Code 1

```
public interface ICalculatorReliableActor : IActor
{
    Task<double> Add(double x, double y);
    Task<double> Subtract(double x, double y);
    Task<double> Multiply(double x, double y);
    Task<double> Divide(double x, double y);
}
```

Code 2

```
public class CalculatorReliableActor : Actor, ICalculatorReliableActor
 {
    public async Task<double> Add(double x, double y)
    {
       ActorEventSource.Current.ActorMessage(this, "Adding");

       return await Task.FromResult(x + y);
    }
    public async Task<double> Subtract(double x, double y)
    {
       ActorEventSource.Current.ActorMessage(this, "Subtracting");

       return await Task.FromResult(x - y);
    }
    public async Task<double> Multiply(double x, double y)
    {
       ActorEventSource.Current.ActorMessage(this, "Multiplying");

       return await Task.FromResult(x * y);
    }
    public async Task<double> Divide(double x, double y)
    {
       ActorEventSource.Current.ActorMessage(this, "Dividing");

       return await Task.FromResult(x / y);
    }
 }
```

Code 3

```
    public class Program
    {
        public static void Main(string[] args)
        {
            var proxy = ActorProxy.Create<ICalculatorReliableActor>(ActorId.NewId(),
"fabric:/ServiceFabricReliableActor");

            Console.WriteLine("From Actor {0}: {1}", proxy.GetActorId(), proxy.Add(1, 1).Result);
            Console.WriteLine("From Actor {0}: {1}", proxy.GetActorId(), proxy.Subtract(0, 1).Result);
            Console.WriteLine("From Actor {0}: {1}", proxy.GetActorId(), proxy.Multiply(0.5f, 3f).Result);
            Console.WriteLine("From Actor {0}: {1}", proxy.GetActorId(), proxy.Divide(11f, 2f).Result);
            Console.WriteLine("Done");
            Console.ReadLine();
        }
    }
```

Code 4

[initially do it w/ public get/set, and a string for the op code]

```
public class CalculateMessage
{
    public enum Op
    {
        Add,
        Subtract,
        Multiply,
        Divide
    }

    public CalculateMessage(Op operation, double x, double y)
    {
        Operation = operation;
        X = x;
        Y = y;
    }

    public Op Operation { get; private set; }
    public double X { get; private set; }
    public double Y { get; private set; }
}
```

Code 5

```
public class CalculatorActor : UntypedActor
  {
    protected override void OnReceive(object message)
    {
      if (message as CalculateMessage == null)
      {
        throw new ArgumentException();
      }

      CalculateMessage msg = (CalculateMessage) message;

      Console.Write("From Actor {0}", this.);

      switch(msg.Operation)
      {
        case CalculateMessage.Op.Add:
          Console.WriteLine(msg.X + msg.Y);
          break;
        case CalculateMessage.Op.Subtract:
          Console.WriteLine(msg.X - msg.Y);
          break;
        case CalculateMessage.Op.Multiply:
          Console.WriteLine(msg.X * msg.Y);
          break;
        case CalculateMessage.Op.Divide:
          Console.WriteLine(msg.X / msg.Y);
          break;
        default:
          throw new ArgumentException();
      }
    }
  }
```

Code 6

```
public static ActorSystem MyActorSystem;

 static void Main(string[] args)
 {
     MyActorSystem = ActorSystem.Create("MySystem");

     var calculator = MyActorSystem.ActorOf<CalculatorActor>();

     var addMessage = new CalculateMessage(CalculateMessage.Op.Add, 1,1);
     var subMessage = new CalculateMessage(CalculateMessage.Op.Subtract, 0, 1);
     var divMessage = new CalculateMessage(CalculateMessage.Op.Divide, 11f, 2f);
     var mltMessage = new CalculateMessage(CalculateMessage.Op.Multiply, 0.5f, 3f);

     calculator.Tell(addMessage);
     calculator.Tell(subMessage);
     calculator.Tell(divMessage);
     calculator.Tell(mltMessage);

     Console.ReadLine();
 }
```