console.WriteLine(...). Therefore, before the program goes to the last line(Console.WriteLine), ~Somesubscriber() will be called 10 times.

(3). SomeSubscriber.Count is a static variable, it is lively in the whole process so it gets increased by 10 times call to ~Somesbuscriber()

Question 2:

I used the factory design pattern to implement this CoinJar application. Code is below.

```
using System;

using System.Text;


namespace CoinJarApp

{

    class Program

    {

        static void Main(string[] args)

        {

            CoinJar coinJar = new USCoinJar();


            CoinFactory factory1 = new CentFactory();

            coinJar.AddCoin(factory1.GetCoin());


            CoinFactory factory2 = new NickleFactory();

            coinJar.AddCoin(factory2.GetCoin());


            CoinFactory factory3 = new DimeFactory();

            coinJar.AddCoin(factory3.GetCoin());


            CoinFactory factory4 = new QuarterFactory();

            coinJar.AddCoin(factory4.GetCoin());
            CoinFactory factory5 = new HalfDollarFactory();
            coinJar.AddCoin(factory5.GetCoin());

            CoinFactory factory6 = new DollarFactory();
            coinJar.AddCoin(factory6.GetCoin());

            //will not be added because it is not US coin
            bool ret = true;
            CoinFactory factory7 = new PennyFactory();
            ret = coinJar.AddCoin(factory7.GetCoin());
            if (!ret)
            {
                Console.WriteLine("unable to add the coin because it is not US coin");
            }

            Console.WriteLine("Coin Jar has coin amount:"+coinJar.Amount.ToString());
            Console.WriteLine("Coin Jar has coin volumn:" + coinJar.Volumn.ToString());
        }
    }


    #region enum
    enum CoinType
    {
```

```
    Cent = 1,

    Nickel = 2,

    Dime = 3,

    Quarter = 4,

    HalfQuarter = 5,

    Dollar = 6

}


enum CurrencyType

{

    USD = 1,

    CAD = 2

}

#endregion


#region coin class

abstract class Coin

{

    public abstract decimal Amount

    {

        get;

    }


    public abstract CurrencyType CType

    {
        get;
    }

    public abstract float Volumn
    {
        get;
    }
}

abstract class USCoin: Coin
{
    CurrencyType cType = CurrencyType.USD;

    public override CurrencyType CType
    {
        get { return cType; }
    }
}

class Cent : USCoin
{
    decimal amount = 0.01m;
    public override decimal Amount
    {
        get { return amount; }
    }

    float volumn = 0.0122f;
    public override float Volumn
    {
        get { return volumn; }
    }
}

class Nickle: USCoin
{
    decimal amount = 0.05m;
    public override decimal Amount
    {
        get { return amount; }
    }

    float volumn = 0.0243f;
    public override float Volumn
    {
        get { return volumn; }
    }
}

class Dime : USCoin
```

```
{
    decimal amount = 0.10m;
    public override decimal Amount
    {
        get { return amount; }
    }

    float volumn = 0.0115f;
    public override float Volumn
    {
        get { return volumn; }
    }
}

class Quarter : USCoin
{
    decimal amount = 0.25m;
    public override decimal Amount
    {
        get { return amount; }
    }

    float volumn = 0.027f;
    public override float Volumn
    {
        get { return volumn; }
    }

}

class HalfDollar : USCoin
{
    decimal amount = 0.50m;
    public override decimal Amount
    {
        get { return amount; }
    }

    float volumn = 0.0534f;
    public override float Volumn
    {
        get { return volumn; }
    }

}

class Dollar : USCoin
{
    decimal amount = 1.00m;
    public override decimal Amount
    {
        get { return amount; }
    }

    //I assume US dollar volumn is 0.07
    float volumn = 0.0700f;
    public override float Volumn
    {
        get { return volumn; }
    }

}

abstract class CACoin : Coin
{
    CurrencyType cType = CurrencyType.CAD;

    public override CurrencyType CType
    {
        get { return cType; }
    }
}

class Penny : CACoin
{
    decimal amount = 0.01m;
    public override decimal Amount
    {
        get { return amount; }
    }

    float volumn = 0.0122f;
    public override float Volumn
    {
        get { return volumn; }
    }
}
#endregion

#region CoinFactory
abstract class CoinFactory
{
    public abstract Coin GetCoin();
}

class CentFactory : CoinFactory
{
    public override Coin GetCoin()
    {
        return new Cent();
    }
}

class NickleFactory : CoinFactory
{
    public override Coin GetCoin()
    {
        return new Nickle();
    }
}

class DimeFactory : CoinFactory
{
    public override Coin GetCoin()
    {
        return new Dime();
    }
}

class QuarterFactory : CoinFactory
{
    public override Coin GetCoin()
    {
        return new Quarter();
    }
```

```
}

class HalfDollarFactory : CoinFactory
{
    public override Coin GetCoin()
    {
        return new HalfDollar();
    }
}

class DollarFactory : CoinFactory
{
    public override Coin GetCoin()
    {
        return new Dollar();
    }
}

class PennyFactory : CoinFactory
{
    public override Coin GetCoin()
    {
        return new Penny();
    }
}
```

```
#endregion


#region CoinJar

abstract class CoinJar

{

    public abstract void Reset();

    public abstract bool AddCoin(Coin coin);


    public abstract float Volumn

    {

        get;

    }


    public abstract decimal Amount

    {

        get;

    }

    public abstract float MaxVolumn

    {

        get;

    }

}


class USCoinJar : CoinJar

{
```

```
public override float MaxVolumn

{

   get {return 32;}

}


private float volumn;

public override float Volumn

{

   get { return volumn; }

}


private decimal amount;

public override decimal Amount

{

   get { return amount; }

}


public override void Reset()

{

   volumn = 0;

   amount = 0;

}


public override bool AddCoin(Coin coin)

{
```

```
            if (coin.CType != CurrencyType.USD)
            {
                return false;
            }

            float totalVolumn = Volumn + coin.Volumn;
            if (totalVolumn > MaxVolumn)
            {
                return false;
            }

            amount += coin.Amount;
            volumn = totalVolumn;

            return true;
        }

    }
    #endregion
}
```