```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using O2DESNet;
using O2DESNet.Distributions;

namespace MMnQueue
{
    public class MMnQueue_Modular : State<MMnQueue_Modular.Statics>
    {
        #region Statics
        public class Statics : Scenario
        {
            /*******************************************************/
            /* All static properties shall be public,              */
            /* for both getter and setter.                         */
            /*******************************************************/
            public int ServerCapacity { get; set; }
            public double HourlyArrivalRate { get; set; }
            public double HourlyServiceRate { get; set; }
        }
        #endregion

        #region Dynamics
        /*********************************************************/
        /* All dynamic properties shall have only public getter,  */
        /* where setter should remain as private.                 */
        /*********************************************************/
        public Generator<int> Generator { get; private set; }
        public Queueing<int> Queue { get; private set; }
        public Server<int> Server { get; private set; }
        public int QueueLength { get { return Queue.Occupancy; } }
        public int NInService { get { return Server.Occupancy; } }
        public HourCounter HourCounter_QueueLength { get { return Queue.HourCounter; } }
        public HourCounter HourCounter_NInService { get { return Server.OccupationCounter; } }
        #endregion

        public MMnQueue_Modular(Statics config, int seed, string tag = null) : base(config, seed,
tag)
        {
            Name = "MMnQueue_Modular";

            Generator = new Generator<int>(new Generator<int>.Statics
            {
                Create = rs => Generator.Count,
                InterArrivalTime = rs => TimeSpan.FromHours(Exponential.Sample(rs, 1 /
Config.HourlyArrivalRate)),
                SkipFirst = false,
            },
            DefaultRS.Next());
            Generator.OnArrive.Add(i => Queue.Enqueue(i));

            Queue = new Queueing<int>();
            Queue.OnDequeue.Add(i => Server.Start(i));

            Server = new Server<int>(new Server<int>.Statics
            {
                ServiceTime = (i, rs) => TimeSpan.FromHours(Exponential.Sample(rs, 1 /
Config.HourlyServiceRate)),
                Capacity = Config.ServerCapacity,
            },
            DefaultRS.Next());
            Server.OnStateChg.Add(() => Queue.UpdToDequeue(Server.Vacancy > 0));

            InitEvents.Add(Generator.Start());
        }
```

```
        public override void WarmedUp(DateTime clockTime)
        {
            Generator.WarmedUp(clockTime);
            Queue.WarmedUp(clockTime);
            Server.WarmedUp(clockTime);
        }

        public override void WriteToConsole(DateTime? clockTime = null)
        {
            Console.WriteLine(clockTime);
            Console.WriteLine("Queue Length:\t{0}", QueueLength);
            Console.WriteLine("# in Serveice:\t{0}", NInService);
            Console.WriteLine("Avg. Q Length:\t{0:F2}", HourCounter_QueueLength.AverageCount);
            Console.WriteLine("Server Util.:\t{0:F2}", HourCounter_NInService.AverageCount /
Config.ServerCapacity);
        }
    }
}
```